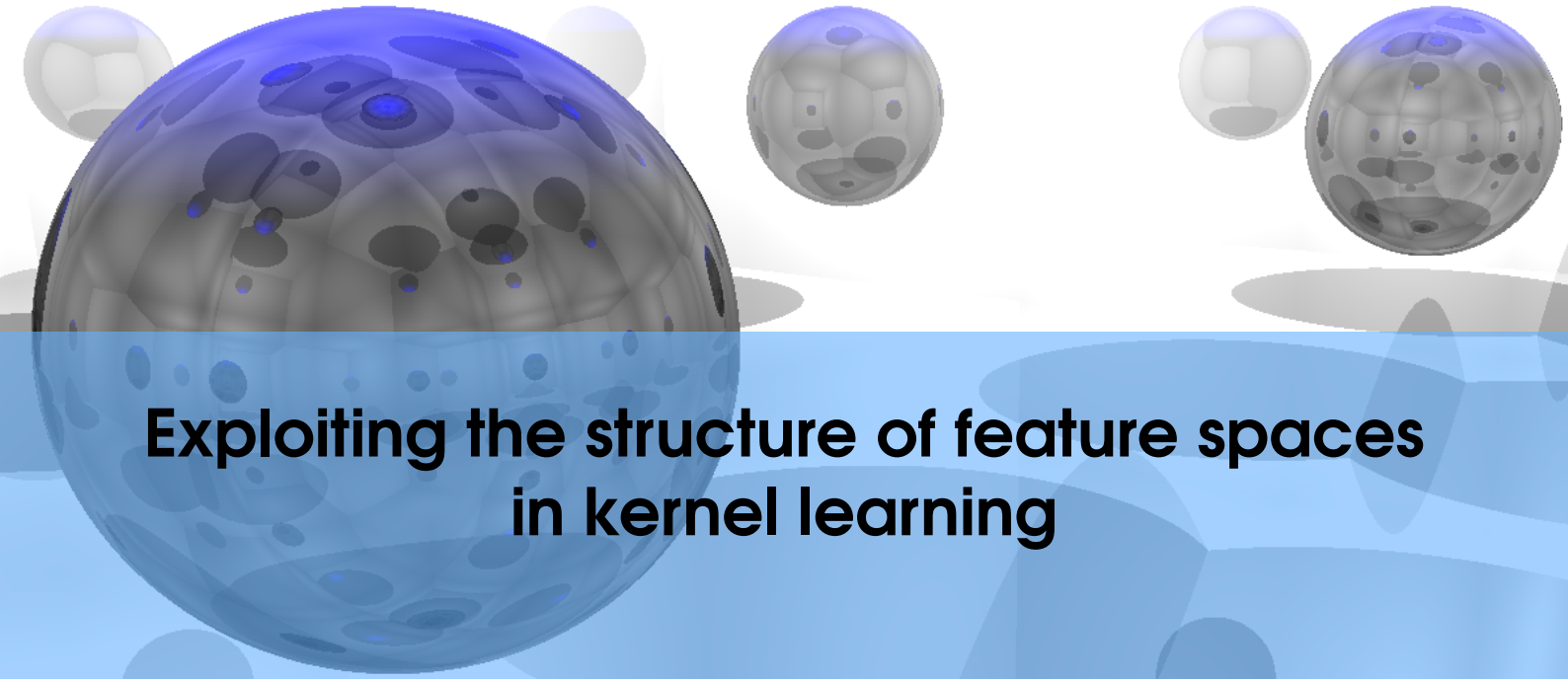


Exploiting the structure of feature spaces in kernel learning

PhD student: Michele Donini Advisor: Fabio Aiolli

Doctoral School in Mathematical Sciences

Computer Science area, XXVIII course

University of Padua, Italy

Year 2016

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Sede Amministrativa: Università degli Studi di Padova
Dipartimento di Matematica

SCUOLA DI DOTTORATO DI RICERCA IN: SCIENZE MATEMATICHE
INDIRIZZO: INFORMATICA
CICLO: XXVIII

EXPLOITING THE STRUCTURE OF FEATURE SPACES IN KERNEL LEARNING

Direttore della Scuola: Ch.mo Prof. Pierpaolo Soravia

Coordinatore d'indirizzo: Ch.mo Prof. Francesca Rossi

Supervisore: Ch.mo Prof. Fabio Aiolli

Dottorando: Michele Donini

Contents

Riassunto

Il problema dell'apprendimento della reppresentazione ottima per un task specifico è divenuto un importante argomento nella comunità dell'apprendimento automatico.

In questo campo, le architetture di tipo deep sono attualmente le più avanzate tra i possibili algoritmi di apprendimento automatico. Esse generano modelli che utilizzando alti gradi di astrazione e sono in grado di scoprire strutture complicate in dataset anche molto ampi. I kernel e le Deep Neural Network (DNN) sono i principali metodi per apprendere una rappresentazione di un problema in modo ricco (cioè deep).

Le DNN sfruttano il famoso algoritmo di back-propagation migliorando le prestazioni degli algoritmi allo stato dell'arte in diverse applicazioni reali, come per esempio il riconoscimento vocale, il riconoscimento di oggetti o l'elaborazione di segnali.

Tuttavia, gli algoritmi DNN hanno anche delle problematiche, ereditate dalle classiche reti neurali e derivanti dal fatto che esse non sono completamente comprese teoricamente. I problemi principali sono: la complessità della struttura della soluzione, la non chiara separazione tra la fase di apprendimento della rappresentazione ottimale e del modello, i lunghi tempi di training e la convergenza a soluzioni ottime solo localmente (a causa dei minimi locali e del vanishing gradient).

Per questi motivi, in questa tesi, proponiamo nuove idee per ottenere rapprensetazioni ottimali sfruttando la teoria dei kernel. I metodi kernel hanno un elegante framework che separa l'algoritmo di apprendimento dalla rappresentazione delle informazioni. D'altro canto, anche i kernel hanno alcune debolezze, per esempio essi non scalano e, per come sono solitamente utilizzati, portano con loro una rappresentazione poco ricca (shallow).

In questa tesi, proponiamo nuovi risultati teorici e nuovi algoritmi per cercare di risolvere questi problemi e rendere l'apprendimento dei kernel in grado di generare rappresentazioni più ricche (deeper) ed essere più scalabili.

Verrà quindi presentato un nuovo algoritmo in grado di combinare migliaia di kernel deboli con un basso costo computazionale e di memoria. Questa procedura, chiamata EasyMKL, supera i metodi attualmente allo stato dell'arte combinando frammenti di informazione e creando in questo modo il kernel ottimale per uno specifico task.

Perseguendo l'idea di creare una famiglia di kernel deboli ottimale, abbiamo creato una nuova misura di valutazione dell'espressività dei kernel, chiamata Spectral Complexity. Sfruttando questa misura siamo in grado di generare famiglia di kernel deboli con una struttura gerarchica nelle feature definendo una nuova proprietà riguardante la monotonicità della Spectral Complexity.

Mostriamo la qualità dei nostri kernel deboli sviluppando una nuova metologia per il Multiple Kernel Learning (MKL). In primo luogo, siamo in grado di creare una famiglia ottimale di kernel deboli sfruttando la proprietà di monotinicità della Spectral Complexity; combiniamo quindi la famiglia di kernel deboli ottimale sfruttando EasyMKL e ottenendo un nuovo kernel, specifico per il singolo task; infine, siamo in grado di generare un modello sfruttando il nuovo kernel e kernel machine (per esempio una SVM).

Inoltre, in questa tesi sottolineiamo le connessioni tra Distance Metric Learning, Feature Larning e Kernel Learning proponendo un metodo per apprendere la famiglia ottimale di kernel deboli per un algoritmo MKL in un contesto differente, in cui la regola di combinazione è il prodotto componente per componente delle matrici kernel. Questo algoritmo è in grado di generare i parametri ottimali per un kernel RBF anisotropico. Di conseguenza, si crea un naturale collegamento tra il Feature Weighting, le combinazioni dei kernel e l'apprendimento della metrica ottimale per il task.

Infine, l'importanza della rappresentazione è anche presa in considerazione in tre task reali, dove affrontiamo differenti problematiche, tra cui: il rumore nei dati, le applicazioni in tempo reale e le grandi moli di dati (Big Data).

Abstract

The problem of learning the optimal representation for a specific task recently became an important and not trivial topic in the machine learning community.

In this field, deep architectures are the current gold standard among the machine learning algorithms by generating models with several levels of abstraction discovering very complicated structures in large datasets. Kernels and Deep Neural Networks (DNNs) are the principal methods to handle the representation problem in a deep manner.

A DNN uses the famous back-propagation algorithm improving the state-of-the-art performance in several different real world applications, e.g. speech recognition, object detection and signal processing.

Nevertheless, DNN algorithms have some drawbacks, inherited from standard neural networks, since they are theoretically not well understood. The main problems are: the complex structure of the solution, the unclear decoupling between the representation learning phase and the model generation, long training time, and the convergence to a sub-optimal solution (because of local minima and vanishing gradient).

For these reasons, in this thesis, we propose new ideas to obtain an optimal representation by exploiting the kernels theory. Kernel methods have an elegant framework that decouples learning algorithms from data representations. On the other hand, kernels also have some weaknesses, for example they do not scale and they generally bring a shallow representation.

In this thesis, we propose new theory and algorithms to fill this gap and make kernel learning able to generate deeper representation and to be more scalable. An algorithm able

to combine thousands of *weak* kernels with low computational and memory complexities is proposed. This procedure, called EasyMKL, outperforms the state-of-the-art methods combining the fragmented information in order to create an optimal kernel for the given task.

Pursuing the idea to create an optimal family of *weak* kernels, we create a new measure for the evaluation of the kernel expressiveness, called spectral complexity. Exploiting this measure we are able to generate families of kernels with a hierarchical structure of the features by defining a new property concerning the monotonicity of the spectral complexity.

We prove the quality of these *weak* families of kernels developing a new methodology for the Multiple Kernel Learning (MKL). Firstly we are able to create an optimal family of *weak* kernels by using the monotonically spectral-complex property; then we combine the optimal family of kernels by exploiting EasyMKL, obtaining a new kernel that is specific for the task; finally, we are able to generate the model by using a kernel machine.

Moreover, we highlight the connection among distance metric learning, feature learning and kernel learning by proposing a method to learn the optimal family of *weak* kernels for a MKL algorithm in the different context in which the combination rule is the product element-wise of kernel matrices. This algorithm is able to generate the best parameters for an anisotropic RBF kernel and, therefore, a connection naturally appears among feature weighting, combinations of kernels and metric learning.

Finally, the importance of the representation is also taken into account in three tasks from real world problems where we tackle different issues such as noise data, real-time application and big data.

Introduction & Background

1. Introduction

Machine learning's purpose is to create algorithms able to optimize a performance criterion using examples from data or from past experience [8]. Given a model defined by some parameters, learning is the execution of a method to optimize them exploiting the training data. The model obtained should be able to make predictions in the future or to extract knowledge from data.

The theory of statistics is the most important tool exploited by machine learning in order to build mathematical models, since one of machine learning crucial tasks is making inference from a sample. Inference is the process of deriving logical conclusions from premises known, or assumed to be true (i.e. training examples). On the other hand, machine learning is more than only inference from a set of past experiences.

In fact, computer science is the second tool exploited by machine learning with two principal roles. Firstly, in training, where the efficiency of the algorithms is fundamental to solve the optimization problem and to store the solutions. Secondly, once we have a model, its representation and solution for inference needs to be efficient as well.

When we have to deal with a machine learning problem, the first step is to define a representation of the task, i.e. we have to select how to describe the known information. Typically, this pre-training step is performed to describe the problem that we have to solve in the best way possible by defining a set of features (explicitly or implicitly). In real world applications, the efficiency of the learning and the space and time complexity required by the methods have the same importance as to the predictive accuracy of the model.

In the following sections, the contextualization of the thesis will be presented, starting

from the introduction of the tackled problem.

1.1 The Representation Problem

The data representation plays a key role in the success of machine learning methods. Due to the current data growth in size, heterogeneity and structure, the new generation of algorithms are expected to solve increasingly challenging problems. This must be done under growing constraints such as computational resources, memory budget and energy consumption. The representation should ideally distill the relevant information about a learning problem in a compact manner, such that it becomes possible to learn the model from a small number of examples.

In the past, the research community has been focused on investigating new algorithms to obtain a model from a fixed a *priori* representation. In fact, the learning process was considered mainly the process of choosing an appropriate function from a given set of functions [188].

A new point of view is arising in this last decade, and the problem of learning the optimal representation has become a hot topic in the most important conferences and journals. When dealing with the representation of a task, a plethora of questions arise. Some questions are simple to be formulated but the answers are not easy to be found. The first questions are: how can we find automatically a good representation for a specific task? How can we compare two representations to assert that one is better than another?

The research community has addressed the representation of a task by injecting some reasonable priors in general-purpose fashion. Examples of these injected priors are several, the most commonly used has been described by Bengio et al. in [28]:

- **Smoothness**: the assumption that the model to be learned can be described by a *smooth* function f, i.e. if two examples $\mathbf{x}$ and $\mathbf{y}$ are similar ($\mathbf{x} \approx \mathbf{y}$) then the model has to produce similar outputs: $f(\mathbf{x}) \approx f(\mathbf{y})$;
- **Multiple explanatory factors**: the idea that a single factor can not describe the model and the underlying factors of variation have to be recovered (i.e the idea of the distributed representations);
- **Hierarchical organization of the explanatory factors**: the concept of a hierarchical organization of data with more *abstract* concepts higher in the hierarchy;
- **Shared factors across tasks**: the existence of shared factors among the tasks (e.g. Multitask learning, Transfer Learning);
- **Manifolds**: the assumption that a task can be represented using a manifold of smaller dimensionality than the original representation;
- **Sparsity**: the idea that only a small fraction of all the possible factors is really relevant.

Finally, the main assumption is that if we are able to use an optimal representation, all the explanatory factors can be related using simple (e.g. linear) models.

The most ingenuous way to take advantage of a good representation for a specific (not general-purpose) task is the simple feature engineering. An *expert* of the task has to create a set of features *ad hoc* in order to solve the problem at the best of his knowledge. This approach is time consuming, sub-optimal and it contains a strong bias derived from the personal experience of the expert. However, in some real world applications, where the task to tackle is well understood, this methodology is currently applied with good results [102, 167].

Problems exist where we are not able to exploit the knowledge of an expert, nor to create specific features for a single task. For example, when dealing with speech recognition our brain does operations that we are not able to describe and it is not possible to create features of phenomena that we do not understand in depth.

In this context, a new branch of algorithms became famous in the last years, called *deep learning* algorithms [125]. Deep learning generates models that are composed by a sequence of layers to learn representations of data with several levels of abstraction. The gold standard algorithms in the deep learning framework are represented by the Deep Neural Networks (DNNs). The DNNs are able to discover very complicated structures in large datasets by exploiting the back-propagation algorithm. These methods have changed the perspective in the machine learning community and improved the state-of-the-art in several different fields of application (e.g. speech recognition, visual object recognition, object detection and many others). Moreover, the deep convolutional neural networks have dramatically increased the performance of the machine learning algorithms when dealing with the processing of images, video, audio, text and speech (i.e. sequential or space-temporal data).

However, the biggest problem with deep learning is the very complex structure of the solution that imposes the utilization of the results as a sort of *black box*. Also, the convergence to the optimal solution is not theoretically guaranteed in these algorithms, and they suffer the same problems the standard *neural networks* [162, 67, 29, 135, 183, 209, 92, 146, 94, 93, 124]:

- They are theoretically not well understood;
- There is not a clear decoupling between the representation and the model generation;
- They have a high training time (more than 10 layers are difficult to be handled also using a non-common GPU cluster);
- They converge to a sub-optimal solution because of the local minima and the vanishing gradient issues;

For these reasons, new ideas are required to find deep solutions that do not suffer these

issues.

A possible solution is to exploit the more solid theoretical framework concerning kernels. In a general space $\mathcal{X}$, a kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a positive semi-definite function and represents a dot product in an implicitly defined Hilbert space X (a.k.a., feature space). K represents a similarity measure between the elements in $\mathcal{X}$. Given a kernel K, its feature mapping is a (typically non linear) embedding $\boldsymbol{\phi} : \mathcal{X} \to X$. The kernel K can be written in the form:

$$K(x,y) = \boldsymbol{\phi}(x) \cdot \boldsymbol{\phi}(y), \ \ \forall x,y \in \mathcal{X}. \tag{1.1}$$

So, given a kernel, the explicit evaluation of the vector $\boldsymbol{\phi}(x)$ is avoidable and we are able to obtain a significant improvement in the performance of the kernel methods.

In the past, kernels have consistently outperformed previous generations of learning techniques because they provided a flexible and expressive learning framework that has been successfully applied to a wide range of real world problems. For example, kernel methods are widely applied in machine learning for structured data because, unlike the majority of machine learning techniques, their application to any type of data is painless as long as a kernel function for such data is defined. Kernel methods offer an elegant framework that decouples learning algorithms from data representations. On the other hand, kernels have lost some of their initial appealing in the research community, due to some of their weaknesses:

- The scaling problem and the high computational complexity: dealing with kernel based methods imposes the storage in memory of a kernel matrix, i.e. a matrix with a number of entries quadratic with respect to the number of examples. So, kernel methods are not able (in general) to tackle a task when the number of examples becomes huge;
- The shallowness of the representation: the representation implicitly defined by a shallow kernel does not take into account several layer of abstraction and is fixed *a priori*.

Moreover, the so called local kernels suffer the *curse of dimensionality* [27], i.e. the problem of learning a model in a high dimensional space. This concept was coined by Bellman [25] and it refers to the exponential growth of hyper-volume as a function of dimensionality. Formally, we consider a local kernel K a kernel function with the following behavior:

$$\lim_{\|\mathbf{x}-\mathbf{x}_i\|_2 \to +\infty} K(\mathbf{x},\mathbf{x}_i) = C_i, \tag{1.2}$$

where $\mathbf{x}$ is a test example, $\mathbf{x}_i$ is a training example and C_i is a constant that does not depend on $\mathbf{x}$. For example, in the RBF kernel this constant equals 0. In the binary classification

problem, as consequence of this behavior of the local kernels, the models generated by a kernel machine collapse to constant models (i.e. classifiers with the same output for all the examples) or to nearest neighbor models (i.e. the predicted class of an example is defined only by the nearest neighbor example in the high-dimensional space). In both cases, the obtained models have poor prediction (constant or highly local). Moreover, when $\mathbf{x}$ is a high-dimensional vector, the nearest neighbor example is not much closer than the other examples, due to the geometrical properties of the high-dimensional spaces highlighted by Bellman.

New solutions have been recently presented with the aim to circumvent the difficulties concerning the scaling and the computational efficiency issues. For example, the random approximations of the kernels in order to avoid computational and memory issues are presented in [53, 174, 208]. These techniques consist in the approximation of the features of the Reproducing Kernel Hilbert Space (RKHS) to generate a linear kernel that approximates the original one. This new kernel can be used as a linear kernel in the original space and can be easily scaled-up. Using a linear kernel, the computational efficiency arises in a natural way exploiting the very efficient and distributed linear kernel machines (e.g. Pegasos [170]).

As pointed out before, the second critical issue about kernels is that they bring shallow and local information. In theory, kernels learn non-linear functionsϕ in the input space $\mathcal{X}$ and then they appear to be flexible as much as the DNN algorithms. However, traditionally, kernels were used to implement only a linear function in a pre-defined RKHS (the feature space). Starting from a fixed *a priori* kernel on a space $\mathcal{X}$, the entire learning phase is performed in a single step by exploiting the implicit representation $\boldsymbol{\phi}_{\mathbf{K}}(\mathbf{x})$ of the examples $\mathbf{x} \in \mathcal{X}$, as summarized in the following scheme:

$$\mathbf{x} \in \mathcal{X} \rightarrow \mathbf{K}_{fixed} \sim \boldsymbol{\phi}_{\mathbf{K}}(\mathbf{x}) \rightarrow \mathbf{w}_{learned} \cdot \boldsymbol{\phi}_{\mathbf{K}}(\mathbf{x}).$$

Learning the weight vector $\mathbf{w}$ implicitly can been seen as a single layer learning and it is in contrast with the deep learning paradigm. In practice, the $\boldsymbol{\phi}_{\mathbf{K}}$ is fixed *a priori* using the validation of the hyperparameters of the kernel (e.g. the parameter γ of the RBF kernel). This methodology is not the best way to proceed if we are interested in finding the optimal representation because the fixed kernel is typically not the optimal kernel for a specific task.

Learning a new implicit representation defined by a kernel is one the current challenges in the machine learning research community. A theoretical connection between kernels and deep networks has already been highlighted in the past [142] building a sequence of deeper and deeper kernels that reproduce the mapping performed by more and more layers of the deep network and measuring how these increasingly complex kernels fit the learning

problem.

Kernel learning has the goal to learn the optimal kernel (and then the best implicit feature map ϕ) given a specific task or set of tasks. Multiple Kernel Learning (MKL) [79] is one of the most popular approach to kernel learning. MKL algorithms are designed to combine a set of *weak* kernels to obtain a better one. MKL has the support of a theoretical framework [51] that claims a fundamental rule to overcome the shallowness of the single kernel representation. In particular, it has been proved that combining a large number of different kernels produces just a minor penalty in the generalization bounds. This result suggest that it is possible to combine thousands or millions of kernels without falling in the overfitting problem. A kernel can been seen as a different point of view of a task and then, the combination of millions of different point of views can be a solution to create a sort of *deeper* kernel, i.e. a kernel that is not shallow.

1.2 Aim and original contributions of this thesis

This thesis is focused on feature and kernel learning, aiming it improving the state-of-the-art on MKL and feature engineering in general. The original contribution of this thesis is four-fold:

1. **Analysis of the MKL framework** pursuing the idea of the *deeper* kernel. The problem of MKL is the computational complexity (in time and memory) and the creation of too shallow kernels. In Chapter 7, we propose a new algorithm able to combine thousands of kernels, called EasyMKL. A new scenario can be tackled in MKL where the *weak* kernels contain only a fragment of the information and the goal is to combine these fragments to obtain a *deeper* representation. Exploiting this new methodology, we obtain a MKL algorithm that outperforms both the strong average baseline and the state-of-the-art MKL methods. Finally, EasyMKL also seems quite robust with respect to the noise introduced by features which are not informative and works well even when exploiting the classical MKL framework, with a small number of *weak* kernels.
2. **Study of the importance of the expressiveness** of the family of *weak* kernels in order to inject a structure of expressiveness on them. MKL optimizes the margin among the examples and does not take into account the complexity of the combined kernels in the *weak* family. In this sense, a new technique is provided to measure the kernel complexity, called spectral complexity, in Chapter 8. This measure is easy to be evaluated and we exploit it to improve the MKL performance by creating optimal *weak* kernels families as a sort of pre-training of the data. Exploiting this measure, we define a new property of the *weak* families of kernels considering the monotonicity of the spectral complexity. This property imposes a hierarchical

structure among the features contained in different *weak* kernels. We show that using these families, EasyMKL improves its results in term of AUC and we apply this methodology in two contexts: dot-product kernels, namely the kernels in the form $K(\mathbf{x},\mathbf{z}) = f(\mathbf{x}\cdot\mathbf{z})$ and the ODD graph-kernel. In both cases, our approach improves the state-of-the-art results (see Chapter 9).

These results open a new scenario in MKL, that is beyond the combination of a large number of *weak* kernels. Performing the creation of the correct*weak* family we are adding a third layer in the learning phase. In fact, this methodology have three steps:

(a) The first step is to create a good (and possibly large) family of *weak* kernels starting from the raw information;
(b) Then, the second step is to combine all the *weak* kernels by exploiting MKL algorithms in order to fix the final representation;
(c) Finally, the last step is to create a model using a kernel machine.

Practically, the methodology can be summarized as:

$$\mathbf{x}\in\mathcal{X}\rightarrow\underbrace{\{\mathbf{K}_1,\ldots,\mathbf{K}_R\}_{learned}}_{a}\rightarrow\underbrace{\mathbf{K}_{learned}=\sum_{i=1}^{R}\mu_i\mathbf{K}_i\sim\boldsymbol{\phi}_{\mathbf{K}}(\mathbf{x})}_{b}\rightarrow\underbrace{\mathbf{w}_{learned}\cdot\boldsymbol{\phi}_{\mathbf{K}}(\mathbf{x})}_{c}.$$

It is important to note that the first step of this procedure ($\mathbf{x}\in\mathcal{X}\rightarrow\{\mathbf{K}_1,\ldots,\mathbf{K}_R\}_{learned}$) is performed by considering only the hierarchical structure of the features, implicitly defined in the RKHS.

3. **Connections** of kernel learning, metric learning, feature learning and MKL. Studying the representation problem from different point of views, some of these connections naturally arise, for example:
   - Learning a metric to create new kernels or to perform explicit feature weighting;
   - Using single feature kernels (i.e. $1-$rank or $1-$feature kernels) in a *weak* family to connect MKL to the Group Lasso feature learning.

   In Chapter 10, we study a new algorithm that is able to learn the hyperparameters of an Anisotropic RBF exploiting the connection among feature weighting, distance metric learning and MKL. This algorithm can also be seen as a MKL algorithm where the rule for the combination of the *weak* kernels is the entry-wise product of the *weak* matrices. We consider each *weak* kernel as a $1-$feature RBF (i.e. RBF kernel that considers only 1 feature) and we optimize the parameters of these family of RBFs. Exploiting this methodology, we are able to create an optimal family of *weak* kernels for this previously fixed MKL rule.
4. **Real world tasks**. The applicability of the new methods is widely important, not only on benchmark datasets. We prove our ideas in different real situations:

- In Chapter 11, a biomedical application will be presented, concerning the segmentation of brain MRI volumes. A new two-steps model is applied in order to exploit the domain knowledge in *a priori* manner and using this model the different underlying factors of variation for this task are automatically highlighted improving the previous state-of-the-art results.
- In Chapter 12, an application for smartphone will be presented, called *Climb The World*. The machine learning goal of this application is to count in real-time the number of stair-steps performed by a user by exploiting the sensors of the device. An extensive work of feature engineering and feature selection has been performed, using EasyMKL, to obtain optimal results under the constraints of battery consumption, imposed by the devices. The results achieved the goal of a very good real-time classification of the stairsteps and, at the best of our knowledge, this is the first algorithm that is able to solve this very difficult task.
- In Chapter 13, a new Multitask Learning (MTL) algorithm will be presented dealing with large scale datasets by using the alternating direction method of multipliers (ADMM). In the MTL context, this algorithm outperforms the previous state-of-the-art methods used in the classification of large scale dataset.

1.2.1 Structure of the thesis

This thesis is divided in four different parts. In Part I, the problem of representation in machine learning is presented. The principal notations are discussed in Chapter 2 and the state-of-the-art of the topics covered by this thesis are presented in Chapters 3, 4, 5 and 6.

In Part II, the original results of this thesis are depicted in four different chapters, concerning the creation of new algorithms to generate the optimal representation of a specific task:

- A new MKL algorithm is presented in Chapter 7. This algorithm is able to combine thousands of kernels using a fix amount of memory and with linear computation complexity and is called *EasyMKL*;
- A new measure for kernel complexity is defined in Chapter 8, called *spectral complexity*. This new measure has a strict connection to the *empirical Rademacher complexity*;
- In Chapter 9, a new methodology is applied to generate good families of *weak* kernels by exploiting the *spectral complexity* in two different cases. Firstly, in the space of the Dot-Product Polynomials (DPP) and then in the context of graph-kernels;
- In Chapter 10, a new algorithm is depicted that is able to learn the parameters of an Anisotropic Radial Basis Function (ARBF) kernel highlighting the connection

between metric learning, MKL and feature learning.

In Part III, three different real world applications are presented. In Chapter 11, a biomedical application about the segmentation of brain MRI volumes is presented. A new two-steps model is applied to exploit the domain knowledge in *a priori* manner.

In Chapter 12, an applications for smartphone is presented, called *Climb The World*. The learning task in this application was to count the number of stair-steps performed by a user by exploiting the sensors of the device. An extensive work of feature engineering and feature selection has been performed, using EasyMKL, to generate an optimal model under the constraints of battery consumption, imposed by the devices.

Finally, in Chapter 13, a new Multitask Learning (MTL) algorithm is presented that is able to deal with large scale datasets by using the Alternating Direction Method of Multipliers (ADMM).

In Part IV, conclusions and future work of this thesis are drawn.

1.3 Publications

A large part of this thesis has been presented in the following international peer-reviewed journals and conferences:

1.3.1 Journals

- (submitted) **Increasing People Physical Activity using a Mobile Application.** Fabio Aiolli, Matteo Ciman, Michele Donini and Ombretta Gaggi. *Pervasive and Mobile Computing 2015.*

- **EasyMKL: a scalable multiple kernel learning algorithm.** Fabio Aiolli and Michele Donini. *Neurocomputing 2015*.

1.3.2 Conferences

- (submitted) **Distributed Variance Regularized Multitask Learning.** Michele Donini, David Martinez-Rego and Massimiliano Pontil. *IJCNN 2016.*

- (submitted) **Learning the Kernel in the Space of Dot Product Polynomials.** Michele Donini and Fabio Aiolli. *ECML 2016.*

- **Advances in Learning with Kernels: Theory and Practice in a World of growing Constraints.** Luca Oneto, Nicolo Navarin, Michele Donini, Fabio Aiolli and

Davide Anguita. *ESANN 2016.*

- **Measuring the Expressivity of Graph Kernels through the Rademacher Complexity.** Luca Oneto, Nicolo Navarin, Michele Donini, Sperduti Alessandro, Fabio Aiolli and Davide Anguita. *ESANN 2016.*

- **Multiple Graph-Kernel Learning.** Fabio Aiolli, Michele Donini, Nicoló Navarin and Alessandro Sperduti. *SSCI 2015.*

- **Feature and kernel learning.** Verónica Bolón-Canedo, Michele Donini and Fabio Aiolli. *ESANN 2015.*

- **ClimbTheWorld: Real-time stairstep counting to increase physical activity.** Fabio Aiolli, Matteo Ciman, Michele Donini and Ombretta Gaggi. *MOBIQUITOUS 2014.*

- **Learning Anisotropic RBF Kernels.** Fabio Aiolli and Michele Donini. *ICANN 2014.*

- **Easy multiple kernel learning.** Fabio Aiolli and Michele Donini. *ESANN 2014.*

- **A Serious Game to persuade people to use stairs.** Fabio Aiolli, Matteo Ciman, Michele Donini and Ombretta Gaggi. *Persuasive 2014.*

- **Stacking Models for Efficient Annotation of Brain Tissues in MR Volumes.** Fabio Aiolli, Michele Donini, Enea Poletti and Enrico Grisan. *Medicon 2013.*

2. Notation

In this chapter, we will introduce the main notations used in this thesis. The principal mathematical notations exploited in this thesis are presented in Table 2.1. In Section 2.1, the notations concerning the binary classification problem are depicted. Finally, in Section 2.2, the notations about graphs are introduced.

2.1 Binary Classification Problem

Considering the classification task, we define the training examples as $\{(x_i, y_i)\}_{i=1}^{l}$, and test examples as $\{(x_i, y_i)\}_{i=l+1}^{L}$, x_i in a generic set $\mathcal{X}$, y_i with values $+1$ or -1. The matrix $\mathbf{K} \in \mathbb{R}^{L \times L}$ is the complete kernel matrix containing the values of the kernel of each (training and test) data pair. Further, we indicate with an hat, like for example $\hat{\mathbf{y}} \in \mathbb{R}^{l}$ or $\hat{\mathbf{K}} \in \mathbb{R}^{l \times l}$, the submatrices (or subvectors) obtained considering training examples only.

Fixed a training set, $\hat{\Gamma}$ will denote the domain of probability distributions $\boldsymbol{\gamma} \in \mathbb{R}_{+}^{l}$ defined over the sets of positive and negative training examples:

$$\hat{\Gamma} = \{\boldsymbol{\gamma} \in \mathbb{R}_{+}^{l} \mid \sum_{i \in \oplus} \gamma_i = 1, \sum_{i \in \ominus} \gamma_i = 1\}, \tag{2.1}$$

where $\oplus$ (resp. $\ominus$) is the set of the indices of the positive examples (resp. negative examples). Note that any element $\boldsymbol{\gamma}$ of the set $\hat{\Gamma}$ corresponds to a pair of points, the first in the convex hull of positive training examples $\mathbf{x}_{+} = \sum_{j \in \oplus} \gamma_j \mathbf{x}_j$, and the second in the convex hull of negative training examples $\mathbf{x}_{-} = \sum_{j \in \ominus} \gamma_j \mathbf{x}_j$.

Symbol	
$\mathbf{M}$	The bold capital letter are used to identify the matrices
$\mathbf{M}[i,j]$	The MatLab notation identifies the element in position i,j of the matrix $\mathbf{M}$
$\mathbf{M}[:,j]$	The column j of the matrix $\mathbf{M}$
$\mathbf{M}[i,:]$	The row i of the matrix $\mathbf{M}$
$\mathbf{v}$	The bold lower case letter are used to represent vectors
$\|\cdot\|_1$	The 1-norm of a matrix or a vector
$\|\cdot\|_2$	The 2-norm of a matrix or a vector
$\|\cdot\|_F$	The Frobenious norm of a matrix
$\|\cdot\|_T$	The Trace norm of a matrix
$eig(\mathbf{M})$	The set of the eigenvalues of a matrix
$\mathbf{1}_L$	A vector in $\mathbb{R}^L$ where all the elements are equal to 1
$\mathbf{I}_L$	The identity matrix in $\mathbb{R}^{L\times L}$
$\sqrt{\mathbf{D}}$	The element-wise squared root of a matrix $\mathbf{D}$
$\mathbb{E}$	The average of a variable
Var	The variance of a variable
$sign(\cdot)$	A function $\mathbb{R}\rightarrow\{-1,1\}$ that returns 1 if the argument is positive, -1 otherwise
$\bigotimes$	The element-wise product of vectors or matrices
$\sum$	The element-wise summation of scalars, vectors or matrices
$\prod$	The Capital Pi notation for a product of sequences of scalars, vectors or matrices
$\mathcal{O}$	The big O notation
$a\propto$ b	The two quantities $a,b\in\mathbb{R}$ are proportional

Table 2.1: Mathematical symbols.

R Usually, it is possible to consider the generic set $\mathcal{X}$ equal to $\mathbb{R}^m$. Then, $\mathbf{X} \in \mathbb{R}^{L \times m}$ denotes the matrix where examples are arranged in rows and $\hat{\mathbf{X}} \in \mathbb{R}^{l \times m}$ is the sub-matrix obtained considering training examples only. In this case, the i^{th} example is represented by the i^{th} row of $\mathbf{X}$, namely $\mathbf{X}[i,:]$.

2.1.1 Quality measures for binary classifications

Dealing with classification task, we are interested in finding which method has the best performance. The choice of an appropriate measure is very important to evaluate correctly the quality of a specific algorithm. The most important measures of quality are: accuracy, precision, recall, F_β and AUC[1].

Accuracy

The accuracy of a model is the fraction of examples that are correctly classified using that model. It is a widely used metric for the performance of a classification algorithm. Despite the popularity of this measure, the accuracy suffers of some critical drawbacks:

- It assumes equal cost for both false positive and false negative errors;
- The *base rate* problem: in the case of unbalanced dataset, a classifier that predicts constantly the predominant class could have a good accuracy;
- It does not consider the percent reduction in error (i.e. an improvement in accuracy of 0.1 from 0.8 to 0.9 is a reduction in error of 50%, on the other hand, an increase of accuracy of 0.01 from 0.99 to 1.0 is a reduction in error of 100%).

Precision, Recall and F_β

In a classification task, the precision for a class is the number of true positives (i.e. the number of items correctly labeled as belonging to the positive class) divided by the total number of elements labeled as belonging to the positive class (i.e. the sum of true positives and false positives, which are items incorrectly labeled as belonging to the class). Recall is defined as the number of true positives divided by the total number of elements that actually belong to the positive class (i.e. the sum of true positives and false negatives, which are items which were not labeled as belonging to the positive class but should have been).

The analytic formulas for these measures are:

$$\text{precision} = \frac{TP}{TP+FP}$$
$$\text{recall} = \frac{TP}{TP+FN}$$

where TP stands for True Positives, FP as False Positives and FN as False Negatives.

[1] http://www.cs.cornell.edu/courses/cs578/2003fa/performance_measures.pdf

F_β is a measure that consider a combination of precision and recall. This combination is weighted by a parameter $\beta \in \mathbb{R}$. The analytic formula of F_β is:

$$F_\beta = (1+\beta^2)\frac{precision \cdot recall}{\beta^2 precision + recall}.$$

AUC

Solving a classification problem, the large part of the algorithms learn a model that is able to return a score value for each example. The union of all these scores can be seen as a ranking of the examples. The goal of a good model is to force the positive examples at the highest positions of the ranking. The Area Under Curve (AUC) is a metric that exploits the ranking given by the model.

Firstly, we have to define the Receiver Operating Characteristic (ROC) curve. The ROC curve is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. The curve is created by plotting the true positive rate against the false positive rate at various threshold settings. The true-positive rate is also known as sensitivity or recall. The false-positive rate is also known as the $fall-out$ and can be calculated as: 1−specificity. The ROC curve is thus the sensitivity as a function of fall-out. The AUC is the (estimated) integral of the ROC Curve.

Given the prediction scores of m examples, $\mathbf{r} \in \mathbb{R}^m$, we can define analytically the *Receiver Operating Characteristic Area Under Curve* metric (AUROC or AUC) as:

$$\text{AUC}(\mathbf{r}) = \frac{\sum_{i=1}^{m} \psi(\mathbf{x}_i)}{2\, n_p\, n_n} \in [0,1], \tag{2.2}$$

where we define n_p and n_n as the number of positive and negative examples, and the function $\psi(\mathbf{x}_i)$ as:

$$\psi(\mathbf{x}_i) = |\{\mathbf{x}_j : y_j \neq y_i, r_j < r_i, j = 1,\ldots,m\}| \text{ if } y_i = 1, \tag{2.3}$$

$$\psi(\mathbf{x}_i) = |\{\mathbf{x}_j : y_j \neq y_i, r_j > r_i, j = 1,\ldots,m\}| \text{ if } y_i = -1. \tag{2.4}$$

The AUC represents the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one (assuming positive ranks higher than negative).

2.1.2 Datasets for Binary Vectorial Classification

In this section, we will present the 15 datasets used as benchmark in this thesis. These datasets are different for typology of the information, number and typology of the features and number of examples.

The following list contains a brief description of the benchmark datasets:

- **Haberman**: The dataset contains cases from a study that was conducted between 1958 and 1970 at the University of Chicago's Billings Hospital on the survival of patients who had undergone surgery for breast cancer.
- **Liver**: This dataset is about BUPA liver disorders. The first 5 variables are all blood tests which are thought to be sensitive to liver disorders that might arise from excessive alcohol consumption.
- **Diabetes**: Diabetes patient records were obtained from two sources: an automatic electronic recording device and paper records. The automatic device had an internal clock to timestamp events, whereas the paper records only provided "logical time" slots (breakfast, lunch, dinner, bedtime). This dataset contain has been prepared for the use of participants for the 1994 AAAI Spring Symposium on Artificial Intelligence in Medicine.
- **Abalone**: Predicting the age of abalone from physical measurements. The age of abalone is determined by cutting the shell through the cone, staining it, and counting the number of rings through a microscope. Other measurements, which are easier to obtain, are used to predict the age. From the original data examples with missing values were removed.
- **Australian**: This file concerns credit card applications. All attribute names and values have been changed to meaningless symbols to protect confidentiality of the data. This dataset is interesting because there is a good mix of attributes: continuous, nominal with small numbers of values, and nominal with larger numbers of values.
- **Pendigits**: A digit database by collecting 250 samples from 44 writers. The authors used a WACOM PL-100V pressure sensitive tablet with an integrated LCD display and a cordless stylus. The tablet sends x and y tablet coordinates and pressure level values of the pen at fixed time intervals (sampling rate) of 100 milliseconds. These writers are asked to write 250 digits in random order inside boxes of 500 by 500 tablet pixel resolution. This dataset has been binarized for the classification problem of *even vs. odd*.
- **Heart**: This dataset is a heart disease database.
- **German**: This dataset classifies people described by a set of attributes as good or bad credit risks.
- **Ionosphere**: This dataset consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. The targets were free electrons in the ionosphere. Received signals were processed using an autocorrelation function whose arguments are the time of a pulse and the pulse number. There were 17 pulse numbers for the Goose Bay system. Instances in this database are described by 2 attributes per pulse number, corresponding to the complex values

returned by the function resulting from the complex electromagnetic signal.

- **Splice**: Splice junctions are points on a DNA sequence at which superfluous DNA is removed during the process of protein creation in higher organisms. The problem posed in this dataset is to recognize, given a sequence of DNA, the boundaries between exons (the parts of the DNA sequence retained after splicing) and introns (the parts of the DNA sequence that are spliced out).
- **Sonar**: The task is to train a network to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock.
- **Mush**: This dataset includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family. Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This dataset has been binarized.
- **Batch2**: This binarized dataset contains 1244 measurements from 16 chemical sensors exposed to 6 gases at different concentration levels. This dataset has been binarized.
- **Colon**: This dataset contains expression levels of 2000 genes taken in 62 different samples. For each sample it is indicated whether it came from a tumor biopsy or not.
- **Gisette**: This dataset is about a handwritten digit recognition problem. The problem is to separate the highly confusable digits '4' and '9'. This dataset is one of five datasets of the NIPS 2003 feature selection challenge. A truncated version of this dataset, called **GisetteT**, is also used with a random sampling of 4000 examples from the original Gisette dataset.

Finally, in Table 2.2, the principal information about the datasets are summarized and in Table 2.3, the links to download the datasets are provided.

2.2 Graphs

In this section we summarize the principal notations about graphs used in this thesis (in Section 9.5) . Firstly, we can introduce a formal definition of a graph G:

Definition 2.2.1 A graph G can be descried as triplet $G = (V_G, E_G, L_G)$ where V_G is the set of vertices, E_G the set of edges and L_G a function mapping nodes to labels.

Now, we are ready to define a list of the most used definitions and properties concerning graphs:

- A graph is **undirected** if $(v_i, v_j) \in E_G \Leftrightarrow (v_j, v_i) \in E_G$, otherwise it is **directed**.
- A **path** $p(v_i, v_j)$ of length n in a graph G is a sequence of nodes $v_1, \ldots, v_n$, where $v_1 = v_i$, $v_n = v_j$ and $(v_k, v_{k+1}) \in E_G$ for $1 \leq k < n$.
- A **cycle** is a path for which $v_1 = v_n$. A graph is acyclic if it has no cycles.

Dataset	Source	Features	Examples
Haberman	*UCI* [18]	3	306
Liver	*UCI*	6	345
Diabetes	*UCI*	8	768
Abalone	*UCI*	8	4177
Australian	*Statlog*	14	690
Pendigits	*UCI*	16	4000
Heart	*UCI*	22	267
German	*Statlog*	24	1000
Ionosphere	*UCI*	34	351
Splice	*UCI*	60	1000
Sonar	*UCI*	60	208
Mush	*UCI*	112	2000
Batch2 [190]	*UCI*	128	1244
Colon	*UCI*	2000	62
GisetteT	*NIPS*03	5000	4000
Gisette [82]	*NIPS*03	5000	13500

Table 2.2: Datasets information: name, source, number of features and number of examples.

Dataset	URL
Haberman	`archive.ics.uci.edu/ml/datasets/Haberman's+Survival`
Liver	`archive.ics.uci.edu/ml/datasets/Liver+Disorders`
Diabetes	`archive.ics.uci.edu/ml/datasets/Diabetes`
Abalone	`archive.ics.uci.edu/ml/datasets/Abalone`
Australian	`www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/australian`
Pendigits	`archive.ics.uci.edu/ml/machine-learning-databases/pendigits/`
Heart	`archive.ics.uci.edu/ml/datasets/Statlog+(Heart)`
German	`archive.ics.uci.edu/ml/datasets/Statlog+(German+Credit+Data)`
Ionosphere	`archive.ics.uci.edu/ml/datasets/Ionosphere`
Splice	`www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/splice_scale`
Sonar	`www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/sonar_scale`
Mush	`archive.ics.uci.edu/ml/datasets/Mushroom`
Batch2	`archive.ics.uci.edu/ml/machine-learning-databases/00270/`
Colon	`www.inf.ed.ac.uk/teaching/courses/dme/html/datasets0405.html`
Gisette	`archive.ics.uci.edu/ml/datasets/Gisette`

Table 2.3: URLs of the datasets.

- A **Directed Acyclic Graph** (DAG) is a tree where each node has at most one incoming edge.
- The **root** of a tree T is represented by $r(T)$.
- The **children** of a node $v \in V_T$ are all the nodes v' s.t. $(v, v') \in E_T$. $ch_v[j]$ refers to the j-th child of v.
- The maximum number of children in a tree is called ρ.
- A **proper subtree** rooted at node v comprises v and all its descendants.

2.2.1 Datasets

In this thesis, we compared different methods for the classification of structured data using five real-world graph datasets from bioinformatics: CAS[2], CPDB [91], AIDS [198], NCI1 [194] and GDD [63]. The first four datasets represent chemical compounds: nodes represent atoms and are labeled according to the atom type, and edges represent bonds between atoms. GDD is a dataset of proteins. In this case, each node in a graph represents an amino acid and is labeled according to the amino acid type. In GDD, there is an edge connecting two nodes if the corresponding amino acids are less than 6 apart in the 3-dimensional folding of the protein. All the datasets encode binary classification problems.

[2] http://www.cheminformatics.org/datasets/bursi

3. Topics

Finding the best representation of the data has been an active research area in the last few decades obtaining success in many different applications. For example, with the advent of big data, the adequate identification of the relevant information has converted representation learning in an even more indispensable step. An important part of this thesis is concerning the representation given by feature and kernel learning.

In kernel methods, features are implicitly represented by means of feature mappings and kernels. It has been shown that the correct selection of the kernel is a crucial task, as long as an erroneous selection can lead to poor performance. Unfortunately, manually searching for an optimal kernel is a time-consuming and a sub-optimal choice.

This chapter is concerned with the background of the methods exploited in this thesis to learn automatically features, kernels and new representations from the data. Firstly, we can introduce a taxonomy of these methods in two families: *Explicit Feature Learning* (EFL) and *Implicit Feature Learning* (IFL).

3.1 Explicit and Implicit Feature Learning

It is possible to divide the representation learning methods in two different families. In the first family are contained the methods where the features are learned explicitly in the original space. We call the methods in this family the *Explicit Feature Learning* (EFL) algorithms. EFL methods exploit the combination of the information contained in the data in an explicit way. They highlight clearly which features is more important than the others, and due to this particular behavior, the EFL methods are useful to identify the motivation

behind the importance of the information. In real wold application, EFL methods have a widely success because of the simple interpretation of the results.

On the other hand, there are a plethora of methods that are, indirectly, able to exploit the information in the RKHS (or in a different non linear feature space). We can call this second family of methods *Implicit Feature Learning* (IFL). This family contains methods that are more complex and sophisticated than the EFL methods. IFL becomes popular only few years ago and nowadays the research community is very active on this topic.

In the following sections, the topics covered by this thesis will be presented, starting from a brief overview of kernel learning, feature learning and metric learning. Finally, a deep analysis of the EFL and IFL families will be depicted, concerning the principal methods exploited in Part II and Part III.

3.1.1 Kernel Learning

Kernel machines and kernel-based algorithms are very popular in machine learning and have shown state-of-the-art performance. Kernel methods have been used to tackle a variety of learning tasks (e.g. classification, regression, clustering and more). In these algorithms the features are provided intrinsically using a positive semi-definite kernel function that can be interpreted as a similarity measure (i.e. a scalar product) in a high dimensional Hilbert space. The goal of Kernel Learning (KL) is to create methods able to provide automatically good kernels for a particular problem and avoiding, in this way, the arbitrary user's choice of a kernel function. A bad choice of the kernel function can make learning very difficult because it corresponds to a sub-optimal choice of the features.

KL is often adopted within a semi-supervised learning setting [44, 212, 50, 130] and tries to learn the kernel matrix using all the available data (labeled and unlabeled examples) optimizing an objective function that improves the accordance between the kernel and the set of i.i.d. labeled data. Different ways have been investigated to obtain this result, e.g. maximizing the alignment [123, 52] or exploiting bounds on the *(local) Rademacher complexity* [48]. Conversely, unlabeled data are typically used to regularize the models avoiding the non-smoothness of the discriminant function.

Evaluation of the KL methods

The performance evaluation of a KL algorithm applied to a binary classification task can be computed by exploiting one of the several measures of quality, e.g, accuracy, precision, recall, F_β and more. In this thesis, the experimental evaluation of a kernel will be performed by using the *Receiver Operating Characteristic Area Under Curve* metric (AUC). The AUC represents the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one (assuming positive ranks higher than negative). We have chosen the AUC measure because we are able to avoid the

validation of the threshold parameter. This parameter is influenced by the kernel machine (i.e. the classifier) and not by (the quality) of the kernel.

3.1.2 Metric Learning

It is well known that different features typically have unequal impact and importance in solving a given classification task. This motivates several feature selection methods to select or weight different features in different ways. While feature selection is generally very difficult to perform with nonlinear kernels, one can learn the metric directly from data more easily [24]. This task is known as Distance Metric Learning (DML). For example, many researchers (see [75], [76], [169], [64]) have proposed a number of algorithms for the optimization of the Mahalanobis distance. Specifically, they replace the common Euclidean metric with the more powerful distance

$$(\mathbf{x}_i - \mathbf{x}_j)^\top \mathbf{M} (\mathbf{x}_i - \mathbf{x}_j) \tag{3.1}$$

and try to learn the combination matrix $\mathbf{M}$. The learned distance is typically optimized for (and used in) a nearest neighbors setting. The high number of free parameters to learn together with the fact that these methods are used with nearest neighbors, make these approaches prone to overfitting, in particular when the training sample is small.

3.1.3 Feature Learning

In the last few years, several datasets with high dimensionality have become publicly available on the Internet. This fact has brought an interesting challenge to the research community, since for the machine learning methods it is difficult to deal with a high number of input features. To cope with the problem of the high number of input features, dimensionality reduction techniques can be applied to reduce the dimensionality of the original data and improve learning performance. These dimensionality reduction techniques usually come in two flavors: *feature selection* and *feature extraction*.

Feature selection and feature extraction each have their own merits [211]. On the one hand, feature extraction techniques achieve dimensionality reduction by combining the original features. In this manner, they are able to generate a set of new features, which is usually more compact and of stronger discriminating power. It is preferable in applications such as image analysis, signal processing, and information retrieval, where model accuracy is more important than model interpretability. On the other hand, feature selection achieves dimensionality reduction by removing the irrelevant and redundant features. It is widely used in data mining applications, such as text mining, genetics analysis, and sensor data processing. Due to the fact that feature selection maintains the original features, it is especially useful for applications where the original features are important for model

interpretation and knowledge extraction.

Feature selection methods are usually divided into three major approaches based upon the relationship between a feature selection algorithm and the inductive learning method used to infer a model [81]. *Filters* rely on the general characteristics of training data and carry out the feature selection process as a pre-processing step independently from the induction algorithm. On the contrary, *wrappers* involve optimizing a predictor as a part of the selection process. In between them one can find *embedded* methods, which perform feature selection in the process of training and are usually specific to given learning machines. Popular and widely-used feature selection methods are Correlation-based Feature Selection (CFS) [86], minimum Redundancy Maximum Relevance (mRMR) [150], or Support Vector Machine - Recursive Feature Elimination (SVM-RFE) ß3], among the others.

As for feature extraction, the most popular method is Principal Component Analysis (PCA) [109], which converts a set of observations of possibly correlated features into a set of values of linearly uncorrelated features called principal components. The number of principal components is less than or equal to the number of original features.

3.2 Explicit Feature Learning

3.2.1 Feature Selection methods

New feature selection methods have been steadily appearing during the last few decades. The proliferation of feature selection algorithms, however, has not brought about a general methodology that allows for intelligent selection from existing algorithms. Nevertheless, although a vast body of feature selection methods exists, some of them have been able to stand out and their use has become very popular among researchers. A brief description of them can be found in Table 3.1.

	Filter/Embedded	Ranker/Subset	Reference
Chi-Squared	Filter	Ranker	Liu and Setiono [131]
F-score (Fisher score)	Filter	Ranker	Duda et al. [65]
Information Gain	Filter	Ranker	Quinlan [155]
ReliefF	Filter	Ranker	Kononenko [121]
mRMR	Filter	Ranker	Peng et al. [150]
SVM-RFE	Embedded	Ranker	Guyon et al. [83]
CFS	Filter	Subset	Hall [86]
FCBF	Filter	Subset	Yu and Liu [206]
INTERACT	Filter	Subset	Zhao and Liu [210]
Consistency	Filter	Subset	Dash and Liu [60]

Table 3.1: Frequently used feature selection methods.

These methods are usually present in the review papers that appear in the literature for feature selection. In [161], classical feature selection techniques are provided in the form of a basic taxonomy and their applicability to bioinformatics applications is discussed. Another work on comparing state-of-the-art feature selection methods when dealing with thousands of features, using both synthetic data and real data, is presented in [98]. Brown et al. [38] presented a unifying framework for feature selection based on information theory, covering up to 17 different methods. More recently, the performance of well-known feature selection methods in the presence of several complications (such as noise, redundancy or interaction between attributes) was tested in [33].

Recent contributions

There exist numerous papers and books proving the benefits of the feature selection process [81]. In [161], classical feature selection techniques are provided in the form of a basic taxonomy and their applicability to bio-informatics applications is discussed. Another work on comparing state-of-the-art feature selection methods when dealing with thousands of features, using both synthetic data and real data, is presented in [98]. Brown et al. [38] presented a unifying framework for feature selection based on information theory, covering up to 17 different methods. More recently, the performance of well-known feature selection methods in the presence of several complications (such as noise, redundancy or interaction between attributes) was tested in [33].

However, since none of the existing methods mentioned has demonstrated significantly superiority over the others, researchers are usually focused on finding a good method for a specific problem setting. Therefore, new and novel feature selection methods are constantly appearing using different strategies. In the last few years, the review of the literature has shown a tendency to mix algorithms, either in the form of hybrid methods [187, 97, 126, 200] or ensemble methods [88, 203, 204, 114, 34].

Applications

Feature selection methods are currently being applied to problems of very different areas. In the next paragraphs we will describe some of the most popular applications that are promoting the use of feature selection:

- **Computational biology.** Bio-informatic tools have been widely applied to genomics, proteomics, gene networks, structure prediction, disease diagnosis and drug design. DNA microarrays have been widely used in simultaneously monitoring mRNA expressions of thousands of genes in many areas of biomedical research. These data sets typically consist of several hundred samples as opposed to thousands of genes, whereby feature selection is paramount. Because of this, a myriad of works in the feature selection field have been devoted to help in the classification of DNA

microarrays. A complete review of up-to-date feature selection methods developed for dealing with microarray data can be found in [32].

- **Face recognition**. The recognition of a human face has a wide range of applications, such as face-based video indexing and browsing engines, biometric identity authentication, human-computer interaction, and multimedia monitoring/surveillance. An important issue in this field is to determine which features from an image are the most informative for recognition purposes, so feature selection algorithms for face recognition have been recently suggested [129, 134, 9, 158, 137, 111]
- **Health studies**. The recent explosion in data available for analysis is as evident in health care as anywhere else. Private and public insurers, health care providers, particularly hospitals, physician groups and laboratories, and government agencies are able to generate far more digital information than ever before. Many health studies are longitudinal: each subject is followed over a period of time and many covariates and responses of each subject are collected at different time points. Feature selection has proven effective in helping with the diagnosis of several diseases, such as retinopathy of prematurity [12], evaporative dry eye [160], pulmonary nodules [127] or cardiac pacemaker implantation [103], among others.
- **Financial engineering and risk management**. Technological revolution and trade globalization have introduced a new era of financial markets. Over the last three decades, an enormous number of new financial products have been created to meet customer demands. The stock market trend is very complex and is influenced by various factors. Therefore it is very necessary to find out the most significant factors of the stock market and feature selection can be applied to achieve this goal [99, 116, 128, 186].
- **Text classification**. The categorization of documents into a fixed number of predefined categories has become a popular problem in Internet applications such as Spam email or shopping. Each unique word in a document is considered as a feature, so it is highly important to select a subset of all the possible features, allowing to reduce the computational requirements of learning algorithms. In the last few years, a number of works which promote the use of feature selection for text categorization have been presented [70, 115, 59, 71, 187].

3.2.2 Distance Metric Learning

Distance metric learning (DML) methods try to learn the best metric for a specific input space and dataset. The performance of a learning algorithm (nearest-neighbors classifiers, kernel algorithms etc.) mostly depends on the metric used. Many DML algorithms have been proposed. All of them try to find a positive semi-definite (PSD) matrix $\mathbf{M} \in \mathbb{R}^{m \times m}$

such that the induced metric

$$d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j)^\top \mathbf{M}(\mathbf{x}_i - \mathbf{x}_j) \tag{3.2}$$

is optimal for the task at hand. For example, the Euclidian distance is a special case where $\mathbf{M} = \mathbf{I}$. There are three principal families of DML algorithms [197]: *eigenvector methods*, *convex optimization* and *neighborhood component analysis*.

In the eigenvector methods, the matrix $\mathbf{M}$ is parameterized by the product of a real valued matrix with its transposed, namely $\mathbf{M} = \mathbf{L}^\top \mathbf{L}$, in order to maintain the matrix positive semi-definite. In this case, the matrix $\mathbf{M}$ is called *Mahalanobis metric*. These methods use the covariance matrix to optimize the linear transformation $\mathbf{x}_i \rightarrow \mathbf{L}\mathbf{x}_i$ that projects the training inputs. Finding the optimal projection is the task of eigenvector methods with a constraint that defines $\mathbf{L}$ as a projection matrix: $\mathbf{L}\mathbf{L}^\top = \mathbf{I}$. These algorithms do not use the training labels and then they are totally unsupervised.

Convex optimization algorithms represent another family of DML algorithms. It is possible to formulate a DML as a convex optimization problem over the cone of correct matrices $\mathbf{M}$. This cone is the cone of positive semi-definite matrices

$$\mathfrak{M} = \{\mathbf{M} \in \mathbb{R}^{m \times m} : \forall \tau \in eig(\mathbf{M}), \tau \geq 0\}. \tag{3.3}$$

Algorithms in this family are supervised and optimal positive semi-definite matrix $\mathbf{M}$ is obtained optimizing the square root of the *Mahalanobis metric* and enforcing the SDP constraint $\mathbf{M} \succeq 0$. There are also online versions of convex optimization algorithms for DML, like *POLA* [169] for example.

Another family of algorithms for DML is called neighborhood component analysis. In [76], for example, the authors try to learn a *Mahalanobis metric* from the expected leave-one-out classification error. In this case they use a stochastic variant of k-nearest neighbor with *Mahalanobis metric*. This algorithm has an objective function that is not convex and can suffer from local minima. Metric Learning by Collapsing Classes (MLCC) [75] is an evolution of the above mentioned method that can be formulated by a convex problem but with the hypothesis that the examples in each class have only one mode. Another important algorithm in this family is the Large Margin Nearest Neighbor Classification (LMNNC) [197] that learns a *Mahalanobis distance metric* with a k-nearest neighbor by semi-definite programming and also in this case we have the semi-positive constraint for $\mathbf{M}$ in the optimization problem. Finally, a generalization of the LMNNC is the Gradient Boosted LMNNC (GB-LMNNC) [113] that learns a non-linear transformation directly in the function space. Specifically, it extends the *Mahalanobis metric* between two examples (e.g.: $\|\mathbf{L}\mathbf{x}_i - \mathbf{L}\mathbf{x}_j\|_2$) by using a non linear transformation ϕ to define the new Euclidian

distance $\|\boldsymbol{\phi}(\mathbf{x}_i) - \boldsymbol{\phi}(\mathbf{x}_j)\|_2$. Given the non linearity of $\boldsymbol{\phi}$, GB-LMNNC uses the gradient boosted regression tree in order to change the metric (GBRT) [72]. So, the algorithm learns and combines an ensemble of multivariate regression trees (that are *weak* learners) using gradient boosting that minimizes the original LMNN objective function in the function space.

3.3 Implicit Feature Learning

3.3.1 Transductive Feature Extraction with Non Linear Kernels

Algorithms in this class of KL methods are able to perform feature extraction in the feature space defined by a non linear kernel. In this case, the feature mapping from the input space to the feature space is not explicitly defined. A popular solution is given by *Kernel Principal Component Analysis* (KPCA) [164]. KPCA uses the *kernel trick* to implicitly find the projections on the eigenvectors (principal directions) and the corresponding eigenvalues of the covariance matrix in feature space. KPCA can also be considered as an unsupervised manifold learning technique mapping data points into a lower-dimensional space.

The intrinsic problem of these algorithms is the transductive environment that they require (i.e. both training and test example needs to be available before training the classifier). This problem can be overcome by using *Out-Of-Sample* techniques to approximate the kernel values on new examples. Empirical experiments have shown that errors on examples in the *Out-Of-Sample* set and examples in the *In-Sample* set are similar [30].

3.3.2 Spectral Kernel Learning

These methods are founded on the spectral decomposition of the Laplacian graph $\mathbf{L}$, that is an undirected graph that contains the manifold structure of the data. By using these methods we are interested in finding the smoothest components of$\mathbf{L}$ (i.e. the eigenvectors with the smaller eigenvalues) and hence building a kernel which penalizes large changes between strongly connected nodes. This can be made by changing the spectral representation of $\mathbf{L}$ rescaling the eigenvalues in according to the semi-supervised information, for examples using linear programming [132]. More specifically, these algorithms are based on the possibility to write a semi-positive definite matrix $\mathbf{L} \in \mathbb{R}^{l \times l}$ using the equation of its spectral decomposition:

$$\mathbf{L} = \sum_{s=1}^{l} \lambda_s \mathbf{u}_s \mathbf{u}_s^T = \mathbf{U}\mathbf{D}_{\boldsymbol{\lambda}}\mathbf{U}^T \tag{3.4}$$

where $\lambda_s \geqslant 0 \ \forall s = 1, \ldots, l$ are the eigenvalues, with their corresponded eigenvectors $\{\mathbf{u}_s\}_{s=1}^{l}$,$\mathbf{U}$ is the matrix of the eigenvectors in rows and $\mathbf{D}_{\boldsymbol{\lambda}}$ is a diagonal matrix with the eigenvalues as non-null entries. The matrices defined by $\mathbf{u}_i\mathbf{u}_i^T$ are $1-$rank projection

matrices and are kernels obtained from $\mathbf{L}$. Then, the matrix $\mathbf{L}$ is a weighted sum of $1-$rank matrices where the weights are the eigenvalues. A nonparametric spectral transformation $\tau : \lambda_s \to \mu_s$ optimized to the specific task is performed using the semi-supervised information contained in $\mathbf{L}$. Basically, a new set of features are created using the *spectral embedding*, and the matrix of the examples (where a row is an example) is now defined by

$$\mathbf{X} = \mathbf{U}\sqrt{\mathbf{D}_{\boldsymbol{\mu}}} \in \mathbb{R}^{l \times l}. \tag{3.5}$$

In particular, the i^{th} example is defined by $\mathbf{x}_i = [\sqrt{\mu_s}\mathbf{u}_{i,s}]_{s=1}^{l}$ highlighting the relationship between feature learning and spectral kernel learning. Clearly, changing the eigenvalues, the algorithm is indirectly changing the weight of the features of the *spectral embedding*.

3.3.3 Multiple Kernel Learning (MKL)

MKL [79] is one of the most popular paradigm used to learn kernels in real world applications [39, 213, 196, 181]. The kernels generated by these techniques are combinations of previously defined *weak* kernels $\mathbf{K}_1, ..., \mathbf{K}_R$ with a constraint in the form:

$$H_R^q = \{\mathbf{x} \mapsto \mathbf{w} \cdot \boldsymbol{\phi}_{\mathbf{K}}(\mathbf{x}) : \mathbf{K} = \sum_{k=1}^{R} \mu_k \mathbf{K}_k, \boldsymbol{\mu} \in \Gamma_q, \|\mathbf{w}\| \leqslant 1\} \tag{3.6}$$

with $\Gamma_q = \{\boldsymbol{\mu} : \boldsymbol{\mu} \succcurlyeq 0, \|\boldsymbol{\mu}\|_q = 1\}$ and considering the function $\boldsymbol{\phi}_{\mathbf{K}}$ as the feature mapping from the input space to the feature space. The value q being the kind of mean used, is typically fixed to 1 or 2.

Using this formulation, we can be interested in the meaning of a sum of kernels in the feature space. It is well know that the sum of two kernels can be seen as the concatenation of the features contained in both the RKHS [172]. Extending the same idea, the weighted sum of a list of *weak* kernels can be seen as a weighted concatenation of all the features contained in all the RKHS (where the weights are the square roots of the learned weights μ_k).

These algorithms are supported by several theoretical results that bound the *estimation error* (i.e. the difference between the true error and the empirical margin error). These bounds exploit the *Rademacher complexity* applied to the combination of kernels [51, 101, 118, 110]. Existing MKL approaches can be divided in two principal categories. In the first category, *Fixed or Heuristic*, some fixed rule is applied to obtain the combination. They usually get results scalable with respect to the number of kernels combined but their effectiveness will critically depend on the domain at hand. On the other hand, the *Optimization based* approaches learn the combination parameters by solving an optimization problem that can be integrated in the learning machine (e.g. structural risk based target

function) or formulated as a different model (e.g. alignment, or other kernel similarity maximization) [157, 189, 117, 202, 182, 17, 179].

In the *Fixed or Heuristic* family there are some very simple (but effective) solutions. In fact, in some applications, the average method (that equal to the sum of the kernels [23]) can gives better results than the combination of multiple SVMs each trained with one of these kernels [148]. Another solution, can be the element-wise product of the kernel matrices contained in the *weak* family.

Some heuristic methods exploit the notion of *kernel alignment* [55]. This similarity measure can be seen as the cosine of the angle between two kernels. Formally, given two kernels $\mathbf{K}_1$ and $\mathbf{K}_2$ their kernel alignment is:

$$A(\mathbf{K}_1, \mathbf{K}_2) = \frac{\langle \mathbf{K}_1, \mathbf{K}_2 \rangle_F}{\sqrt{\langle \mathbf{K}_1, \mathbf{K}_1 \rangle_F \langle \mathbf{K}_2, \mathbf{K}_2 \rangle_F}}, \tag{3.7}$$

where $\langle \mathbf{K}_1, \mathbf{K}_2 \rangle_F = \sum_{i=1}^{L} \sum_{i=1}^{L} \mathbf{K}_1[i,j]\mathbf{K}_2[i,j]$. For example, Qiu and Lance [154] exploits the kernel alignment to propose the following heuristic rule for the k^{th} weight:

$$\mu_k = \frac{A(\mathbf{K}_k, \hat{\mathbf{y}}\hat{\mathbf{y}}^t)}{\sum_{h=1}^{R} A(\mathbf{K}_k, \hat{\mathbf{y}}\hat{\mathbf{y}}^t)}, \tag{3.8}$$

where $\hat{\mathbf{y}}\hat{\mathbf{y}}^t$ is the *ideal kernel* for a binary classification task. This formula sets the weight of the k^{th} kernel in the *weak* family proportional to its alignment with respect to the ideal kernel.

The second family of MKL algorithms are defined exploiting an optimization problem. Unexpectedly, finding a good kernel by solving an optimization problem turned out to be a very challenging task, e.g. trying to obtain better performance than the simple average of the weak kernels is not an easy task[1]. Moreover, the *Optimization based* MKL algorithms have a high computational complexity, for example using semidefinite programming or quadratically constrained Quadratic Programming (QP).

The most famous MKL algorithms in the *Optimization based* family are the following:

- **Simple MKL (SMKL)** An iterative algorithm by Rakotomamonjy [157] that implements a linear approach with kernel weights in a simplex. Basically SMKL works by repeating two main steps:
  - A SVM optimization problem defined on current weights.
  - Updating of the kernel weights using a gradient function.
- **Generalized MKL (GMKL)** This algorithm, by Varma and Babu [189], is called Generalized Multiple Kernel Learning (GMKL). GMKL exploits a nonlinear approach and tackles the problem with a technique that regularizes both the hyperplane

[1] `www.cse.msu.edu/~cse902/S14/ppt/MKL_Feb2014.pdf`

weights and the kernel combination weights. This algorithm tries to optimize a non convex problem, different from SMKL but, in general, with better results [79].

- **Lasso-based MKL (GLMKL)** The algorithm, by Kloft [117] and Xu [202] considers a regularization of kernel weights with 1-norm and finds the kernel weights by solving a small QP problem at each iteration.
- **SMO Projected Gradient Descent Generalized MKL (SPG-GMKL)** This algorithm is inspired by the GMKL and learns simultaneously both kernel and SVM parameters. This particular algorithm [104] exploits an efficient and highly scalable implementation and is a state-of-the-art method with respect to the computational performance.
- **Localized MKL (LMKL)** This method combines kernels using weights calculated from a gating model. The optimization problem is not convex and a two-step alternate optimization procedure is used to find the classifier parameters and the gating model parameters [78].
- **Non Linear MKL (NLMKL)** This algorithm, by Cortes et al. [49], exploits a non-linear kernel combination based on kernel ridge regression (KRR) and polynomial combination of kernels.

In Table 3.2, these algorithm are summarized with respect to their computation complexity.

	Learner	Complexity	Reference
SimpleMKL	SVM	Gradient + QP	Rakotomamonjy et al. [157]
GMKL	SVM	Gradient + QP	Varma and Babu [189]
GLMKL	SVM	Analytical + QP	Kloft et al. [117]
SPG-GMKL	SVM	Gradient + QP	Jain et al. [104]
LMKL	SVM	Gradient + QP	Gönen and Alpaydin [78]
NLMKL	KRR	Gradient + Matrix Inversion	Cortes et al. [49]

Table 3.2: Frequently used MKL *Optimization based* methods.

In the last years, the importance of combining a large amount of kernels becomes clear creating the idea of *deeper kernel*[2]. New methods can combine thousands of kernels with acceptable computational complexity contrasting the previous idea that KL is shallow in general. In fact, having MKL algorithms which are scalable opens a new scenario for MKL. While standard MKL algorithms typically cope with a small number of strong kernels and try to combine them (each kernel representing a different view of the same task). In this case, the kernels are individually well designed by experts and their optimal combination hardly leads to a significant improvement of the performance with respect to, for example, a simple averaging combination. In the new scenario, the MKL paradigm

[2] `http://videolectures.net/roks2013_shawe_taylor_kernels/`

can be exploited to combine a very large amount of *weak* kernels, aiming at boosting their combined accuracy in a way similar to feature weighting.

Theoretical results prove that the combination of a large number of kernels using the MKL paradigms is able to add only a small penalty in the *generalization error*, as presented in [101]. In fact, if we consider the class of linear function in the feature space of a kernel $\mathbf{K}$ as

$$\mathcal{F}_{\mathbf{K}} = \{\mathbf{x} \rightarrow \mathbf{w} \cdot \phi_{\mathbf{K}}(\mathbf{x}) : \|\mathbf{w}\|_2 = 1\}, \tag{3.9}$$

the bound on the generalization error of MKL gives only a logarithmic additive dependence with respect to the number of kernels if we learn a function in the family:

$$\mathcal{F} = \{\sum_{i=1}^{R} a_i f_i : f_i \in \mathcal{F}_{\mathbf{K}_i}, a_i \in \mathbb{R} \geq 0, \sum_{i=1}^{R} a_i \leq B\}. \tag{3.10}$$

In this way, tacking a set of linear *weak* kernels that are evaluated over a single feature, the connection between MKL and feature learning is clear. The single kernel weight is, in fact, the weight of the feature. Using this framework, we can weight the information contained into bunch of features, evaluated in different ways (i.e. using different kernels that can consider different subsets of features) [16, 106]. Also Cortes[3] highlighted, in the last few years, the connection between MKL and feature selection with Group-Lasso.

Hierarchical Kernel Learning

Hierarchical Kernel Learning (HKL) [13] is a generalization of the MKL framework. The idea is to design a hierarchy over the given *weak* kernels. The key assumption of the HKL is that the base kernels can be embedded in a Directed Acyclic Graph (DAG). Under this constraint, the HKL algorithms maintain an acceptable polynomial complexity (otherwise the number of base kernels grow exponentially with respect to the number of features). The key advantage of HKL is the ability to perform non-linear feature selection.

In fact, the *weak* kernels can be embedded on a DAG by exploiting a norm regularization over the associated RKHS. This regularization has to be able to induce a specific sparsity pattern over the selected *weak* kernels. In the context of HKL the *weak* kernel are also called basis or local kernels.

Many kernels can be written as a sum of base kernels indexed by a certain set V, in the form:

$$K(\mathbf{x}, \mathbf{z}) = \sum_{i \in V} K_i(\mathbf{x}, \mathbf{z}). \tag{3.11}$$

RBF and polynomial kernels can be written in this form. Two simplified examples are the following:

[3] `http://www.cse.msu.edu/~cse902/S14/ppt/MKL_Feb2014.pdf`

- **RBF** for the case $\mathbf{x} = x \in \mathbb{R}$, by exploiting the eigen-decomposition of the non centered covariance operator for a normal distribution with variance $1/4a$ [199]:

$$K(x,z) = \sum_{k=0}^{\infty} \frac{(b/A)^k}{2^k k!} \Omega(x)\Omega(z) \tag{3.12}$$

  where $\Omega(x) = \exp(-\frac{b}{a}(a+c)x^2 H_k(\sqrt{2}cx))$, $c^2 = a^2 + 2ab$, $A = a+b+c$ and H_k is the k^{th} Hermite polynomial function.
- **Polynomial kernel** of degree q and coefficient c:

$$K_{q,c}(\mathbf{x},\mathbf{z}) = \sum_{d=0}^{q} \binom{q}{d} c^{q-d} (\mathbf{x}\cdot\mathbf{z})^d. \tag{3.13}$$

HKL assumes one separate predictor $\mathbf{w}_i$ for each kernel K_i and then, the final prediction can be written as a function f such that:

$$f(\mathbf{x}) = \sum_{i \in V} \mathbf{w}_i \cdot \boldsymbol{\phi}_i(\mathbf{x}). \tag{3.14}$$

HKL imposes a $\ell_1 - \ell_2$ norm using the following regularizer:

$$\sum_{g \in G} \|\mathbf{w}_g\|_2 = \sum_{g \in G} \sqrt{\sum_{j \in g} (\mathbf{w}_j)^2}, \tag{3.15}$$

where G is a partition of the features. When we have not a previously fixed partition of the features (from *a priori* knowledge), a structured matrix factorization has been proposed in [15, 107] in order to obtain a strictured regularization of the base kernels. This regularization can be used to obtain a very sparse decomposition, solving an optimization problem.

This implies that the prediction function involves very few kernels. Considering the condition of strictly positivity of the kernels, it can be shown that this hierarchical penalization induces a strong sparsity property [105]: if a kernel is not selected, then none of the kernels associated with its descendants are selected. Also, the weight of a kernel associated to a specific node is always greater than the weight of the kernels associated with its descendants.

These limitations are a serious problem of the HKL algorithms, in fact, they lead to selection of many redundant features. In [105] two evolutions of the HKL are presented: the Generalized HKL (gHKL) and the Generalized HKL for MTL (gHKL$_{MT}$). The gHKL framework has a more flexible kernel selection pattern by exploiting a ℓ_1/ℓ_p regularizer, with $p \in (1,2)$, maintaining the polynomial complexity of the method. This regularizer mitigates the weight bias towards the kernels that are nearer to the sources of the DAG,

improving the overall performance.

Applications of MKL

- **Object Recognition.** The goal of visual object recognition is to determine if a given image contains one or more objects belonging to the class of interest (without necessarily locating the position of the object in the image). It is a challenging task because of the large variations in the visual appearance of objects from the same class caused by viewpoint variation and occlusion, articulation, background clutter, illumination, and pose change. A number of studies have shown that MKL is a useful tool for object recognition, where each image is represented by multiple sets of features and MKL is applied to combine different feature sets. It is well known that given a sufficient number of training examples and feature/kernel types, MKL is more effective for object recognition than simple kernels combination (e.g., choosing the best performing kernel or average of kernels) [205, 40].
- **Biomedical applications.** In biomedical applications, there are usually a small number of sources and most of these data sources are carefully selected and preprocessed. They thus often are directly relevant to the problem. In these cases, using a MKL approach to combine the source of information is a useful way to improve the performance of the biomedical applications. For example when we have to deal with the biomedical data fusion [43].
- **Health studies.** When we have to exploit the health care data, the most important step is to correctly select the helpful information from the noise. Nowadays, we have the possibility to obtain large amount of health data than ever before. So, given a specific task, it is mandatory to use only the really useful information. From the different sources of the health information, MKL is able to rank which is the most important point of views in order to solve a specific task [207, 133].

4. Graph Kernels

Historically, the machine learning community has been focused on data represented as vectors, because they are a flexible and mathematically convenient representation. However, in the last few years, there has been an increasing interest for data represented in a structured form. Indeed, in many application domains, it is easy to define a representation for the data as a structure. For example, XML files have a natural representation as trees, as well as the tree of a sentence in natural language processing. Unfortunately, it is not straightforward to define vectorial representations that are general enough and that do not force a loss of information. Graphs are a very convenient way to store information, and they can encode concepts and relationships from very different application domains in a natural way. Recently, different ambitious projects have been started with the aim to build a graph database that encodes (some of the) human knowledge, such as the Google knowledge graph [177] or NELL [139]. In Chemoinformatics a chemical compound can be easily represented as a graph, with nodes that represent atoms and edges that represent bonds between atoms. In computer vision, recent advances in image segmentation and semantic parsing gave a viable way to construct a graph representation of an image.

Dealing with data represented as graphs is not simple, since even the basic operations can be computationally expensive (see for example the graph isomorphism problem [159]). The approach that most graph kernels follow is to compare two graphs with respect to the substructures they share. In order to be computationally efficient, the considered substructures are limited to particular types, for which the computation of the isomorphism is efficient. This comes at the cost of a reduced discriminative ability of the kernel.

Moreover, in order to control the complexity, the kernels usually depend on a parameter that regulates the size of the considered structures. In this way, they balance the tradeoff between efficiency and expressiveness of the kernel (i.e. how a kernel is able to discriminate between similar examples). An open problem, when we have to deal with kernels for structured data, is the weight given to each feature. The particular weighting policy to apply can strongly influence the predictive performance of the kernel. At the moment, the policy is usually integrated in the kernel definition, possibly including also strong regularization terms, and no much work has been done to evolve this methodology. In addition to that, among the different graph kernels that have been proposed in literature, usually it is not straightforward to decide which one will be most suited for a specific task.

4.1 An introduction to Graph kernels

The vast majority of kernels for structured data are based on the convolution framework. The idea is to decompose a structure into a set of simpler structures (where a *base* kernel between these structures is given) and to define the kernel as a function of the base kernels. More formally, let $\mathcal{X}$ be a space of objects (that are our data points) and let each object x be associated with a finite subset $\mathcal{X}'_x$ of a space $\mathcal{X}'$. Furthermore, assume that a kernel over this domain $k : \mathcal{X}' \times \mathcal{X}' \to \mathbb{R}$ is defined. To define an $R-convolution$ kernel, Haussler [89] assumed a *finite* relation $R \subseteq \mathcal{X}' \times \mathcal{X}$, and let

$$K(x,y) = \sum_{(x',x)\in R} \sum_{(y',y)\in R} k(x',y').$$

Many recently defined graph kernels are members of this general framework.

Many graph kernels depend on a parameter h that defines the maximum depth of the considered substructures. Usually, too small values of h do not allow the kernel to capture enough global information about the structure of the graphs, resulting in poor predictive performances. In the same way, too high values of h tend to obfuscate the contributions of the small substructures. Moreover, the higher the h, the slower the calculation of the kernel becomes.

4.1.1 Ordered Decomposition DAG Kernels for Graphs

This section briefly describes the framework of ODD-Kernels for graphs, proposed in [57]. The idea of the ODD kernel framework is to decompose the input graph into a set of substructures for which the isomorphism can be computed efficiently, i.e. subtrees.

In order to map the graphs into trees, two intermediate steps are needed:

1. map the graph G into a multiset of DAGs $\{DD_G^{v_i} | v_i \in V_G\}$, where $DD_G^{v_i}$ is obtained by keeping each edge in the shortest path(s) connecting v_i with any $v_j \in V_G$. The

Decomposition DAGs kernel for graphs can be defined as

$$DDK_{K_{DAG}}(G_1, G_2) = \sum_{D_1 \in DD_{G_1}} \sum_{D_2 \in DD_{G_2}} K_{DAG}(D_1, D_2). \tag{4.1}$$

2. Since the vast majority of DAG kernels are extensions of kernels for ordered trees, a strict partial order between DAG nodes in $DD_G^{v_i}$ has been defined yielding Ordered Decomposition DAGs $ODD_G^{v_i}$. The ordering function considers the label of the vertex $L(v)$ and, recursively, the labels of the children of v [57].
3. Finally, any Ordered DAG (ODD) is mapped into a multiset of trees. Let us define $T(v_i)$ as the tree resulting from the visit of $ODD_G^{v_i}$ starting from node v_i: the visit returns the nodes reachable fromv_i in $ODD_G^{v_i}$. Note that if a node v_j can be reached more than once, more occurrences of v_j will appear in $T(v_i)$. Also, the tree visit $T(v_i)$ can be stopped when the tree $T(v_i)$ reaches a maximum height r. Such tree is referred to as $T_r(v_i)$.

In [57] the Ordered Decomposition DAGs kernel is defined as:

$$ODDK(G_1, G_2) = \sum_{\substack{OD_1 \in ODD_{G_1} \\ OD_2 \in ODD_{G_2}}} \sum_{j=1}^{r} \sum_{\substack{v_1 \in V_{OD_1} \\ v_2 \in V_{OD_2}}} C(r(T_j(v_1)), r(T_j(v_2))) \tag{4.2}$$

where C is a function defining a tree kernel. Among the kernels for trees defined in literature, the one employed in the paper is the *Subtree kernel* [192], which counts the number of shared proper subtrees between the two input trees. In this case, the C function is defined as follows:

$$C_{ST}(v_1, v_2) = \begin{cases} \lambda & \text{if } L(v_1) = L(v_2) \text{ and} \\ & \rho(v_1) = \rho(v_2) = 0 \\ \lambda \Upsilon(v_1, v_2) & \text{if } L(v_1) = L(v_2) \text{ and} \\ & \rho(v_1) = \rho(v_2) \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

where λ is a weighting parameter, that will be discussed in more detail in the next section, and

$$\Upsilon(v_1, v_2) = \prod_{j=1}^{\rho(v_1)} C_{ST}(ch_{v_1}[j], ch_{v_2}[j]). \tag{4.4}$$

The resulting kernel will be referred to as ODD_{ST}.

Summarizing, Ordered Decomposition DAGs kernel consider as features all the proper subtrees of the trees resulting from breadth-first visits starting from each node in the graph.

In [56] the authors have shown that it is possible to efficiently compute the explicit feature space representation for such a kernel. Let us assume that a bijective function h mapping labeled trees to integers is given (or equivalently, an enumeration of all possible labeled trees). Then, given a value for r, the $i-$th component of $\boldsymbol{\phi}(G)$ is defined as

$$\phi_i(G) = |\{h(T_j(v)) = i : j \leq r, v \in D, D \in ODD_G\}| \cdot \lambda^{size(h^{-1}(i))},$$

For ease of notation, $size(G) = |V_G|$. Finally, it is important to note that the property $h(s) = i$ is equivalent to the isomorphism between trees, which can be solved in polynomial time.

4.1.2 ODD kernels and feature weighting

The proposed approach exploits two convenient considerations:

1. when a tree is present in the feature space representation of a graph, then all its proper subtrees are in the same space as well;
2. the bigger is the tree, the higher is the number of its subtrees that occur.

As a direct consequence, kernels that count common substructures suffer an issue of diagonally dominant gram matrix, i.e. the number of substructures shared by a graph with itself (and thus the corresponding kernel value) is usually much higher than the number of shared substructures between two different instances. This negatively affects the support vector classifier, since the classification of an instance may be dominated by the contribution of the most similar example, leading to a behavior similar to a nearest neighbor classifier. A possible solution for this issue is the down-weighting of big structural features. Usually this is accomplished with a function of the size of a feature, i.e. to the number of co-occurring features. The (down)weighting function, firstly proposed for tree kernels [46], for a feature f is defined as:

$$w(f) = \sqrt{\lambda^{size(f)}} \cdot freq(f), \tag{4.5}$$

where $freq(f)$ is the frequency of a feature in the considered example, and $size(f)$ is the size of the sub-structure associated to the feature f. Here, the weight of a feature grows with its frequency in the example, and exponentially decreases (if $\lambda < 1$) with its size. Note that the selection of an appropriate λ parameter is a key step in the successful application of this weighting scheme. This weighting scheme intuitively makes sense, but there is no reason why other schemes cannot be more suited for a specific task. In a recent work [58], another weighting scheme for graph kernels has been proposed:

$$w(f) = tanh(\sigma \cdot freq(f) \cdot size(f)). \tag{4.6}$$

However, testing different weighting schemes is unpractical for different reasons. Firstly, when we test different weighting schemes we are adding a new parameter to validate. In addition, it is not clear which weight distributions one should test. Finally, in noisy datasets, where the selection for the optimal parameters is a problem, the gap in predictive performance between the different schemes may be not enough to justify the added complexity. Moreover, the kernel may be more prone to overfitting.

From a theoretical point of view, if enough examples are available, the weighting policy should be irrelevant because the weight of each feature should be adjusted by the learning algorithm. However, especially when dealing with real-world graph datasets, this is not the case, and the dependencies among features makes the learning even more difficult.

5. Optimization of the Margin Distribution

In this chapter an algorithm to learn a model that optimizes the margin in a classification task is presented. This algorithm, called *Kernel Optimization of the Margin Distribution* (KOMD), is presented in [6] as a game theoretic interpretation as a two-player zero-sum game. KOMD has been proposed for the problem of margin maximization in a classification task. Specifically, the classification task has been split in two phases. Firstly, in the ranking phase, a total order of the examples is introduced and then, the pure binary classification is performed by applying a threshold to the ranking of the examples obtained in the first phase.

5.1 Playing with the margin

In the first phase of the algorithm (ranking phase), the task is to learn a unitary norm vector $\mathbf{w}$ such that

$$\mathbf{w}^\top(\boldsymbol{\phi}(\mathbf{x}_\oplus) - \boldsymbol{\phi}(\mathbf{x}_\ominus)) > 0 \tag{5.1}$$

for most of positive ($\mathbf{x}_\oplus$) and negative ($\mathbf{x}_\ominus$) instance pairs in the training data. The game basically consists of one player that has to choose one vector of unitary norm $\mathbf{w}$ and the other that picks pairs of positive-negative examples according to two distributions $\boldsymbol{\gamma}^+$ and $\boldsymbol{\gamma}^-$ over the positive and negative examples, respectively. The value of the game is the expected margin obtained using these variables, that is $\mathbf{w}^\top(\boldsymbol{\phi}(\mathbf{x}_p) - \boldsymbol{\phi}(\mathbf{x}_n)), \mathbf{x}_p \sim \boldsymbol{\gamma}^+, \mathbf{x}_n \sim \boldsymbol{\gamma}^-$. The goal of the first player is to maximize this value while the second player wants to minimize it. This problem is equivalent to the hard SVM and can be solved efficiently by

optimizing a simple linearly constrained convex function on variables $\boldsymbol{\gamma} \in \hat{\Gamma}$, namely,

$$\min_{\boldsymbol{\gamma} \in \hat{\Gamma}} \underbrace{\boldsymbol{\gamma}^\top \hat{\mathbf{Y}} \hat{\mathbf{K}} \hat{\mathbf{Y}} \boldsymbol{\gamma}}_{\mathcal{Q}(\boldsymbol{\gamma})}. \tag{5.2}$$

It can be seen that the vector $\boldsymbol{\gamma}^* \in \hat{\Gamma}$ minimizing $\mathcal{Q}(\boldsymbol{\gamma})$ identifies the two nearest points in the convex hulls of positive and negative examples, respectively, in the feature space of the kernel $\mathbf{K}$.

Furthermore, a quadratic regularization over $\boldsymbol{\gamma}$ is introduced, namely, $\mathcal{R}(\boldsymbol{\gamma}) = \boldsymbol{\gamma}^\top \boldsymbol{\gamma}$, that makes the player to prefer optimal distributions (strategies) with low variance. In fact, let p (resp. n) the number of positive examples (resp. negative examples) in the training set, then

$$\mathbb{E}[\boldsymbol{\gamma}_+] = \frac{1}{p} \tag{5.3}$$

$$\mathbb{E}[\boldsymbol{\gamma}_-] = \frac{1}{n} \tag{5.4}$$

is always true by construction, and $\mathbb{E}[\mathbf{v}]$ denotes the expected value of elements in a vector $\mathbf{v}$. It follows that

$$\mathrm{Var}(\boldsymbol{\gamma}_+) = \mathbb{E}[\boldsymbol{\gamma}_+^2] - \mathbb{E}[\boldsymbol{\gamma}_+]^2 = ||\boldsymbol{\gamma}_+||^2 - p^{-2} \tag{5.5}$$

$$\mathrm{Var}(\boldsymbol{\gamma}_-) = \mathbb{E}[\boldsymbol{\gamma}_-^2] - \mathbb{E}[\boldsymbol{\gamma}_-]^2 = ||\boldsymbol{\gamma}_-||^2 - n^{-2}, \tag{5.6}$$

obtaining $\mathcal{R}(\boldsymbol{\gamma}) \propto \mathrm{Var}(\boldsymbol{\gamma}_+) + \mathrm{Var}(\boldsymbol{\gamma}_-)$.

The final best strategy for $\boldsymbol{\gamma}$ will be given by solving the optimization problem

$$\min_{\boldsymbol{\gamma} \in \hat{\Gamma}} (1 - \Lambda)\mathcal{Q}(\boldsymbol{\gamma}) + \Lambda \mathcal{R}(\boldsymbol{\gamma}). \tag{5.7}$$

The regularization parameter Λ has two critical points: $\Lambda = 0$ and $\Lambda = 1$. When $\Lambda = 0$, as we have shown before, there is no regularization, so the solution is the same as the hard SVM one, whereas when $\Lambda = 1$ the minimization problem reduces to

$$\min_{\boldsymbol{\gamma} \in \hat{\Gamma}} \mathcal{R}(\boldsymbol{\gamma}) \tag{5.8}$$

and the optimal solution is analytically defined by the vector of uniform distributions over

positive and negative examples, that is

$$\gamma_i^{\mu} = \frac{1}{p} \text{ when } y_i = +1 \tag{5.9}$$

$$\gamma_i^{\mu} = \frac{1}{n} \text{ when } y_i = -1. \tag{5.10}$$

In this case, the objective solution is the squared distance from the positive and negative centroids in feature space. The external parameter $\Lambda \in (0,1)$ allows to select the correct trade-off. Clearly, a correct selection of this parameter is fundamental if we are interested in finding the best performance for a classification task and this is usually made by validating on training data. In Figure 5.1, an example of the found solutions is depicted for a toy problem varying the value of Λ.

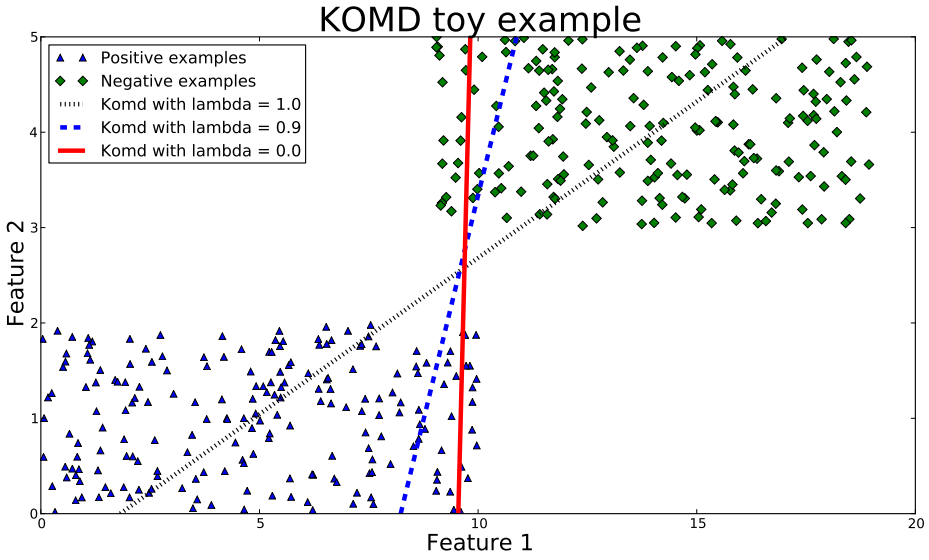


Figure 5.1: KOMD solutions of the first phase found using different Λ in a simple toy classification problem. The ranking among the examples is performed evaluating the *orthogonal projection* of the examples over the line defined by the solution.

Once the model is learned from training data, the evaluation on a new generic example $\mathbf{x}$ is obtained by:

$$f(\mathbf{x}) = \mathbf{w}^{\top}\boldsymbol{\phi}(\mathbf{x}) = \sum_i y_i \gamma_i \mathbf{K}(\mathbf{x}_i, \mathbf{x}) = \mathbf{K}_{tr}(\mathbf{x})\hat{\mathbf{Y}}\boldsymbol{\gamma}, \tag{5.11}$$

where $\mathbf{K}_{tr}(\mathbf{x}) = [\mathbf{K}(\mathbf{x}_1, \mathbf{x}), \ldots, \mathbf{K}(\mathbf{x}_l, \mathbf{x})]^{\top}$, i.e. the vector containing the kernel values with the training examples for $\mathbf{x}$.

When a pure binary classification is required, which is not the case in this thesis, then

the second phase of the algorithm is also performed. The threshold is set corresponding to the point standing in the middle between the optimal point in the convex hull of positive examples and the one in the convex hull of negative examples, that is

$$\theta = \frac{1}{2}(\sum_{i\in\oplus} \gamma_i f(\mathbf{x}_i) + \sum_{i\in\ominus} \gamma_i f(\mathbf{x}_i)) = \frac{1}{2}\sum_i \gamma_i f(\mathbf{x}_i) = \frac{1}{2}\boldsymbol{\gamma}^\top \hat{\mathbf{K}}\hat{\mathbf{Y}}\boldsymbol{\gamma}.$$

Note that, when $\Lambda = 0$ this choice corresponds exactly to the optimal hyperplane of SVM. Finally, a new example $\mathbf{x}$ will be classified according to $sign(f(\mathbf{x}) - \theta)$.

In the following we will refer to the first phase of the algorithm discussed in this section as KOMD (Kernelized Optimization of the Margin Distribution).

6. Rademacher Complexity

The complexity of the hypothesis has a crucial importance for a machine learning method allowing a control of the learning process and a trade-off of between underfitting and overfitting in model inference.

The *Rademacher complexity* theory[119, 21] has been developed in last few years as a data-dependent measure which takes advantage of the actual distribution of the data, differently from the VC-Dimension [10], which is independent from data. Moreover, a data-dependent measure of complexity and a data-dependent hypothesis space have a well known importance in machine learning [171]. A local version of the Rademacher complexity has been developed, called *local Rademacher complexity* by exploiting the idea of taking into account only the function that will be *most likely* chosen by the learning algorithm for the given sample for measuring the complexity of the class, [20, 120]

In the following, we will present the principal ideas concerning the *Rademacher complexity*.

6.1 The expressiveness of a class of functions

Controlling the expressiveness of a class of functions is important in order to avoid the identification of spurious patterns by the learning algorithm. One of the most insidious problem in machine learning is the possibility of detecting pattern in noise (i.e. the learning of a false pattern). It is important to remember that a machine learning algorithm detects a model from data in a probabilistic manner and then is always possible detecting a pattern in noise if the class of functions exploited by the algorithm is not correctly defined.

The *Rademacher complexity* estimates the ability of a class of functions to learn noise. Hence controlling this measure of capacity will intuitively guard against the problem of learning noise.

Now, we are ready to introduce the formal definition of *Rademacher complexity*, starting from the definition of *empirical Rademacher complexity*:

Definition 6.1.1 For a sample $S = \{x_1, ..., x_l\}$ generated by a distribution $\mathcal{D}$ on a set $\mathcal{X}$ and a real-valued function class $\mathcal{F}$ with domain $\mathcal{X}$, the *empirical Rademacher complexity* of $\mathcal{F}$ is the random variable

$$\hat{\mathcal{R}}_l(\mathcal{F}) = \mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{f\in\mathcal{F}}\left|\frac{2}{l}\sum_{i=1}^{l}\sigma_i f(x_i)\right| | x_1, \ldots, x_l\right], \tag{6.1}$$

where $\boldsymbol{\sigma} = \sigma_1, ..., \sigma_l$ are independent uniform $\pm 1-$valued (Rademacher) random variables.

The *Rademacher complexity* of $\mathcal{F}$ is

$$\mathcal{R}_l(\mathcal{F}) = \mathbb{E}_S\left[\hat{\mathcal{R}}_l(\mathcal{F})\right] = \mathbb{E}_{S\boldsymbol{\sigma}}\left[\sup_{f\in\mathcal{F}}\left|\frac{2}{l}\sum_{i=1}^{l}\sigma_i f(x_i)\right|\right]. \tag{6.2}$$

R The supremum over $f \in \mathcal{F}$ measures the best correlation that can be found between a function of the class $\mathcal{F}$ and the random labels.

The following theorem is the most important result concerning the *Rademacher complexity*, where we denote the input space with $\mathcal{Z}$, so in the case of supervised learning we would have $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$:

Theorem 6.1.1 ([172], Theorem 4.9) Fix $\delta \in (0,1)$ and let $\mathcal{F}$ be a class of functions mapping from $\mathcal{Z}$ to $[0,1]$. Let $z_1, \ldots, z_l$ be drawn independently according to a probability distribution $\mathcal{D}$. Then with probability at least $1-\delta$ over random draws of samples of size l, every $f \in \mathcal{F}$ satisfies

$$\mathbb{E}_{\mathcal{D}}[f(z)] \leq \hat{\mathbb{E}}[f(z)] + \mathcal{R}_l(\mathcal{F}) + \sqrt{\frac{\ln(2/\delta)}{2l}} \tag{6.3}$$

$$\leq \hat{\mathbb{E}}[f(z)] + \hat{\mathcal{R}}_l(\mathcal{F}) + 3\sqrt{\frac{\ln(2/\delta)}{2l}}. \tag{6.4}$$

where $\mathbb{E}_{\mathcal{D}}$ is the expectation with respect to the underlying distribution, while $\hat{\mathbb{E}}$ is the empirical expectation measured on a particular sample.

From this theorem, we can observe that with high probability the difference between the true and empirical error of the pattern function is bounded by the*Rademacher complexity*

of the pattern function class, modulo a small additional square root factor. Moreover, we can use directly the *empirical Rademacher complexity* value on the given training set.

In the following, some properties of the *Rademacher complexity* will be depicted. Let $\mathcal{F}, \mathcal{F}_1, \ldots, \mathcal{F}_n$ and $\mathcal{G}$ be classes of real functions,

- **Monotonicity with respect to the inclusion:** $\mathcal{F} \subseteq \mathcal{G} \Rightarrow \hat{\mathcal{R}}_l(\mathcal{F}) \leq \hat{\mathcal{R}}_l(\mathcal{G})$;
- **Multiplication for a constant:** $\forall c \in \mathbb{R}$, $\hat{\mathcal{R}}_l(\mathcal{F}) = |c|\hat{\mathcal{R}}_l(\mathcal{F})$;
- **Sum with a new function:** $\forall h$, $\hat{\mathcal{R}}_l(\mathcal{F}+h) \leq \hat{\mathcal{R}}_l(\mathcal{F}) + 2\sqrt{\hat{\mathbb{E}}[h^2]/l}$;
- **Sum of classes:** $\hat{\mathcal{R}}_l(\sum_{i=1}^{R} \mathcal{F}_i) \leq \sum_{i=1}^{R} \hat{\mathcal{R}}_l(\mathcal{F}_i)$.

Now, we are ready to highlight the connection between the *empirical Rademacher complexity* and (the complexity of) kernels. Firstly, the class of functions that we consider are linear functions with bounded norm

$$\mathcal{F}_B = \{\mathbf{x}_j \to \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}_j) : \|\mathbf{w}\|_2 \leq B\} \tag{6.5}$$

where $\boldsymbol{\phi}$ is the feature mapping for the kernel $\mathbf{K}$. The following bound theorem for the complexity of $\mathcal{F}_B$ holds:

Theorem 6.1.2 ([172], Theorem 4.12). Given a kernel $\mathbf{K}$, evaluated over a set of points $\mathcal{X}$, the empirical Rademacher complexity of the class $\mathcal{F}_B$ satisfies

$$\hat{\mathcal{R}}(\mathcal{F}_B) \leq \frac{2B}{l}\sqrt{tr(\mathbf{K})}. \tag{6.6}$$

Equation 6.6 shows that the *empirical Rademacher complexity* is proportional to the trace of the kernel.

6.1.1 Local Rademacher complexity

In this part, the definition of the *local Rademacher complexity* will be introduced. This measure of complexity takes into account only the function that will be *probably* chosen by the learning algorithm by using the given sample. For this reason, we are able to obtain sharper (fast-rate) bounds than the global Rademacher analysis.

Definition 6.1.2 For any $r > 0$, the *local Rademacher complexity* of a class of functions $\mathcal{F}$ is:

$$\mathcal{R}_l(\mathcal{F}; r) = \mathcal{R}_l(\mathcal{F})(\{f \in \mathcal{F} : \mathbb{E}[f^2] \leq r\}). \tag{6.7}$$

A similar definition can be used for the *empirical local Rademacher complexity* by exploiting the *empirical Rademacher complexity*:

Definition 6.1.3 For any $r > 0$, the *local Rademacher complexity* of a class of functions $\mathcal{F}$ is:

$$\hat{\mathcal{R}}_l(\mathcal{F}; r) = \hat{\mathcal{R}}_l(\mathcal{F})(\{f \in \mathcal{F} : \mathbb{E}[f^2] \leq r\}). \tag{6.8}$$

A connection exists with respect to the kernels also by dealing with the *local Rademacher complexity*. In fact, the following theorem holds:

Theorem 6.1.3 ([47], Theorem 3) Given a kernel function K with eigenvalues in descending order $\{\lambda_i\}_{i=1}^{\infty}$, the local Rademacher complexity of the class $\mathcal{F}_{B=1} =: \mathcal{F}$ satisfies

$$\mathbb{E}[\mathcal{R}_l(\mathcal{F};r)] \leq \sqrt{\frac{2}{l}\sum_{i=1}^{l}\min(r,\lambda_i)}. \tag{6.9}$$

From this theorem, we can note that the *local Rademacher complexity* for kernels can be determined by the tail sum of the eigenvalues.

II A new approach to Multiple Kernel Learning

Motivation

Multiple Kernel Learning (MKL, see Section 3.3.3) is a popular paradigm used to learn kernels. The kernel computed by these techniques are (generally linear) combinations of previously defined *weak* kernels. The main rationale behind this kind of methods is that they can alleviate the user's effort on defining good kernels for a given problem. Using the MKL framework, the algorithm itself is able to select the best combination among a battery of predefined and reasonable weak kernels.

According to [79] the majority of existing MKL approaches can be divided into the following two categories. *Fixed or Heuristic rule* and *Optimization based* approaches.

Generally speaking, best performing approaches to MKL necessitate of complex optimization including semidefinite programming (SDP), or quadratically constrained quadratic programming (QCQP), just to name a few, thus making these approaches unpractical when the number of kernels to be combined is large, say in the order of hundreds. Furthermore, these methods often need to keep in memory the whole set of weak kernels, or at least the submatrices corresponding to the training data pairs. This will exhaust the memory soon when coping with hundreds of examples and hundreds of weak kernels.

The MKL idea can be used in different scenarios. In the first and more popular case, a small number of strong kernels have to be combined each representing a different (possibly orthogonal) view of the same task. Typically these kernels are individually well designed by experts and their optimal combination hardly leads to a significant improvement of the performance with respect to, for example, a simple averaging combination. Alternatively, the MKL paradigm can be exploited to combine an optimized family of *weak* kernels created to obtain the best performance of the MKL algorithm.

A first possibility is to generate a very large set of random *weak* kernels aiming at boosting their accuracy. In this way, the final combination will represent a weighted combination of the different subsets of features. At least in principle, it can boost the performance basically performing a data-driven feature learning/selection/weighting.

With this possibility in mind, we see that for MKL to be effective we need to combine many kernels. For this reason, being non scalable becomes a stringent issue for a MKL method. On the other side, simpler fixed rule algorithms are definitely more scalable in general but they are far less flexible and tend to become ineffective when coping with many kernels and noise. In Chapter 7, we propose a very efficient algorithm, called *EasyMKL*, which is able to cope with thousands of kernels efficiently and we empirically demonstrate the effectiveness of the methods proposed that is almost always more accurate than the baselines especially when noise is present in the weak kernels.

Beyond this possibility, we can create automatically the best family of *weak* kernels by exploiting a hierarchical structure of the features, in order to obtain a set of kernels

with different levels of abstraction. In literature, some possibilities have been already tested. The most important are the Hierarchical Kernel Learning (HKL) algorithms [13] and their evolution: the Generalized HKL (gHKL) [105], presented in Section 3.3.3. These methods perform a non-linear feature selection exploiting the assumption that it is possible to generate a DAG from the features. Each node of the DAG represents a *weak* kernel and a hierarchical structure is imposed to obtain a sparse combination of them.

However, a fundamental aspect was not taken into account until this point: the expressiveness of the *weak* kernels. We are interested in finding the connection between the performance of MKL and the expressiveness of the given family of *weak* kernels. Firstly, we need a correct measure of the expressiveness of the *weak* kernels family. In this context, some questions arise: how can we evaluate the expressiveness of a kernel (or a family of kernels)? Is the expressiveness of a family of kernels important in order to achieve better results applying MKL?

The natural way to create a measure of expressiveness of a kernel is to define a measure that take into account the *empirical Rademacher complexity* of the linear function (with bounded norm) of the RKHS:

$$\mathcal{F}_B = \{\mathbf{x}_j \rightarrow \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}_j) : \|\mathbf{w}\|_2 \leq B\}, \tag{6.10}$$

where $\boldsymbol{\phi}$ is the feature mapping for the kernel $\mathbf{K}$. We will base our observations starting from a famous bound theorem for the complexity of $\mathcal{F}_B$. This theorem ([172], Theorem 4.12) shows how the *empirical Rademacher complexity* $\hat{\mathcal{R}}$ is proportional to the trace of the kernel:

$$\hat{\mathcal{R}}(\mathcal{F}_B) \leq \mathcal{O}(B\sqrt{tr(\mathbf{K})}). \tag{6.11}$$

From these results, in Chapter 8, we will define a new measure for the kernel complexity called *spectral complexity*. In practice, we can consider the expressiveness of a kernel as an estimation of the number of possible labelings that a kernel machine can handle using that kernel as measure of similarity.

The designing of a measure of kernel expressiveness is useful in order to create families of *weak* kernels using a principled methodology. In Chapter 9, we will construct family of *weak* kernels defining a new property, exploiting the definition of spectral complexity for the family of Dot-Product Polynomials (DPP) kernels and for graph-kernels.

We will prove that it is possible to generate families of *weak* kernels with increasing spectral complexity, by defining a property on the monotonicity of the spectral complexity. Several experiments will highlight that using these families of *weak* kernels improves the performance of EasyMKL.

Also, our results are generalizable to all the dot-product kernels of the form

$$K(x,z) = f(x \cdot z), \tag{6.12}$$

where $f : \mathbb{R} \to \mathbb{R}$, such that it defines a kernel. From this hypothesis, f can be written using the Maclaurin expansion as a DPPs with positive coefficients (i.e. our results are not limited to the polynomial kernels). In fact, the wide class of dot-product kernels contains commonly used kernels, like the polynomial kernel and RBF kernel.

Finally, in Chapter 10, we will deal with the combination of a family of *weak* kernels in a different MKL framework. We will combine the family as a element-wise product of the *weak* kernels:

$$\mathbf{K} = \bigotimes_{r=1}^{R} \mathbf{K}_r. \tag{6.13}$$

We will create an optimal family of *weak* kernels for this fixed combination rule. Specifically, we combine a set of RBF kernels evaluated using a single feature for each kernel. Using this methodology, we are learning the parameter of an Anisotropic RBF kernel (ARBF), i.e. a generalization of the standard RBF kernel.

The Python code of our algorithms can be downloaded from the following website:
`https://github.com/jmikko/EasyMKL/`

7. EasyMKL

The goal of Multiple Kernel Learning (MKL) is to combine kernels derived from multiple sources in a data-driven way with the aim to enhance the accuracy of a target kernel machine. State-of-the-art methods of MKL have the drawback that the time required to solve the associated optimization problem grows (typically more than linearly) with the number of kernels to combine. Moreover, it has been empirically observed that even sophisticated methods often do not significantly outperform the simple average of kernels. In this chapter, we propose a time and space efficient MKL algorithm that can easily cope with hundreds of thousands of kernels and more. The proposed method has been compared with other baselines (random, average, ecc.) and three state-of-the-art MKL methods showing that our approach is often superior. We empirically show that the advantage of using the method proposed in this chapter is even clearer when noise features are added. Finally, we have analyzed how our algorithm performances change with respect to the number of examples in the training set and the number of kernels combined.

A summary of this chapter is the following. In Section 7.1 the proposed multiple kernel learning algorithm, namely EasyMKL, is presented. In Section 7.2, experiments are presented with respect to different dimensions, namely, the effectiveness, the scalability, and the stability of the method proposed against different baselines and state-of-the-art methods. Finally, in Section 7.3 we draw the conclusions.

Part of this chapter has been published in [2, 4][1].

[1] Python implementation of EasyMKL: `https://www.researchgate.net/publication/280640138_EasyMKL_-_Python_Source`

7.1 A new scalable MKL algorithm

In MKL we want to find the best combination parameters for a set of predefined kernel matrices. In our context, this is done by learning a vector of coefficients $\boldsymbol{\eta}$ that forms the combined kernel according to:

$$\mathbf{K} = \sum_{r=1}^{R} \eta_r \mathbf{K}_r, \;\; \eta_r \geq 0. \tag{7.1}$$

Clearly, we must restrict the possible choices of such a matrix $\mathbf{K}$ and this can be made by regularizing the learning process. So, we pose the problem of learning the kernel combination as a min-max problem over variables $\boldsymbol{\gamma}$ and $\boldsymbol{\eta}$. Specifically, we propose to maximize the distance between positive and negative examples with a unitary norm vector $\boldsymbol{\eta}$ as the *weak* kernel combination vector, that is

$$\max_{\boldsymbol{\eta}:||\boldsymbol{\eta}||_2=1} \min_{\boldsymbol{\gamma}\in\Gamma} \underbrace{(1-\Lambda)\boldsymbol{\gamma}^\top \hat{\mathbf{Y}}(\sum_{r}^{R} \eta_r \hat{\mathbf{K}}_r)\hat{\mathbf{Y}}\boldsymbol{\gamma} + \Lambda||\boldsymbol{\gamma}||_2^2}_{Q(\boldsymbol{\eta},\boldsymbol{\gamma})}. \tag{7.2}$$

Considering $\mathbf{d}(\boldsymbol{\gamma})$ the vector with the r^{th} entry defined as

$$d_r(\boldsymbol{\gamma}) = \boldsymbol{\gamma}^\top \hat{\mathbf{Y}}\hat{\mathbf{K}}_r\hat{\mathbf{Y}}\boldsymbol{\gamma}, \tag{7.3}$$

we obtain that $Q(\boldsymbol{\eta},\boldsymbol{\gamma})$ is equal to $(1-\Lambda)\boldsymbol{\eta}^\top \mathbf{d}(\boldsymbol{\gamma}) + \Lambda||\boldsymbol{\gamma}||_2^2$ and we can rewrite the original problem in Equation 7.2 as

$$\min_{\boldsymbol{\gamma}\in\Gamma} \max_{\boldsymbol{\eta}:||\boldsymbol{\eta}||_2=1} Q(\boldsymbol{\eta},\boldsymbol{\gamma}) = \min_{\boldsymbol{\gamma}\in\Gamma} \max_{||\boldsymbol{\eta}||_2=1} (1-\Lambda)\boldsymbol{\eta}^\top \mathbf{d}(\boldsymbol{\gamma}) + \Lambda||\boldsymbol{\gamma}||_2^2. \tag{7.4}$$

Now, it is possible to see that the vector $\boldsymbol{\eta}^*$ maximizing the function $Q(\boldsymbol{\eta},\boldsymbol{\gamma})$ above has a simple analytic solution:

$$\boldsymbol{\eta}^* = \frac{\mathbf{d}(\boldsymbol{\gamma})}{||\mathbf{d}(\boldsymbol{\gamma})||_2}. \tag{7.5}$$

Plugging this solution into the min-max problem, we obtain

$$\min_{\boldsymbol{\gamma}\in\Gamma} Q(\boldsymbol{\eta}^*,\boldsymbol{\gamma}) = \min_{\boldsymbol{\gamma}\in\Gamma} (1-\Lambda)||\mathbf{d}(\boldsymbol{\gamma})||_2 + \Lambda||\boldsymbol{\gamma}||_2^2. \tag{7.6}$$

The optimal $\boldsymbol{\gamma}$ will be the (regularized) minimizer of the 2-norm of the vector of distances. This minimizer is not difficult to find as this is a convex function though not quadratic. In order to simplify the problem further we prefer to minimize an upper-bound instead

corresponding to the 1-norm of the vector of distances, thus obtaining

$$\min_{\boldsymbol{\gamma}\in\Gamma}(1-\Lambda)||\mathbf{d}(\boldsymbol{\gamma})||_1+\Lambda||\boldsymbol{\gamma}||_2^2=\min_{\boldsymbol{\gamma}\in\Gamma}(1-\Lambda)\boldsymbol{\gamma}^\top\hat{\mathbf{Y}}(\sum_r^R\hat{\mathbf{K}}_r)\hat{\mathbf{Y}}\boldsymbol{\gamma}+\Lambda||\boldsymbol{\gamma}||_2^2. \tag{7.7}$$

Interestingly, the obtained minimization problem is the same as the KOMD problem, introduced in Chapter 5, where the kernel matrix has been replaced with the simple sum of the *weak* kernel matrices. This will turn out to be very important for the efficiency (especially in terms of space) of the method as will be explained later on. In the following, we refer to this algorithm as EasyMKL.

From a more theoretical point of view, modifying the original problem using the 1-norm upper-bound, in fact, we are changing the optimal solution $\boldsymbol{\eta}^*$ in the initial problem (in Equation 7.4) with a new $\widetilde{\boldsymbol{\eta}^*}$, where

$$\widetilde{\boldsymbol{\eta}^*}=\boldsymbol{\eta}^*\frac{\|\mathbf{d}(\boldsymbol{\gamma})\|_1}{\|\mathbf{d}(\boldsymbol{\gamma})\|_2}. \tag{7.8}$$

Consequently, the problem that we are solving is

$$\min_{\boldsymbol{\gamma}\in\Gamma}(1-\Lambda)\widetilde{\boldsymbol{\eta}^*}\mathbf{d}(\boldsymbol{\gamma})+\Lambda||\boldsymbol{\gamma}||_2^2=\min_{\boldsymbol{\gamma}\in\Gamma}(1-\Lambda)\boldsymbol{\eta}^*\frac{\|\mathbf{d}(\boldsymbol{\gamma})\|_1}{\|\mathbf{d}(\boldsymbol{\gamma})\|_2}\mathbf{d}(\boldsymbol{\gamma})+\Lambda||\boldsymbol{\gamma}||_2^2. \tag{7.9}$$

From this new formulation, we note that, in fact, we have just added a multiplicative coefficient $\frac{\|\mathbf{d}(\boldsymbol{\gamma})\|_1}{\|\mathbf{d}(\boldsymbol{\gamma})\|_2}$ to the original optimal solution $\boldsymbol{\eta}^*$. From the *Hölder's inequality* this coefficient is bounded by the number of kernels and we have that:

$$1\leq\frac{\|\mathbf{d}(\boldsymbol{\gamma})\|_1}{\|\mathbf{d}(\boldsymbol{\gamma})\|_2}\leq\sqrt{R}. \tag{7.10}$$

The value of the ratio between the 1-norm and the 2-norm tends to 1 if the vector $\mathbf{d}(\boldsymbol{\gamma})$ is very sparse, while, it tends to $\sqrt{R}$ when the values in the vector $\mathbf{d}(\boldsymbol{\gamma})$ are similar. In other words, using the proposed formulation, we are promoting sparse solutions of the vector $\mathbf{d}(\boldsymbol{\gamma})$ solving the minimum problem. Then, solving the problem in Equation 7.9, the vector of the weight $\boldsymbol{\eta}^*=\frac{\mathbf{d}(\boldsymbol{\gamma})}{\|\mathbf{d}(\boldsymbol{\gamma})\|_2}$ will result sparser than the solution of the original problem. Clearly, the problem in Equation 7.9 has the same solution as the EasyMKL minimum problem in Equation 7.7.

A final consideration we can do here is that the quality of a kernel does not change if it is multiplied by a positive constant. However, our formulation makes particularly clear that, if kernels with different traces are present in the combination, they have unequal impact in the MKL optimization problem. In fact, most of the bounds related to the difference between the true error and the empirical margin error (i.e. estimation error)

change linearly with respect to the square root of the maximal trace among the combined kernels. Moreover, if we have that $\exists T > 0$ such that

$$T^2 \geq K_r(x,x) \;\; \forall r = 1,...,R \;\; \forall x \tag{7.11}$$

the estimation error (fixed a specific task and without constants) is $\mathcal{O}(T)$ [51, 101]. Thus, if not differently motivated, different traces should be avoided.

7.2 Experiments and Results

In this section, the experiments we have performed to validate the EasyMKL approach (i.e. solving the optimization problem in Equation 7.7) are presented. First of all, an introduction of the experimental setting is given in Section 7.2.1. In Section 7.2.2 we show some results confirming that the maximization of the separation in feature space is a good criterion to pursue to obtain effective kernels. Moreover, an overview of the baselines and state-of-the-art methods is given in Section 7.2.3. In Section 7.2.4 we compare our algorithm against other methods with respect to the AUC over ranking tasks. In Section 7.2.5 we perform experiments concerning time and memory performance of EasyMKL. We tested the stability of EasyMKL by adding different amounts of noise in the data in Section 7.2.6. In Section 7.2.6 we analyze how the cardinality of the training set influences our algorithm. Finally, in Section 7.2.7 we describe experiments that we have performed to understand the difference in performance obtained by changing the number of *weak* kernels.

7.2.1 Experimental setting

Experiments have been performed comparing the proposed methods with other state-of-the-art MKL algorithms with respect to accuracy and computational performance. A total of 7 benchmark datasets with different characteristics and 7 different MKL methods have been considered. Namely, the datasets considered are*Diabetes*, *Australian*, *German*, *Splice*, *Batch2*, *Mush*, *Gisette*, that are summarized in Table 7.1 (see Section 2.1.2 for further details). Data have been scaled to the$[-1,+1]$ interval and the number of training examples selected for the training part is approximately 10% of the dataset. We decided to use relatively small training sets obtaining a final accuracy less influenced by the classifier (on which the combined kernel is applied) and more influenced by the real quality of a kernel. Further, we preferred having larger test sets and many different splits of each single dataset aiming at improving the significance of the overall evaluation.

In this work we want to demonstrate that we can efficiently perform kind of feature selection/weighting via MKL and this can be done by selecting a very large set of *weak*

Data set	Source	Features	Examples	N_{tr}
Diabetes	*UCI*	8	768	77
Australian	*Statlog*	14	690	69
German	*Statlog*	24	1000	100
Splice	*UCI*	60	1000	100
Batch2	*UCI*	128	1244	124
Mush	*UCI*	112	2000	200
Gisette	*NIPS*03	5000	13500	1350

Table 7.1: Datasets information: name, source, number of features, number of examples and cardinality of the training set.

kernels each one individually defined using a small subset of the original features.

Let $d \in \mathbb{N}$ and $\beta > 0$ two parameters, then the r^{th} kernel is constructed by random picking of a bag (replica are allowed) of features F_r, such that $|F_r| \leq d$, and constructing an RBF based *weak* kernel defined according to

$$\mathbf{K}_r[i,j] = \prod_{f \in F_r} e^{-\frac{\beta}{|F_r|}(\mathbf{x}_i^{(f)} - \mathbf{x}_j^{(f)})^2}. \tag{7.12}$$

Note that, using Equation 7.12 we are generating a family of *weak* kernels containing a large number of kernels with poor performance. The original information contained in the raw dataset is maintained but it is shuttered, fragmented and then randomly distributed among the *weak* kernels. The goal of our algorithm is to combine this information in the best way possible, creating a sort of *deeper* kernel. This new kernel has to be able to highlight (implicitly) which non linear combination, of the original features, is more important.

7.2.2 Is data separation a good criterion to maximize?

In this section, we present results showing that the maximization of the distance between positive and negative examples is a good criterion to pursue as it is positively correlated with the AUC obtained in a binary ranking task.

In particular, we designed a simple greedy algorithm which constructs a series of kernels with monotonically increasing data separation. For reader's convenience, we recall the formula of separation:

$$\min_{\boldsymbol{\gamma} \in \hat{\Gamma}} \boldsymbol{\gamma}^\top \hat{\mathbf{Y}} \hat{\mathbf{K}} \hat{\mathbf{Y}} \boldsymbol{\gamma}. \tag{7.13}$$

The algorithm starts with a null kernel and considers sequentially the available *weak*

kernels. At each iteration, the current kernel is updated with a *weak* kernel only if this addition increases the separation of training data. Note that, this algorithm corresponds to have $\boldsymbol{\eta}_r \in \{0,1\}$. The algorithm is described in detail in Algorithm 1.

Algorithm 1 A greedy MKL algorithm which aims at greedily maximizing the separation of positive and negative examples in the training set.

Require: $\Lambda \in [0,1]$
Ensure: A kernel matrix $\mathbf{K}^*$ and a vector $\boldsymbol{\gamma}^*$
1: $\mathbf{K}^* = \mathbb{O}$, $c = 1$, $d^* = 0$
2: **for** $r = 1$ **to** r_{max} **do**
3: pick a new $\mathbf{K}_r$ such that $\text{tr}(\mathbf{K}_r) = k$
4: set $\mathbf{K} = \frac{1}{c+1}(c\mathbf{K}^* + \mathbf{K}_r)$
5: set $\boldsymbol{\gamma}^* = \text{KOMD}(\hat{\mathbf{K}}, \Lambda)$
6: compute current distance $d = \boldsymbol{\gamma}^{*\top}\hat{\mathbf{Y}}\hat{\mathbf{K}}\hat{\mathbf{Y}}\boldsymbol{\gamma}^*$
7: **if** $d > d^*$ **then**
8: $d^* = d$, $\mathbf{K}^* = \frac{1}{c+1}(c\mathbf{K}^* + \mathbf{K}_r)$, $c = c+1$
9: **end if**
10: **end for**
11: **return** $\mathbf{K}^*$, $\boldsymbol{\gamma}^* = \text{KOMD}(\hat{\mathbf{K}}^*, \Lambda)$

In Figure 7.1, an example of a plot obtained applying the greedy algorithm on the *Splice* dataset, using $10,000$ *weak* kernels and generated as described in Section 7.2.1($d = 5$), is given. This experiment clearly shows the strong correlation between the data separation and the AUC obtained in the test set using the produced kernel.

7.2.3 Methods

We have then considered three baseline methods for our experiments. The first method **SVM** is the classical SVM trained with all the features. This is reported for the sake of completeness and to give an idea of the difficulty of the datasets. Note that, the feature space in this case is different from MKL methods and we can consider these results as a very strong baseline. The second method, called Random Kernel (**Random**), consists of random picking from available kernels, which is equivalent to set only one η_r equal to 1 and all the others equal to 0. We decided to add this baseline as an indicator of how much information is brought from single kernels. Intentionally, the performance of this baseline could be really poor. Finally, the last baseline method is the Average Kernel (**Average**) where the same weight η_r is given to all the available *weak* kernels. Despite its simplicity, it is known (see [201]) that this is a strong baseline that beats other more advanced techniques quite often and can obtain very good results. We have then considered three state-of-the-art algorithms of MKL with SVM: *Simple MKL* (SMKL) [157], *Generalized MKL* (GMKL) [189], *Lasso-based MKL* (GLMKL) [117, 202]. All these three algorithms

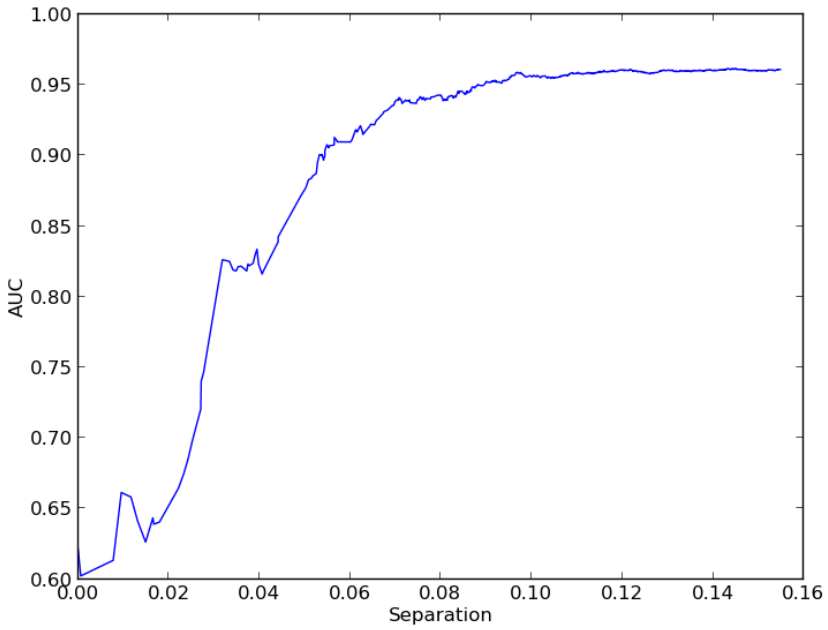


Figure 7.1: Value of the quality function (Separation) versus the AUC obtained using the greedy algorithm over the *Splice* dataset.

are in the optimization based family of MKL methods. A MATLAB implementation [2] of these methods by Mehmet Gönen and Ethem Alpaydin [79] has been used. Moreover, a very efficient MKL method called *SMO Projected Gradient Descent Generalized MKL* (SPG-GMKL) [104] has been used in order to compare che computational complexity in time and memory.

The validation of the methods was performed before each type of experiment. A subset of examples (validation set) was selected for each method and dataset and was only used to select the best hyperparameters. The kernel machines (SVM and KOMD) were validated using the same subset of data to obtain a fair comparison. The performance of SVM and KOMD were similar in most of the cases, thus confirming the findings in [6].

7.2.4 AUC comparisons

The quality of the different combined kernels has been evaluated by means of AUC (Area Under Curve), which measures how good the induced ranking order is with respect to the target classification task. In this way, we do not necessitate of a threshold setting which is an additional, and useless, degree of freedom when evaluating kernels performance. Weak kernels have been generated according to the method depicted in Section 7.2.1. A number of $10,000$ *weak* kernels have been constructed for each dataset. Two different values for d,

[2] `http://www.cmpe.boun.edu.tr/~gonen/mkl`.

Average AUC (RBF all features)			
Algorithm	*Diabetes*	*Australian*	*German*
SVM	80.39%	91.37%	70.79%

Average AUC (RBF all features)			
Algorithm	*Splice*	*Batch2*	*Mush*
SVM	86.12%	95.20%	98.52%

Table 7.2: AUC results on three datasets of SVM (RBF all features).

that is $d \in \{5, 10\}$, have been used in the generation algorithm with the RBF parameter β_0 obtained by model selection on a standard SVM. The experimental setting is summarized below:

1. Repeat for $r = 1$ to $R = 10,000$: pick a random $p \in \{1, ..., d\}$ and generate a *weak* kernel K_r using p features picked randomly (with replacement).
2. Combine $\{\mathbf{K}_r\}_{r=1}^{R}$ with a MKL algorithm to obtain a kernel $\mathbf{K}$.
3. Rank test examples using $\mathbf{K}$ and SVM (KOMD for the proposed method) and evaluate the quality of the ranking with the AUC metric.

The same experiments have been repeated 1,000 times averaging the AUC results in order to improve the significance of the evaluation. Aiming at setting our experiments as fair as possible, the KOMD algorithm has been used with the kernel produced by our MKL method, while SVM has been used on all the others. For what concerns the SVM baseline, RBF kernels with all the features have been used in this case. The obtained results are summarized in Table 7.2 .

The results obtained with the MKL algorithms are summarized in Tables 7.3 and 7.4 (for $d = 5$ and $d = 10$) showing that standard MKL methods do not have performances significantly better than the simple kernel averaging. On the other side, EasyMKL has significantly better AUC in four out of six datasets. Also, EasyMKL has a small standard deviation (std) with respect to the other methods and this fact highlights the stability of our algorithm. This trend is confirmed in Table 7.5 where the proposed method is always significantly better than the average baseline. Unfortunately, given the relatively large number of examples of the *Gisette* dataset, it was not possible to run state-of-the-art MKL methods on this data without running out the memory.

7.2.5 Performance in time and memory

We have also performed experiments[3] to evaluate the computational time and the memory used by EasyMKL. We compared our algorithm with a state-of-the-art algorithm called

[3]For these experiments we used a CPU Intel Core i7-3632QM @ 2.20GHz 2.20GHz

Average AUC (RBF subset $d = 5$)			
Algorithm	*Diabetes*	*Australian*	*German*
Random	$62.13_{\pm 6.20}\%$	$65.56_{\pm 21.32}\%$	$49.59_{\pm 11.68}\%$
Average	$63.01_{\pm 0.41}\%$	$92.09_{\pm 0.51}\%$	$\mathbf{73.04_{\pm 0.37}}\%$
SMKL	$77.52_{\pm 2.11}\%$	$91.64_{\pm 3.18}\%$	$69.71_{\pm 2.78}\%$
GMKL	$75.43_{\pm 3.23}\%$	$90.37_{\pm 4.01}\%$	$69.53_{\pm 3.87}\%$
GLMKL	$70.95_{\pm 2.75}\%$	$85.27_{\pm 3.56}\%$	$68.61_{\pm 3.02}\%$
EasyMKL	$\mathbf{79.91_{\pm 0.13}}\%$	$\mathbf{92.17_{\pm 0.11}}\%$	$72.71_{\pm 0.21}\%$

Average AUC (RBF subset $d = 5$)			
Algorithm	*Splice*	*Batch2*	*Mushrooms*
Random	$52.61_{\pm 10.22}\%$	$81.14_{\pm 11.83}\%$	$28.58_{\pm 24.56}\%$
Average	$86.56_{\pm 0.25}\%$	$95.35_{\pm 0.43}\%$	$\mathbf{98.45_{\pm 0.49}}\%$
SMKL	$83.48_{\pm 2.15}\%$	$94.82_{\pm 2.18}\%$	$97.49_{\pm 3.41}\%$
GMKL	$83.70_{\pm 4.24}\%$	$92.53_{\pm 3.99}\%$	$97.39_{\pm 4.89}\%$
GLMKL	$85.36_{\pm 1.98}\%$	$94.54_{\pm 3.34}\%$	$96.38_{\pm 4.72}\%$
EasyMKL	$\mathbf{87.87_{\pm 0.18}}\%$	$\mathbf{98.98_{\pm 0.21}}\%$	$97.91_{\pm 0.24}\%$

Table 7.3: $AUC_{\pm std}$ results on three datasets of various MKL methods and baselines (RBF feature subset $d = 5$).

Average AUC (RBF subset $d = 10$)			
Algorithm	*Diabetes*	*Australian*	*German*
Random	$71.44_{\pm 7.05}\%$	$70.36_{\pm 12.59}\%$	$56.93_{\pm 7.14}\%$
Average	$71.94_{\pm 0.34}\%$	$92.11_{\pm 0.42}\%$	$\mathbf{73.33_{\pm 0.36}}\%$
SMKL	$75.21_{\pm 3.41}\%$	$87.94_{\pm 2.12}\%$	$70.92_{\pm 2.56}\%$
GMKL	$72.01_{\pm 5.76}\%$	$86.88_{\pm 3.17}\%$	$70.89_{\pm 3.01}\%$
GLMKL	$71.85_{\pm 5.01}\%$	$84.80_{\pm 2.09}\%$	$69.75_{\pm 2.78}\%$
EasyMKL	$\mathbf{79.61_{\pm 0.16}}\%$	$\mathbf{92.17_{\pm 0.11}}\%$	$73.21_{\pm 0.11}\%$

Average AUC (RBF subset $d = 10$)			
Algorithm	*Splice*	*Batch2*	*Mushrooms*
Random	$56.76_{\pm 8.52}\%$	$85.30_{\pm 8.24}\%$	$44.58_{\pm 37.26}\%$
Average	$90.42_{\pm 0.34}\%$	$94.70_{\pm 0.53}\%$	$\mathbf{98.86_{\pm 0.19}}\%$
SMKL	$86.84_{\pm 3.42}\%$	$95.07_{\pm 4.87}\%$	$97.48_{\pm 2.37}\%$
GMKL	$84.48_{\pm 6.02}\%$	$94.92_{\pm 6.97}\%$	$96.99_{\pm 4.32}\%$
GLMKL	$86.41_{\pm 5.51}\%$	$94.18_{\pm 6.09}\%$	$95.25_{\pm 3.12}\%$
EasyMKL	$\mathbf{91.19_{\pm 0.18}}\%$	$\mathbf{97.08_{\pm 0.16}}\%$	$98.29_{\pm 0.12}\%$

Table 7.4: $AUC_{\pm std}$ results on three datasets of various MKL methods and baselines (RBF feature subset $d = 10$).

Algorithm	RBF $d = 5$	RBF $d = 10$	RBF $d = 20$
Average	$95.52_{\pm 0.15}\%$	$94.72_{\pm 0.13}\%$	$93.65_{\pm 0.13}\%$
EasyMKL	$\mathbf{95.83}_{\pm \mathbf{0.12}}\%$	$\mathbf{95.40}_{\pm \mathbf{0.10}}\%$	$\mathbf{94.64}_{\pm \mathbf{0.11}}\%$

Table 7.5: Average $AUC_{\pm std}$ on the *Gisette* (NIPS2003 feature selection challenge) of EasyMKL (with RBF subset) and comparison with the Average baseline.

SPG-GMKL [104]. A C++ implementation of SPG-GMKL provided by the authors[4] has been used. SPG-GMKL with this implementation is more than 100 times faster than Simple MKL [104]. The *Splice* dataset has been selected for this experiment with 100 training examples. A variable number of RBF kernels has been generated with a parameter β picked randomly in $(0,1)$ and using all the 60 features of the *Splice* dataset. We have studied the performance in time and memory, fixed an upper bound of 2 GB for the memory. Results are reported in Figure 7.2.

Based on our experiments, the time complexity of SPG-GMKL is linear with the number of kernels, with a constant of 0.15 seconds per kernel. EasyMKL has a linear increase in time too, with 0.17 seconds per kernel. The memory used by SPG-GMKL has a sub-linear growth with the number of kernels and has reached 2 GB with only 400 kernels. Viceversa, the memory used by our algorithm is independent from the number of kernels to combine. This is due to the fact that the optimization in our algorithm consists of a simple KOMD optimization on the sum of kernels. The computation of this sum of matrices can be easily implemented incrementally and only two kernel matrices need to be stored in memory. It follows that, with our method, we can use an *unlimited* number of different kernels with only a small memory requirement (e.g. 47 MB for the *Splice* dataset).

7.2.6 Stability with noisy features

In this set of experiments we injected artificial noise features, that is, features uncorrelated with the label of the example, to study the robustness of EasyMKL with respect to the Average Kernel method.

Similarly to [82], given a dataset of L examples, each new noise feature h is created using three simple steps:

1. Pick a feature f randomly from the set of the original features.
2. Let $\mathcal{F} = \{x_i^{(f)} : i = 1,...,L\}$ be the list of values of f (with replica).
3. For each example x_i with $i = 1,...,L$, define the value of $x_i^{(h)}$ as a randomly picked value from $\mathcal{F}$.

Using this method we have created features with a distribution that is similar to the original features but uncorrelated with the labels (i.e. permutations of the feature val-

[4] http://www.cs.cornell.edu/∼ashesh/pubs/code/SPG-GMKL/download.html

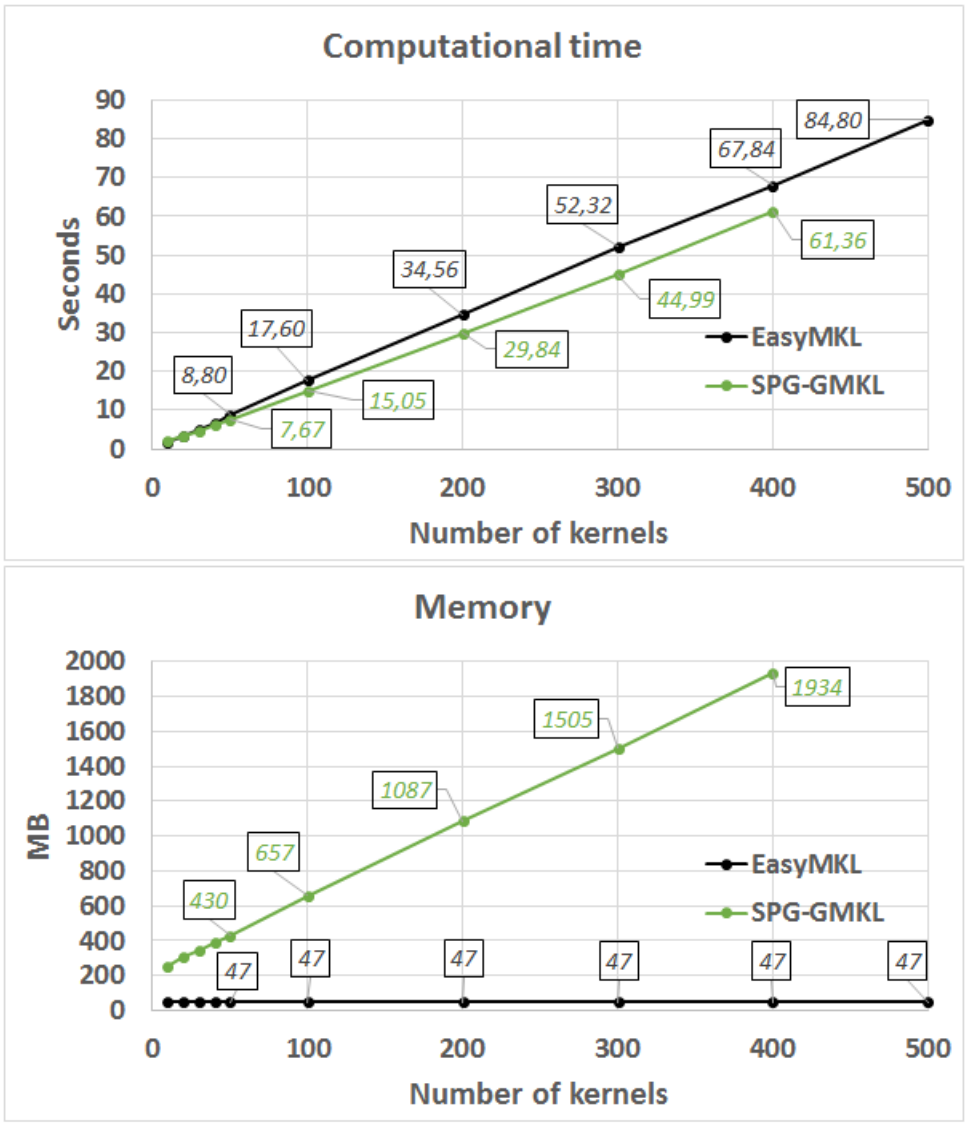


Figure 7.2: Time and memory used by SPG-GMKL and EasyMKL with different amounts of kernels combined. The kernels are created using all the features of the *Splice* dataset. The experiment with 500 kernels of SPG-GMKL is not reported as it exceeds the limit of 2 GB of memory.

ues). We have generated noise features in different percentages of the original features. With these new datasets we have repeated the same experiments described in Section 7.2.4 with the RBF *weak* kernels created using d equals to 5 and 10 and three datasets: *Splice*, *Batch2*, and *Mush*. We have fixed the percentages of noise features $p \in \{0, 50, 100, 150, ..., 900, 950, 1000\}$. Finally, we compared EasyMKL against the Average Kernel algorithm.

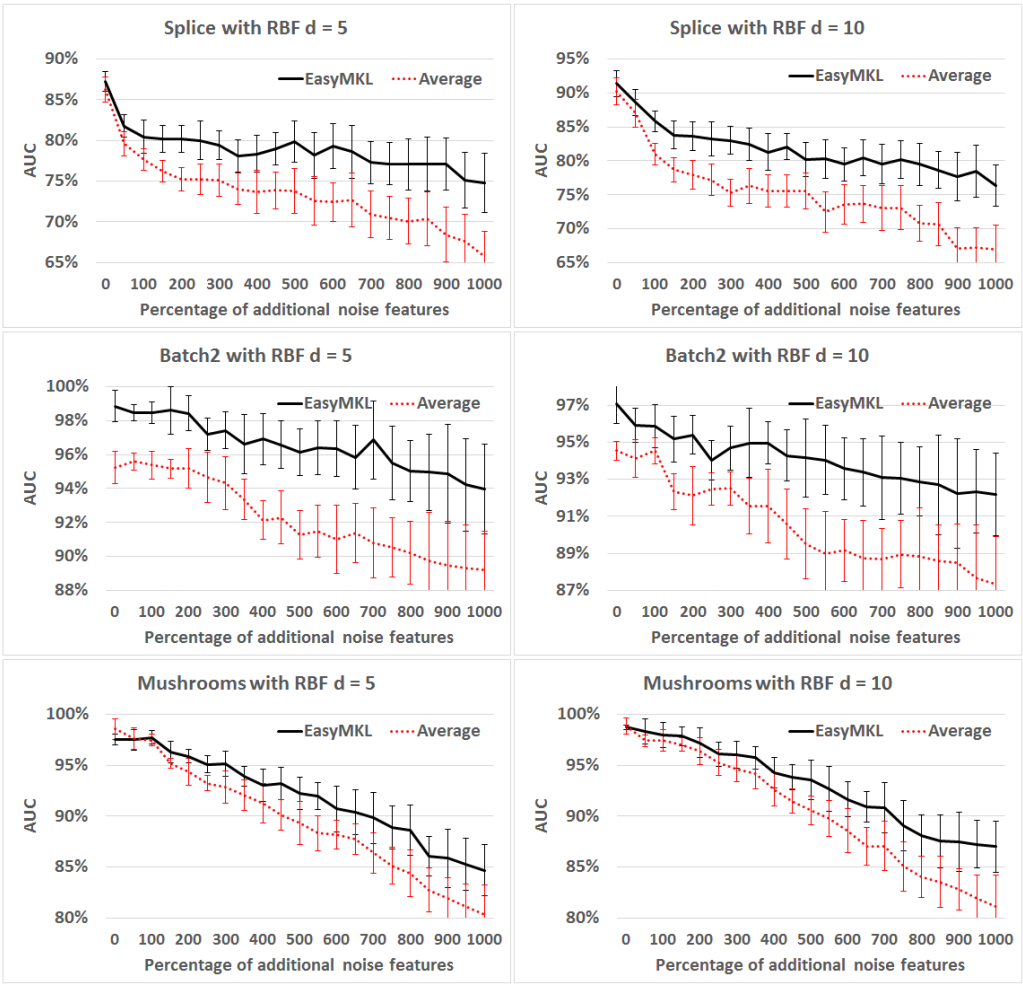


Figure 7.3: AUC comparisons with standard deviation of EasyMKL against the Average Kernel algorithm with respect to different percentages of additional noise features using different datasets (*Splice*, *Batch2* and *Mush*) and two different values of d (5 and 10).

The results are summarized in Figure 7.3. We can note that our algorithm obtains good results even with a $1,000\%$ of additional noise features and it outperforms the baseline algorithm. EasyMKL increases the gap from the baseline monotonically with respect to the increase of the percentage of noise features. However, quite surprisingly, these results confirm that the Average Kernel algorithm is still a very strong baseline.

Performance with respect to different cardinalities of the training set

In this section we have analyzed how the size of the training set influences the performance of EasyMKL. We compared our results against the Average Kernel method, that is used as baseline. We performed these experiments over the *Gisette* dataset using the same experimental setting as in Section 7.2.4 but with different sizes of the training set. Specifically, we have selected the size of the training set varying from 10 to 200 examples. For each size, the result reported in Figure 7.4 has been obtained evaluating the average on a large number of runs.

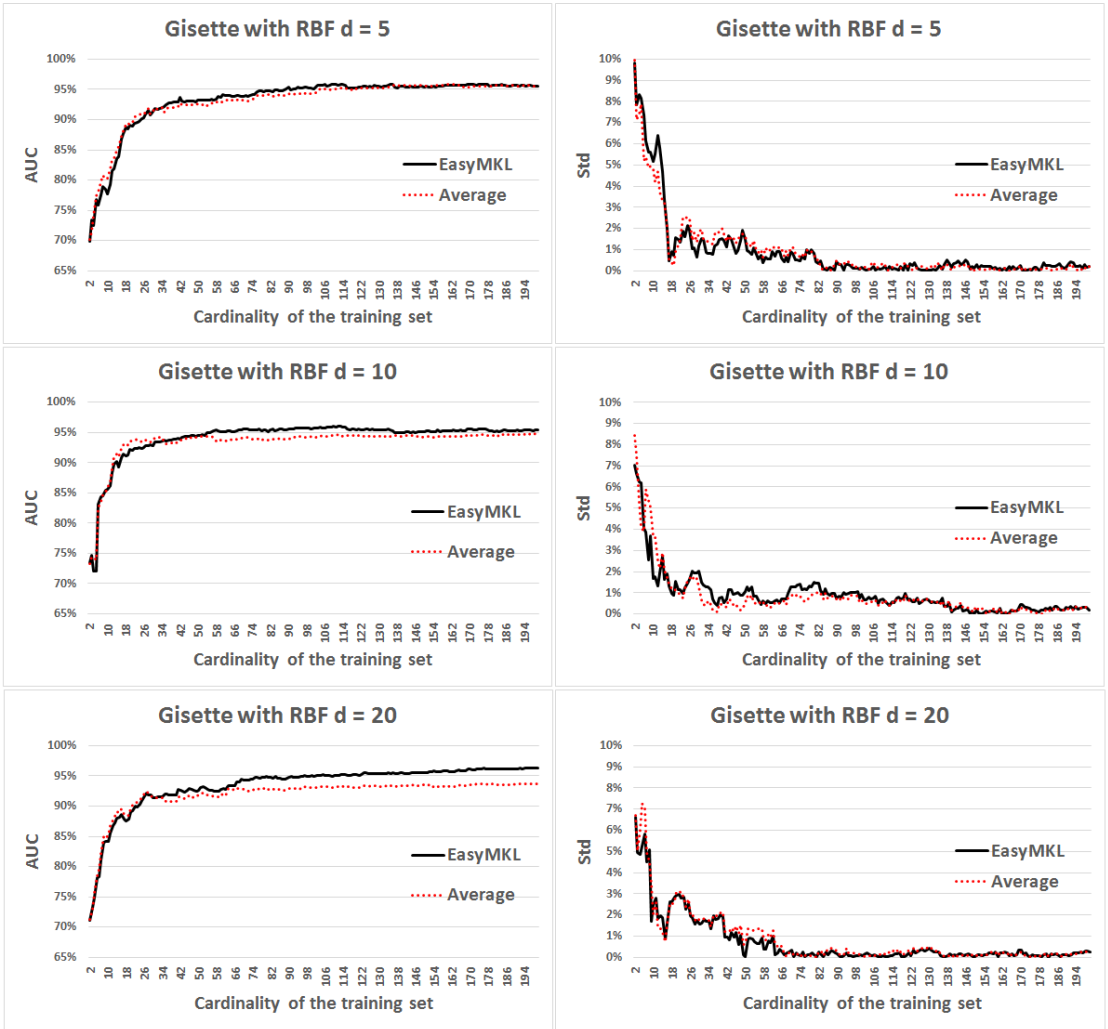


Figure 7.4: AUC comparisons (left figures) with standard deviation (right figures) of EasyMKL against the Average Kernel algorithm with respect to different cardinality of the training set using the *Gisette* dataset and three different values of d (5,10 and 20).

From these results, it is clear that our algorithm outperforms the Average Kernel algorithm when the training set is sufficiently large. This is due to the fact that EasyMKL effectively uses the training set to learn which features are more important (see experiments

in Section 7.2.6) and, when data are abundant, it can exploit this information to outperform the baseline. Not surprisingly, when the size of the training set is small the two algorithms perform similarly (and the standard deviation is too big to claim any result).

7.2.7 Performance changing the number of *weak* kernels

We also performed experiments analyzing the performance of EasyMKL against the Average Kernel method using different numbers of *weak* kernels. The kernels have been generated using the same setting described in Section 7.2.4. The number of generated kernels R was selected ranging from $R = 10$ to $R = 200$ (with a step of 10 kernels). We performed the experiments fixing $d = 5$ (i.e. very *weak* kernels) and using two datasets: *Splice* and *German*.

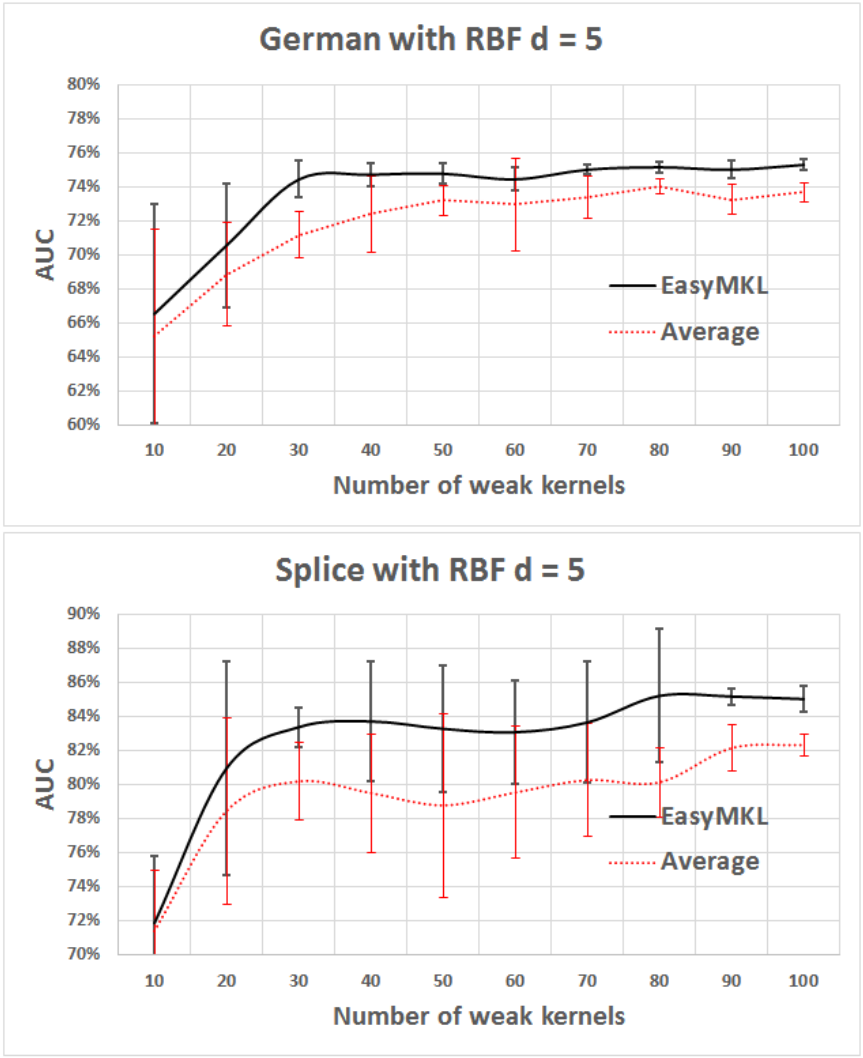


Figure 7.5: AUC comparisons with standard deviation of EasyMKL against the Average Kernel algorithm with respect to different number of *weak* kernels using the *German* dataset (24 features) and the *Splice* dataset (60 features) with $d = 5$.

The experiments have been repeated several times for each value of R and the results are reported in Figure 7.5. We can observe that EasyMKL outperforms the Average Kernel method and these experiments confirm that our algorithm also works well when combining a small number of *weak* kernels. In particular, EasyMKL obtains a steeper gain in performance with the first *weak* kernels (i.e. from 10 to 30 *weak* kernels).

7.3 Summary of the results

In this chapter we have proposed the EasyMKL algorithm which is able to cope efficiently with a very large number of different kernels. The experiments we reported have shown that the proposed method is more accurate than MKL state-of-the-art methods. We have also discussed time and memory requirements with respect to the number of combined kernels showing that EasyMKL uses only a constant amount of memory and it has only a linear time complexity. Finally, EasyMKL seems also quite robust with respect to the noise introduced by features which are not informative and works well even if used with a small number of *weak* kernels.

8. Spectral Complexity

In this chapter, a new measure of kernel complexity will be defined. This measure, called *spectral complexity*, is connected to the *empirical Rademacher complexity* and, for this reason, to the generalization error as well. The spectral complexity is related to the expressiveness of the family of linear functions with bounded norm implicitly defined by the kernel. Intuitively, this expressiveness can be seen as the quantity of different labelings that a family of functions is able to shatter.

8.1 A good measure for kernel complexity

Dealing with kernel complexity, we are interested in the expressiveness of a family of linear function in the Reproducing Kernel Hilbert Space (RKHS) defined by the kernel. These linear functions are in the form $\mathcal{F}_B = \{\mathbf{x}_j \to \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}_j) : \|\mathbf{w}\|_2 \leq B\}$ where $\boldsymbol{\phi}$ is the mapping from $\mathcal{X}$ to the RKHS. This definition highlights a connection between the measures for kernel complexity and $\hat{\mathcal{R}}$, i.e. the *empirical Rademacher complexity*. In fact, the following inequality holds:

$$\hat{\mathcal{R}}(\mathcal{F}_B) \leq \mathcal{O}(B\sqrt{tr(\mathbf{K})}). \tag{8.1}$$

Given a kernel matrix, a good complexity measure for kernels should be easy to compute and satisfies some properties. For example, a correct measure for kernel complexity should be invariant to the multiplicative constants. It is well know, and it is a reasonable property, that the kernel $\mathbf{K}$ has the same expressivity as the kernel $c\mathbf{K}$ $\forall c > 0$.

In literature, the trace of a kernel is often used as (part of the) bounds on the general-

ization error [122]. However, the simple trace is not a good measure for the estimation of the kernel expressivity. In fact, if two kernels differ only for a multiplicative constant, they have different traces.

It is also possible to defined the extreme cases of a good measure, i.e. the kernel matrices with minimal and maximal expressiveness. Excluding the kernel matrix with all the entries equal to zero, the kernel matrix with lower complexity is the matrix defined by $\mathbf{1}_L\mathbf{1}_L^\top \in \mathbb{R}^{L\times L}$. This matrix has only an eigenvalue different from zero and, in fact, it admits only two types of labeling (all the examples have to be positive or negative). On the other hand, the matrix with the highest complexity has to be the identity matrix $\mathbf{I}_L$. The identity matrix admits all the possible labelings of the examples and, in fact, using this matrix as similarity measure in a large margin kernel machine (e.g. KOMD, Chapter 5), every classification task becomes separable in training data, as explained in the following preliminary section.

8.2 Eigenvalues and Data Separation

Given two generic points in the convex hulls of positive and negative training examples in feature space, specified by a vector $\boldsymbol{\gamma} \in \hat{\Gamma}$ (see Chapter 2), then their squared distance can be computed by

$$D = ||\sum_{i\in\oplus}\gamma_i\boldsymbol{\phi}(\mathbf{x}_i) - \sum_{i\in\ominus}\gamma_i\boldsymbol{\phi}(\mathbf{x}_i)||^2 = ||\sum_i y_i\gamma_i\boldsymbol{\phi}(\mathbf{x}_i)||^2 = \sum_{i=1}^{l}\sum_{j=1}^{l} y_i y_j\gamma_i\gamma_j\hat{\mathbf{K}}_{ij} \quad (8.2)$$

$$= \boldsymbol{\gamma}^\top\hat{\mathbf{Y}}\hat{\mathbf{K}}\hat{\mathbf{Y}}\boldsymbol{\gamma} \quad (8.3)$$

and $\hat{\mathbf{Y}} = \mathbf{diag}(\hat{\mathbf{y}})$. It follows that the minimum distance between the convex hulls of positive and negative examples can be computed by

$$D_{min} = \min_{\boldsymbol{\gamma}\in\hat{\Gamma}} \gamma^\top\hat{\mathbf{Y}}\hat{\mathbf{K}}\hat{\mathbf{Y}}\boldsymbol{\gamma} \quad (8.4)$$

and similarly the maximum squared distance between pairs of positive and negative examples can be obtained by:

$$D_{max} = \max_{\boldsymbol{\gamma}\in\hat{\Gamma}} \gamma^\top\hat{\mathbf{Y}}\hat{\mathbf{K}}\hat{\mathbf{Y}}\boldsymbol{\gamma}. \quad (8.5)$$

R It is interesting to note that the value D_{min} is obtained solving the objective function Ω from the not regularized version of KOMD presented in Chapter 5.

Using well know properties of symmetric positive semidefinite matrices [172], we can give a lower bound of the minimum distance as follows:

$$\min_{\boldsymbol{\gamma}\in\Gamma} \boldsymbol{\gamma}^\top \hat{\mathbf{Y}}\hat{\mathbf{K}}\hat{\mathbf{Y}}\boldsymbol{\gamma} \geq \lambda_l \min_{\boldsymbol{\gamma}\in\hat{\Gamma}} ||\hat{\mathbf{Y}}\boldsymbol{\gamma}||^2 \geq \frac{2}{l}\lambda_l \tag{8.6}$$

and λ_l is the minimum eigenvalue of the matrix $\hat{\mathbf{K}}$. This implies that, when the kernel matrix is not singular, then any random labeling can be shattered by an hyperplane in feature space as the distance between the convex hulls is strictly larger than zero, and this is true for any labeling.

Similarly, we can give an upper bound to the maximum distance of training examples which depends on the maximal eigenvalue this time, that is

$$\max_{\boldsymbol{\gamma}\in\Gamma} \boldsymbol{\gamma}^\top \hat{\mathbf{Y}}\hat{\mathbf{K}}\hat{\mathbf{Y}}\boldsymbol{\gamma} \leq \lambda_1 \max_{\boldsymbol{\gamma}\in\Gamma} ||\hat{\mathbf{Y}}\boldsymbol{\gamma}||^2 \leq \lambda_1. \tag{8.7}$$

At a first glance these results would suggest that a kernel is a "good" kernel when is able to maximize the minimum distance (separation of training data) and minimize the maximum distance (spreading of training data). Looking now at the kernel spectrum, this is typical when the difference between the maximum eigenvalue and the minimum eigenvalue is small. As an extreme example, this happens with the identity matrix. However, these kinds of kernels simply overfit the training data, i.e. they will be very bad when used on new data. So, we see that the spectrum of a kernel can give precious information about the complexity of the kernel. Note that the trace of the matrix alone is not able to capture this information as there can be matrices with very different complexity and same trace.

Now we are ready to introduce, in a formal way, our new measure of kernel complexity.

8.3 A new measure for kernel complexity

The spectral complexity for a positive semidefinite matrix $\mathbf{K}$ is defined as the ratio between the 1-norm and the 2-norm of its eigenvalues, or equivalently, as the ratio between its trace norm $||\mathbf{K}||_T$ and its Frobenius norm $||\mathbf{K}||_F$.

This measure can be computed in several ways:

$$\mathcal{C}(\mathbf{K}) = \frac{\sum_{i=1}^{L} \lambda_i}{\sqrt{\sum_{i=1}^{L} \lambda_i^2}} = \frac{||\mathbf{K}||_T}{||\mathbf{K}||_F} = \frac{tr(\mathbf{K})}{\sqrt{\sum_{ij} \mathbf{K}[i,j]^2}}, \tag{8.8}$$

since, for a positive definite matrix, the Frobenius norm is equal to the 2-norm of its eigenvalues and the trace is equal to the sum of its eigenvalues.

Note that, the (squared) spectral complexity can also be seen as an approximation of

the rank of a matrix. In fact, from a result in [180]:

$$||\mathbf{K}||_F \leq ||\mathbf{K}||_T \leq \sqrt{\text{rank}(\mathbf{K})}||\mathbf{K}||_F, \tag{8.9}$$

we can easily derive the following strict bounds:

$$1 \leq \mathcal{C}(\mathbf{K}) \leq \sqrt{\text{rank}(\mathbf{K})}. \tag{8.10}$$

In particular, the spectral complexity $\mathcal{C}(\mathbf{K})$ has the following nice properties:

- the identity matrix $\mathbf{I}_L$ having rank equal to L has the maximal spectral complexity with $\mathcal{C}(\mathbf{I}_L) = \sqrt{L}$,
- the kernel $\mathbf{K} = \mathbf{1}_L\mathbf{1}_L^\top$ having rank equal to 1 has the minimal spectral complexity with $\mathcal{C}(\mathbf{1}_L\mathbf{1}_L^\top) = 1$,
- it is invariant to multiplication with a positive scalar as

$$\mathcal{C}(\alpha\mathbf{K}) = \mathcal{C}(\mathbf{K}), \ \ \forall\alpha > 0.$$

An equivalent standardized version can also be used with values in the interval $[0,1]$, that is

$$\mathcal{C}_s(\mathbf{K}) = \frac{R(\mathbf{K}) - 1}{\sqrt{L} - 1} \in [0,1]. \tag{8.11}$$

Now, we are ready to introduce an important definition for a family of *weak* kernels $\mathcal{K}$ (or for a class of kernel functions):

Definition 8.3.1 A class of kernel functions $\mathcal{K}$ is Monotonically Spectral-Complex (MSC) iff, the elements of the class can be ordered ($\mathcal{K} = \{K_1, K_2, \dots\}$) in such a way that, for any possible row-normalized dataset $\mathbf{X}$ and for any pair of indexes $i < j$, the following holds:

$$\mathcal{C}(\mathbf{K}_\mathbf{X}^{(i)}) \leq \mathcal{C}(\mathbf{K}_\mathbf{X}^{(j)}), \tag{8.12}$$

where $\mathbf{K}_\mathbf{X}^{(i)}$ is the kernel matrix evaluated on data $\mathbf{X}$ using the kernel function K_i.

R A row-normalized dataset $\mathbf{X}$ is a dataset where each example is normalized with respect to the $2-$norm, i.e

$$\|\mathbf{X}[i,:]\|_2 = 1, \forall i. \tag{8.13}$$

Geometrically, a row-normalized dataset have all the examples on the surface of an hyper-sphere.

8.3.1 A toy example using the RBF kernel

We consider the widely used RBF kernel in the form

$$K^{\gamma}_{RBF}(\mathbf{x},\mathbf{y}) = \exp(-\gamma\|\mathbf{x}-\mathbf{y}\|_2^2) \text{ with } \gamma \geq 0. \tag{8.14}$$

It is well known that with $\gamma = 0$ the kernel matrix becomes equal to $\mathbf{1}_L\mathbf{1}_L^\top$. On the other hand, for $\gamma \to +\infty$ the kernel matrix equals to $\mathbf{I}_L$. In this section, we will use two benchmark datasets (Heart and Sonar) to compare empirically the spectral complexity with other baseline measures:

- the $rank(\mathbf{K})$ of the kernel matrix (i.e. the number of eigenvalues different from zero):
$$rank(\mathbf{K}) = |\{\lambda_i : \lambda_i > 0, i = 1,\dots,L\}|, \tag{8.15}$$
- the number of eigenvalues that are greater than a fixed threshold of 10^{-5}:
$$treshold(\mathbf{K}) = |\{\lambda_i : \lambda_i > 10^{-5}, i = 1,\dots,L\}|, \tag{8.16}$$
- the *entropy* evaluated over the list of the eigenvalues (i.e. how much the eigenvalues are uniformly distributed):
$$entropy(\mathbf{K}) = \sum_{i=1}^{L} \lambda_i * \log(\lambda_i). \tag{8.17}$$

The results has been standardized in the range $[0,1]$ and are depicted in Figure 8.1 for Heart and Sonar, using $\gamma \in \{2^i : i = -15,-14,\dots,14,15\}$.

All the evaluated measures increase their values monotonically with the parameter γ. *rank* and *threshold* are similar to a *step function* considering only the cardinality of a set of eigenvalues, without take into account the distribution of them. On the other hand, the *entropy* is more similar to our spectral complexity $\mathcal{C}_s$ but it requires the entire list of eigenvalues.

At least empirically, we have shown that a family of *weak* kernels $\mathcal{K}$ defined as

$$\mathcal{K} = \{\mathbf{K}^{\gamma}_{RBF} : \gamma \in \{2^i : i = -15,-14,\dots,14,15\}\} \text{ is a MSC family.} \tag{8.18}$$

Now we are ready to introduce an important property of the spectral complexity. In fact, the spectral complexity is strictly connected to the *empirical Rademacher complexity* as proven in the following section.

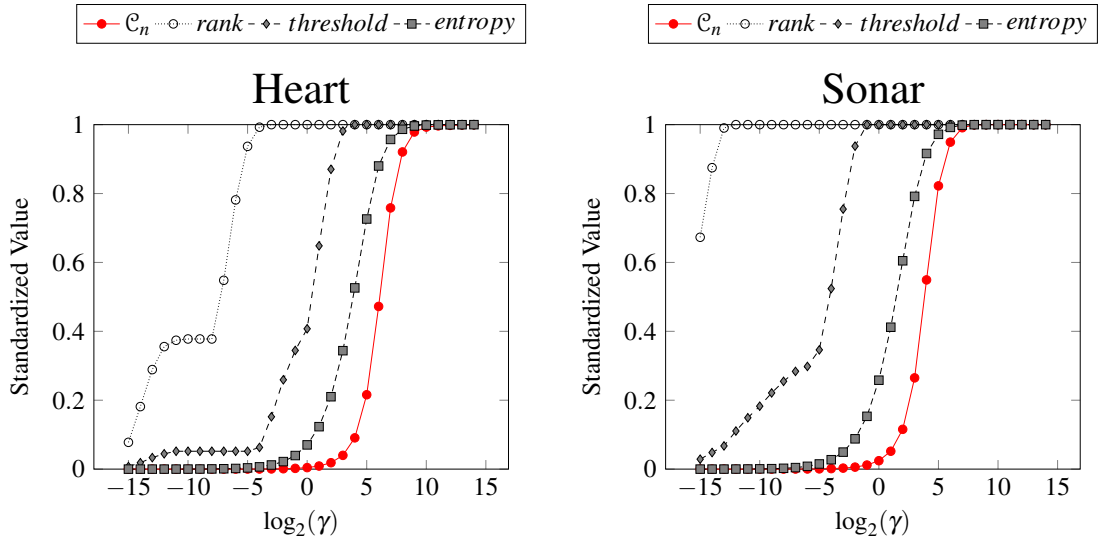


Figure 8.1: The spectral complexity $\mathcal{C}_s$ compared to the rank of the kernel matrix (*rank*), the number of the eigenvalues greater than 10^{-5} (*threshold*) and the *entropy* of the eigenvalues using an RBF kernel function with $\gamma \in \{2^i : i = -15, -14, \ldots, 14, 15\}$ for Heart and Sonar.

8.4 Connection to the Rademacher complexity

Firstly, we define the set of the vectors with fixed 1-norm

$$\Psi = \{\boldsymbol{\alpha} \in \mathbb{R}^L \text{ s.t. } \|\boldsymbol{\alpha}\|_1 = 1\}. \tag{8.19}$$

Then, the class of functions that we consider are linear functions with bounded norm

$$\{\mathbf{x}_j \to \sum_{i=1}^{l} \alpha_i \mathbf{K}_{i,j} : \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha} \leq B^2, \forall \boldsymbol{\alpha} \in \Psi\} \subseteq \tag{8.20}$$

$$\subseteq \{\mathbf{x}_j \to \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}_j) : \|\mathbf{w}\|_2 \leq B\} = \mathcal{F}_B, \tag{8.21}$$

where $\boldsymbol{\phi}$ is the feature mapping for the kernel $\mathbf{K}$. The following bound theorem for the complexity of $\mathcal{F}_B$ holds:

Theorem 8.4.1 ([172], Theorem 4.12). Given a kernel $\mathbf{K}$, evaluated over a set of points $\mathcal{X}$, the empirical Rademacher complexity of the class $\mathcal{F}_B$ satisfies

$$\hat{\mathcal{R}}(\mathcal{F}_B) \leq \mathcal{O}(B\sqrt{tr(\mathbf{K})}). \tag{8.22}$$

Equation 8.27 shows that the *empirical Rademacher complexity* is proportional to the trace of the kernel. Now, we are ready to introduce important observations about the value of $\boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha}$, for a general kernel $\mathbf{K}$. In fact, the following proposition holds:

Proposition 8.4.2 Let $\mathbf{K}$ be a kernel matrix in $\mathbb{R}^{L \times L}$ with eigenvalues $\lambda_1 \geq \cdots \geq \lambda_L \geq 0$,

then

$$\forall \boldsymbol{\alpha} \in \Psi, \ \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha} \leq \lambda_1 \leq \|\mathbf{K}\|_F. \tag{8.23}$$

and λ_1 is the tightest bound possible.

Proof. We can exploit the spectral decomposition of a kernel matrix and rewrite

$$\boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha} = \sum_{i=1}^{L} \lambda_i (\boldsymbol{\alpha}^T \mathbf{u}_i)^2, \tag{8.24}$$

where $\mathbf{u}_i$ is the eigenvector with eigenvalue λ_i. Then, it is easy to see that:

$$(\boldsymbol{\alpha}^T \mathbf{u}_i)^2 = \cos(\theta_{\boldsymbol{\alpha},\mathbf{u}_i})^2 \|\boldsymbol{\alpha}\|_2^2, \tag{8.25}$$

where $\theta_{\boldsymbol{\alpha},\mathbf{u}_i}$ is the angle between the vector $\boldsymbol{\alpha}$ and the eigenvector $\mathbf{u}_i$. Using the properties of the norms ($\|\boldsymbol{\alpha}\|_2^2 \leq \|\boldsymbol{\alpha}\|_1^2 = 1$) and the fact that the eigenvectors $\mathbf{u}_i$ are an orthonormal base we can obtain the final result:

$$\boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha} \leq \sum_{i=i}^{L} \lambda_i \cos(\theta_{\boldsymbol{\alpha},\mathbf{u}_i})^2 \|\boldsymbol{\alpha}\|_1^2 \leq \lambda_1 \leq \sqrt{\sum_{i=1}^{L} \lambda_i^2} = \|\mathbf{K}\|_F.$$

The bound is reached when the kernel matrix $\mathbf{K}$ has only one eigenvalue different from zero, its eigenvector is parallel to the vector $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}$ is such that $\|\boldsymbol{\alpha}\|_2 = \|\boldsymbol{\alpha}\|_1$. ■

Finally, from the previous results the following corollary holds:

Corollary 8.4.3 Let $\mathbf{K}$ be a kernel matrix in $\mathbb{R}^{L \times L}$ with eigenvalues $\lambda_1 \geq \cdots \geq \lambda_L \geq 0$, then

$$\forall \boldsymbol{\alpha} \in \Psi, \ \frac{\boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha}}{\|\mathbf{K}\|_F} \leq 1. \tag{8.26}$$

From Proposition 8.4.2 and using Theorem 8.4.4, given a kernel with the Frobenius norm equal to 1, we have that the upper bound of the *empirical Rademacher complexity* is equal to $\hat{\mathcal{R}}(\mathcal{F}_B) \leq \mathcal{O}(\sqrt{tr(\mathbf{K})})$, fixing $B^2 = B = 1$. Then, the trace of kernels with this property is a good estimator for their *empirical Rademacher complexity*.

Corollary 8.4.3 highlights a method to normalize a kernel obtaining $\|\mathbf{K}\|_F = 1$. Finally, we are ready to prove the following theorem:

Theorem 8.4.4 Given a kernel $\mathbf{K}$, evaluated over a set of points $\mathcal{X}$, the empirical Rademacher complexity $\hat{\mathcal{R}}$ of the class of linear functions $\mathcal{F} = \{\mathbf{x}_j \to \sum_{i=1}^{l} \alpha_i \mathbf{K}[i,j], \boldsymbol{\alpha} \in \Psi\}$ satisfies

$$\hat{\mathcal{R}}(\mathcal{F}) \leq \mathcal{O}(\sqrt{\mathcal{C}(\mathbf{K})}). \tag{8.27}$$

Proof. From the definition of the spectral complexity, we have that $\sqrt{\mathcal{C}(\mathbf{K})} = \sqrt{\frac{tr(\mathbf{K})}{\|\mathbf{K}\|_F}} = \sqrt{tr(\frac{\mathbf{K}}{\|\mathbf{K}\|_F})}$. Then, the spectral complexity is the trace of the kernels with Frobenious norm equal to 1, i.e. in the form $\frac{\mathbf{K}}{\|\mathbf{K}\|_F}$. Using the Corollary 8.4.3 we can claim that $B = B^2 = 1$ is a good value as upper bound of $\boldsymbol{\alpha}^T \frac{\mathbf{K}}{\|\mathbf{K}\|_F}\boldsymbol{\alpha} \leq 1$. Then, from Theorem 8.4.4, for the family $\mathcal{F}_1$ the following inequality holds:

$$\hat{\mathcal{R}}(\mathcal{F}_1) \leq \mathcal{O}(\sqrt{tr(\frac{\mathbf{K}}{\|\mathbf{K}\|_F})}) = \mathcal{O}(\sqrt{\mathcal{C}(\mathbf{K})}).$$

Finally, we can conclude observing that $\mathcal{F}_1 = \mathcal{F}$ for each kernel defined as $\frac{\mathbf{K}}{\|\mathbf{K}\|_F}$. ■

In this chapter we have presented a new measure for kernel complexity, called *spectral complexity*. This new measure is easy to evaluate and is strictly connected to the *empirical Rademacher complexity* as proved in Theorem 8.4.4.

9. *Weak* kernels optimization

In this chapter, we focus on principled ways to design base kernels and make their supervised combination effective. In particular, we propose to design base kernels of increasing expressiveness (complexity) in a way that the supervised MKL algorithm can detect those components which are really useful to the task at hand. For this, the spectral complexity will be exploited as a principled measure of the expressiveness of a kernel.

Intuitively, low complexity kernels correspond to kernels defined on popular features, while, high complexity kernels correspond to kernels defined on more elaborated features that are observed on very few examples. We show how spectral complexity is also very nice at capturing this kind of dependencies between features used by different kernel functions. The hypothesis, similarly to the assumption of the HKL (see Section 3.3.3), is that the features (or the base kernels) can be embedded in a hierarchically connected structure (e.g. a DAG).

Summarizing, the chapter contribution is three-fold:

- Starting from a characterization of the wide class of dot-product kernels $K(\mathbf{x},\mathbf{z}) = f(\mathbf{x}\cdot\mathbf{y})$ as dot-product polynomials (DPP), firstly, we show how common kernels, like the polynomial kernel and RBF, are instances of the class of DPPs and they only differ for the values of the coefficients given to the different terms of the DPP. We propose a MKL approach to learn the coefficients of a general DPP and we empirically show that this approach outperforms the baselines, sometimes significantly, against ten benchmark datasets in terms of AUC;
- We present empirical evidence that building base kernels exploiting the structure

of the features and their dependencies, makes the combined kernel improve upon alternatives which do not exploit this structure;

- We apply the same methodology to kernel for structures, specifically to the ODD graph-kernel. The intrinsic structure of the kernel is used to define a MSC family of weak kernels, as defined in Chapter 8. This gives a principled way to divide the structural features in bunches of increasing spectral complexity.

In the following, the assumption that the datasets are row-normalized is considered. An example $\mathbf{x}$ in a row-normalized dataset is such that $\|\mathbf{x}\|_2 = 1$ and so, it lives on the surface of a hyper-sphere of radius 1.

9.1 Dot-product Polynomial Kernels

A *generalized* polynomial kernel can be built on the top of any other valid base kernel as in $K(\mathbf{x},\mathbf{z}) = p(K_0(\mathbf{x},\mathbf{z}))$, where the base kernel K_0 is any valid kernel and $p : \mathbb{R} \longrightarrow \mathbb{R}$ is a polynomial function with non-negative coefficients, that is $p(x) = \sum_{s=0}^{d} a_s x^s$. In this chapter, we focus on *Dot-Product Polynomials* (DPP) which is the class of generalized polynomial kernels where the simple dot product is used as base kernel, that is $K(\mathbf{x},\mathbf{z}) = p(\mathbf{x}\cdot\mathbf{z})$.

Using a result from harmonic theory [163, 112], we are able to show that a large family of kernels can be formulated as DPP kernels.

Theorem 9.1.1 A function $f : \mathbb{R} \to \mathbb{R}$ defines a positive definite kernel $K : \mathbf{B}(\mathbf{0},1) \times \mathbf{B}(\mathbf{0},1) \to \mathbb{R}$ as $K : (\mathbf{x},\mathbf{z}) \to f(\mathbf{x}\cdot\mathbf{z})$ iff f is an analytic function admitting a Maclaurin expansion with only non-negative coefficients i.e. $f(x) = \sum_{s=0}^{\infty} a_s x^s, a_s \geq 0$.

Basically, this theorem guarantees that any dot product kernel of the form $K(\mathbf{x},\mathbf{z}) = f(\mathbf{x}\cdot\mathbf{z})$, $\mathbf{x}$ and $\mathbf{z}$ defined in the unitary ball, can be characterized by its Maclaurin expansion with non-negative coefficients, that is a DPP. On the other hand, any choice of non negative coefficients of a DPP induces a new valid kernel.

Classical vector kernels can be formulated as a DPP where the coefficients are fixed (in the worst case using an infinite sum):

- The *polynomial kernel*, $K_{D,c}(\mathbf{x},\mathbf{z}) = (\mathbf{x}\cdot\mathbf{z}+c)^D \ \forall c \geq 0 \in \mathbb{R}, D \in \mathbb{N}$ is a DPP kernel. In fact, resorting to the binomial theorem expansion [172], we obtain:

$$K_{D,c}(\mathbf{x},\mathbf{z}) = \sum_{d=0}^{D} \binom{D}{d} c^{D-d} (\mathbf{x}\cdot\mathbf{z})^d = \sum_{d=0}^{D} a_d (\mathbf{x}\cdot\mathbf{z})^d \tag{9.1}$$

where $a_d = \binom{D}{d} c^{D-d}, d \in \{0,\ldots,D\}$ are the coefficients of a DPP. Selecting the particular case with $c = 0$, we obtain the *homogeneous polynomial kernel* (HPK):

$$K_D(\mathbf{x},\mathbf{z}) = (\mathbf{x}\cdot\mathbf{z})^D \tag{9.2}$$

which is also a DPP having only one non null coefficient $a_D = 1$.

- The *RBF kernel*, $K^{\gamma}_{RBF}(\mathbf{x},\mathbf{z}) = e^{-\gamma||\mathbf{x}-\mathbf{z}||^2}\ \forall \gamma \geq 0 \in \mathbb{R}$ is another instance of DPP. In fact, when data are normalized, using the Taylor expansion, we can show the following:

$$\begin{aligned} K^{\gamma}_{RBF}(\mathbf{x},\mathbf{z}) &= e^{-\gamma||\mathbf{x}||^2} e^{-\gamma||\mathbf{z}||^2} e^{2\gamma \mathbf{x}\cdot\mathbf{z}} && (9.3)\\ &= e^{-2\gamma} e^{2\gamma \mathbf{x}\cdot\mathbf{z}} && (9.4)\\ &= e^{-2\gamma} \sum_{s=0}^{\infty} \frac{(2\gamma)^{2s}}{s!} (\mathbf{x}\cdot\mathbf{z})^s. && (9.5)\\ & && (9.6) \end{aligned}$$

Moreover, the list of possible kernels that can be decomposed in the DPP form also contains: linear kernel, ANOVA kernel and others.

9.2 MKL for learning the DPP coefficients

Interestingly, in all the cases above, the obtained kernel matrix can be decomposed as a (possibly infinite) non negative combination of HPK matrices, that is:

$$\mathbf{K} = \sum_{s=0}^{\infty} a_s \mathbf{K}^{(s)}$$

where $\mathbf{K}^{(s)}[i,j] = (\mathbf{x}_i \cdot \mathbf{x}_j)^s$, and this fact makes this decomposition suitable to a solution based on multiple kernel learning. In practice, given a task, we propose to learn the coefficients of a generic DPP in a supervised manner, in a way to specialize the resulting kernel to the task at hand.

As presented in Section 3.3.3, many different algorithms for MKL have been proposed in the last years, including [157, 189, 117, 202, 182, 17, 4]. In principle, any of these algorithms can be used for our purpose. In the next sections, we will exploit the EasyMKL algorithm, presented in Chapter 7 as our MKL algorithm. EasyMKL is very efficient and is able to combine a large number of kernels in a fast way, for this reason is suitable for our purposes.

R In general, EasyMKL does not take into account the complexity of the *weak* kernels but it only focuses on the margin obtained by their combination.

9.3 Exploiting the structure of the features

In this section, we define MSC families of *weak* kernels starting from the structure of the HPKs.

Structure of Polynomial Kernels

The parameter d of a HPK of the form $K(\mathbf{x},\mathbf{z}) = (\mathbf{x}\cdot\mathbf{z})^d$ induces a MSC family of kernel functions. More formally:

Proposition 9.3.1 Let $\mathcal{K}_D = \{K_0,\dots,K_D\}$ the family of kernel functions, where $K_d(\mathbf{x},\mathbf{z}) = (\mathbf{x}\cdot\mathbf{z})^d$ is the d-degree homogeneous polynomial kernel. Then, for any row-normalized dataset $\mathbf{X}$, the family $\mathcal{K}_D$ is MSC.

Proof. Let i, j be two indexes such that $0 \leq i < j$, we need to prove that $\mathcal{C}(\mathbf{K}_\mathbf{X}^{(i)}) \leq \mathcal{C}(\mathbf{K}_\mathbf{X}^{(j)})$ for any row-normalized dataset $\mathbf{X}$. Since data are normalized, for any d and for any $\mathbf{x}$, $K_d(\mathbf{x},\mathbf{x}) = (\mathbf{x}\cdot\mathbf{x})^d = 1^d = 1$. Then, the d-degree HPK is also normalized and the trace is a constant that does not depend on d. So, we only need to prove the following:

$$||\mathbf{K}_\mathbf{X}^{(i)}||_F \geq ||\mathbf{K}_\mathbf{X}^{(j)}||_F$$

which is easy to show, since:

$$||\mathbf{K}_\mathbf{X}^{(i)}||_F = \sum_{i,j}^{L,L}(\mathbf{x}_i\cdot\mathbf{x}_j)^{2i} \geq \sum_{i,j}^{L,L}(\mathbf{x}_i\cdot\mathbf{x}_j)^{2j} = ||\mathbf{K}_\mathbf{X}^{(j)}||_F$$

where we used the fact that $\mathbf{x}_i\cdot\mathbf{x}_j \leq 1$ since the vectors are normalized. Moreover, $z^i \geq z^j$ when $i < j$ and $|z| \leq 1$. ■

Note that, since the feature space of a d-degree HPK is constructed with all possible d-degree monomials. Then, there is a clear dependence from features of higher order HPKs towards features of lower order HPKs. For example, the value of the feature $x_1x_4x_5x_9$ in the 4-degree HPK depends on the values of the features x_1, x_4, x_5 and x_9 in the 1-degree HPK. An example is depicted in Figure 9.1. In general, the higher the order of the HPK, the sparser is the kernel matrix produced. This explains why the Frobenius norm of the kernel matrix decreases with the HPK order.

9.3.1 Margin and Complexity: an example

The structure of the features imposes a particular behavior of the weights assigned by EasyMKL. In fact, EasyMKL assigns the weight of a specific base kernel proportionally with respect to its contribution to the margin. Given one point in the convex hull of positive examples $\mathbf{x}^+$ and one point in the convex hull of the negative examples$\mathbf{x}^-$, i.e. defining a

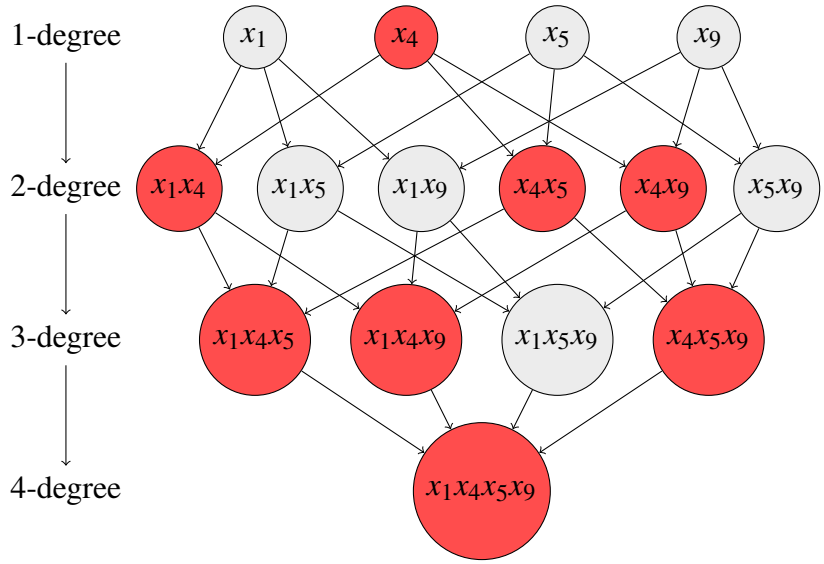


Figure 9.1: Example of dependencies of the features. The arrows represent the dependencies among the features of different degrees. The nodes in red, starting from the top represent the diffusion of the zeros (i.e. the sparsity): if the value of x_4 is zero then the value of all the dependent features is also zero. Conversely, if the value of the root $x_1x_4x_5x_9$ is different from zero then all the features in the graph have values different from zero.

distribution $\boldsymbol{\gamma} \in \Gamma$ such that

$$\mathbf{x}^+ = \sum_{i \in \oplus} \gamma_i \boldsymbol{\phi}_K(\mathbf{x}_i) \text{ and } \mathbf{x}^- = \sum_{j \in \ominus} \gamma_j \boldsymbol{\phi}_K(\mathbf{x}_i), \tag{9.7}$$

it is possible to write the squared distance between them, in the feature space of a base kernel function K as

$$\begin{aligned}
\mathcal{D}_K^2(\mathbf{x}^+, \mathbf{x}^-) =& \|\mathbf{x}^+\|_2^2 + \|\mathbf{x}^-\|_2^2 - 2\mathbf{x}^+ \cdot \mathbf{x}^- \\
=& \sum_{i,j \in \oplus} \gamma_i \gamma_j \boldsymbol{\phi}_K(\mathbf{x}_i) \cdot \boldsymbol{\phi}_K(\mathbf{x}_j) + \sum_{i,j \in \ominus} \gamma_i \gamma_j \boldsymbol{\phi}_K(\mathbf{x}_i) \cdot \boldsymbol{\phi}_K(\mathbf{x}_j) \\
& -2 \sum_{i \in \oplus, j \in \ominus} \gamma_i \gamma_j \boldsymbol{\phi}_K(\mathbf{x}_i) \cdot \boldsymbol{\phi}_K(\mathbf{x}_j) \\
=& \sum_{i,j \in \oplus} \gamma_i \gamma_j K(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i,j \in \ominus} \gamma_i \gamma_j K(\mathbf{x}_i, \mathbf{x}_j) \\
& -2 \sum_{i \in \oplus, j \in \ominus} \gamma_i \gamma_j K(\mathbf{x}_i, \mathbf{x}_j).
\end{aligned}$$

When we deal with a normalized HPKs family of *base* kernel $\mathcal{K}_D = \{K_0, \ldots, K_D\}$, the sum of the first two terms of this equation is equal to 2 when $d = 0$ and is monotonically decreasing with respect to d. In fact, we have that $K_{d+1}(\mathbf{x}_i, \mathbf{x}_j) \leq K_d(\mathbf{x}_i, \mathbf{x}_j), \ \forall d \geq 0$.

Moreover, for a sufficiently large value of $d < \infty$, the two terms converge to $\sum_{i\in\oplus}\gamma_i^2 + \sum_{i\in\ominus}\gamma_i^2 > 0$.

On the other hand, the third term of the equation, that is

$$-2\sum_{i\in\oplus,j\in\ominus}\gamma_i\gamma_j K_d(\mathbf{x}_i,\mathbf{x}_j), \tag{9.8}$$

is a monotonically increasing function with respect to the value of the degree d. In fact, it is equal to -2 when $d=0$ and it increases monotonically its value until 0 (also in this case because of the monotonicity of the value $K_d(\mathbf{x}_i,\mathbf{x}_j)$).

We define $F(d) := \mathcal{D}^2_{K_d}(\mathbf{x}^+,\mathbf{x}^-)$ and from the previous observations, we can claim that for a sufficiently large d, the value of the squared distance converge to $\lim_{d\to+\infty}F(d)(\mathbf{x}^+,\mathbf{x}^-) = \sum_{i\in\oplus}\gamma_i^2 + \sum_{i\in\ominus}\gamma_i^2 > 0$. Now, we are interested in showing a case in which the maximal distance is greater than this value (the maximal margin is not reached using the maximal complexity). We impose the following hypothesis:

- Kernels We use as kernels the normalized HPKs;

- Reasonable labels. We impose the property $\mathbf{x}_i\cdot\mathbf{x}_j \in [1-\varepsilon,1]$ and $\mathbf{x}_s\cdot\mathbf{x}_t\in[0,\varepsilon]$ when $y_iy_j>0$ and $y_sy_t<0$ for a fixed $0<\varepsilon<\frac{1}{2}$ and we suppose a balanced labeling (specifically, we consider n positive and n negative examples);

- Uniform distribution in Γ. We select the uniform distribution $\boldsymbol{\gamma}^{unif}\in\Gamma$ in order to fix the two points $\mathbf{x}^+$ and $\mathbf{x}^-$ as the two centroids of the convex hulls of positive and negative examples.

We have to prove the existence of a d^* such that $\forall d > d^*,\ F(d+1)-F(d)<0$. In our case, we have that:

$$\begin{aligned}F(d+1)-F(d) =& \frac{1}{n^2}\sum_{i,j\in\oplus}(\mathbf{x}_i\cdot\mathbf{x}_j-1)K_d(\mathbf{x}_i,\mathbf{x}_j) + \frac{1}{n^2}\sum_{i,j\in\ominus}(\mathbf{x}_i\cdot\mathbf{x}_j-1)K_d(\mathbf{x}_i,\mathbf{x}_j)\\ &-\frac{2}{n^2}\sum_{i\in\oplus,j\in\ominus}(\mathbf{x}_i\cdot\mathbf{x}_j-1)K_d(\mathbf{x}_i,\mathbf{x}_j).\end{aligned}$$

Now, we are ready to point-out the following interesting facts: (a) when d is sufficiently large the values of $K_d(\mathbf{x}_i,\mathbf{x}_j)$ becomes similar to 0 if $i\neq j$; (b) in the case of $i=j$ we have that $(\mathbf{x}_i\cdot\mathbf{x}_j-1)$ is equals to 0; (c) $(\mathbf{x}_i\cdot\mathbf{x}_j-1) > (\mathbf{x}_s\cdot\mathbf{x}_t-1)$ and $K_d(\mathbf{x}_i,\mathbf{x}_j) > K_d(\mathbf{x}_s,\mathbf{x}_t)$ when $y_iy_j>0$ and $y_sy_t<0$.

Changing ε sufficiently small, we are able to impose that

$$\frac{(\mathbf{x}_i\cdot\mathbf{x}_j-1)K_d(\mathbf{x}_i,\mathbf{x}_j)}{(\mathbf{x}_s\cdot\mathbf{x}_t-1)K_d(\mathbf{x}_s,\mathbf{x}_t)} \geq \frac{(\mathbf{x}_i\cdot\mathbf{x}_j-1)(1-\varepsilon)^d}{(\mathbf{x}_s\cdot\mathbf{x}_t-1)\varepsilon^d} > 1 \text{ if } d \text{ is sufficiently large.}$$

In fact, the values $(\mathbf{x}_i\cdot\mathbf{x}_j-1)$ and $(\mathbf{x}_s\cdot\mathbf{x}_t-1)$ are fixed and $K_d(\mathbf{x}_i,\mathbf{x}_j) = (\mathbf{x}_i\cdot\mathbf{x}_j)^d \geq$

$(1-\varepsilon)^d > \varepsilon^d \geq (\mathbf{x}_s \cdot \mathbf{x}_t)^d = K_d(\mathbf{x}_s, \mathbf{x}_t)$.

Then, the third term of $F(d+1) - F(d)$ (that is the positive term) becomes 0 more rapidly with respect to the first two terms (that are the two negative terms). Then, for a certain d^* the value of $F(d+1) - F(d) \ \forall d > d^*$ becomes negative and $F(d) \ \forall d > d^*$ decreases its value. This prove that, for at least one $d \leq d^*$, the value of $F(d)$ is larger with respect to $\lim_{d \to +\infty} F(d)(\mathbf{x}^+, \mathbf{x}^-)$ and consequently the largest margin is not obtained exploiting the highest complexities.

In conclusion, when the problem has a *reasonable* labeling the plot of the assigned weights to the normalized HPKs will follow a curve with a maximal value that is greater then $\sum_{i \in \oplus} \gamma_i^2 + \sum_{i \in \ominus} \gamma_i^2$ (i.e. greater than the margin obtained using the maximal complexity). In Section 9.4.2, we will show some examples of assigned weights that confirm our observations (see Figure 9.7).

9.4 Experimental Work

In the following sections, first we will show empirically that $\mathcal{K}_D$ is effectively a good choice in practice as family of *weak* kernels and then we will show the importance of the MSC structure of $\mathcal{K}_D$. Experiments have been performed for a total of ten different benchmark datasets. Further details about the datasets are summarized in Table 2.2.

The AUC results are obtained using a stratified nested 10-fold cross validation. Specifically we used the following procedure:

- Each dataset has been divided in 10 folds $\mathbf{f}_1, \ldots, \mathbf{f}_{10}$ respecting the distribution of the labels, where $\mathbf{f}_i$ contains the list of indices of the examples in the fold i^{th};
- One fold $\mathbf{f}_j$ is selected as test set;
- The remaining nine out of ten folds $\mathbf{v}_j = \bigcup_{i=1, i \neq j}^{10} \mathbf{f}_i$ are selected as validation set for the choice of hyper-parameters, applying a new 10-fold cross validation over $\mathbf{v}$ respecting the distribution of the labels;
- The set $\mathbf{v}_j$ is selected as training set to generate a model (using the validated hyper-parameters);
- The test fold $\mathbf{f}_j$ is used as test set to evaluate the performance of the model;
- The reported results are the averages (with standard deviations) obtained repeating this procedure over all the 10 possible test sets $\mathbf{f}_j$ (i.e. for each j in $\{1, \ldots, 10\}$).

.

9.4.1 MKL for learning DPP

In this section we describe the experiments we performed to test the accuracy in terms of AUC of the kernel generated by learning the coefficients of a dot-product polynomial using

$\mathcal{K}_D = \{K_0, \ldots, K_D\}$ as base HPKs, and varying the value of D. This method is indicated with $\mathbf{K}_{MKL}$. For each D, we compare our algorithm against other weighting rules:

- $\mathbf{K}_D$, where the weight η_D is set to 1 (and all the other weights are set to 0). This corresponds to the kernel defined in Equation 9.2;
- $\mathbf{K}_{sum}$, where the weights are set uniformly over the base kernels, that is $\eta_d = \frac{1}{D+1}$ for $d \in \{0, 1, \ldots, D\}$ (as pointed before, this is generally a strong baseline);
- $\mathbf{K}_{D,c}$, where the weights for the base kernels are assigned using the polynomial kernel rule (see Equation 9.1):

$$\eta_d \propto \binom{D}{d} c^{D-d}, \;\; d \in \{0, \ldots, D\}. \tag{9.9}$$

  The value for c has been selected as the one from the set $\{0.5, 1, 2, 3\}$ which obtained the best AUC on the test set.
- $\mathbf{K}^{\gamma}_{RBF}$, where the weights for the weak kernels are assigned according to the *truncated* RBF rule (see Equation 9.6):

$$\eta_d \propto \frac{(2\gamma)^{2s}}{s!}, \;\; d \in \{0, \ldots, D\}. \tag{9.10}$$

  The value for γ has been selected as the one from the set $\{2^i : i \in \{-5, -4, \ldots, 0, 1\}\}$ which obtained the best AUC on the test set.

Note that, the results depicted in the following for $\mathbf{K}_{D,c}$ and $\mathbf{K}^{\gamma}_{RBF}$ are optimistic estimates of the real performance because of the *a posteriori* selection of the best parameters c and γ, respectively.

As described above, a stratified nested 10-fold cross validation has been performed selecting the EasyMKL hyper-parameter Λ from the set of values:

$$\Lambda \in \{0.0, 0.1, \ldots, 0.9, 1.0\}.$$

The AUC results of these experiments are depicted in Figures 9.2, 9.3, 9.4 for all datasets. As the reader can see from the figure, our method consistently and significantly outperforms both the single base kernel solution $\mathbf{K}_D$ and the average solution $\mathbf{K}_{sum}$, especially for high polynomial degrees where $\mathbf{K}_D$ and $\mathbf{K}_{sum}$ tend to overfit. Moreover, our solution is always comparable and often better than the optimistic AUC performance of $\mathbf{K}_{D,c}$ and $\mathbf{K}^{\gamma}_{RBF}$ weighting rules.

9.4.2 Is the HPK structure important?

In this section, we empirically show that the structure present in HPKs is indeed useful in order to obtain good results using our MKL approach. With this aim, we built two alterna-

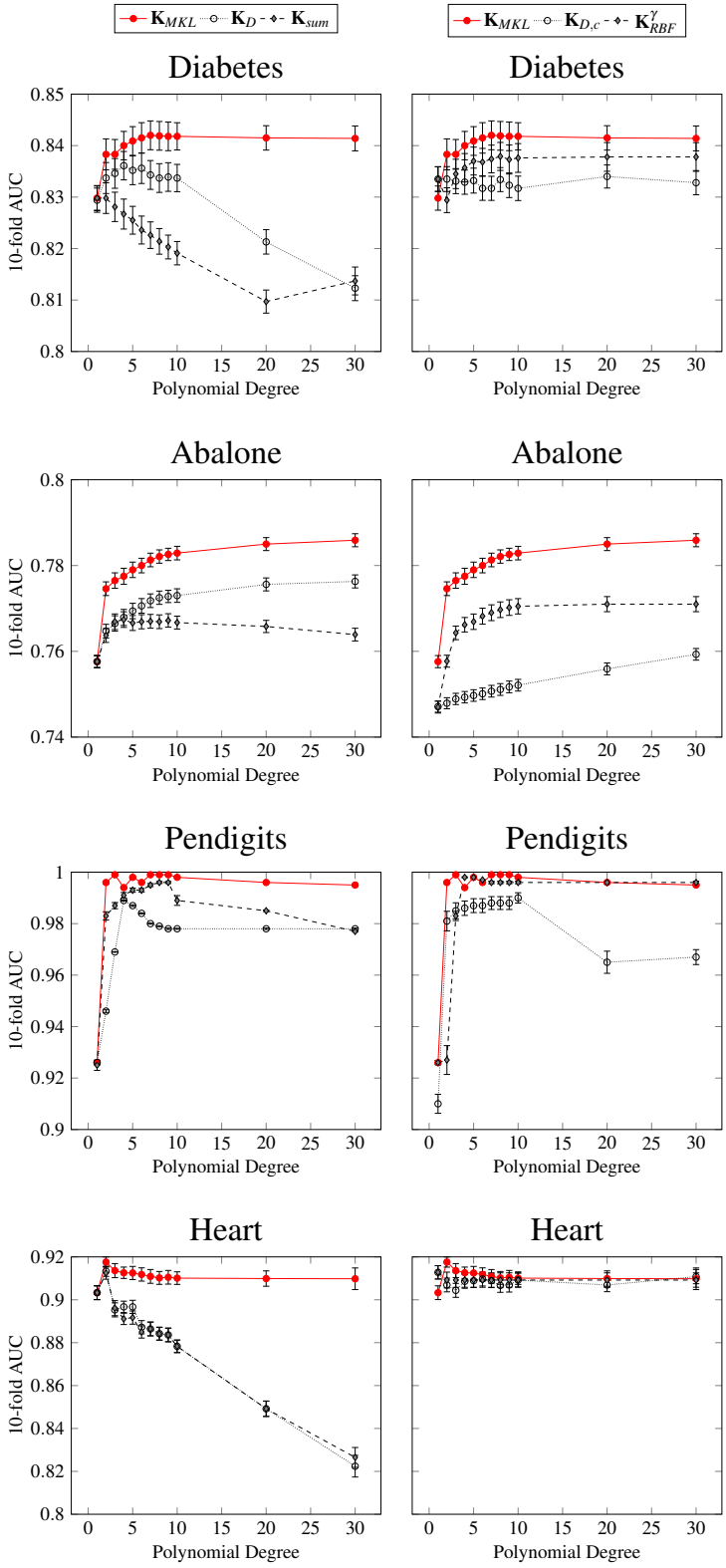


Figure 9.2: AUC with standard deviation among all the ten benchmark datasets for different values of D. A stratified nested 10-fold cross validation has been performed with the Λ hyper-parameter of EasyMKL selected from $\{0.0, 0.1, \ldots, 0.9, 1.0\}$. Our proposed solution $\mathbf{K}_{MKL}$ (solid red line) has been compared against the baselines $\mathbf{K}_D$ and $\mathbf{K}_{sum}$ and the optimistic AUC results of $\mathbf{K}_{D,c}$ and $\mathbf{K}^{\gamma}_{RBF}$ where *a posteriori* optimal $c \in \{0.5, 1, 2, 3\}$ and optimal $\gamma \in \{2^i : i = -5, -4, -3, -2, -1, 0, 1\}$ have been selected, respectively.

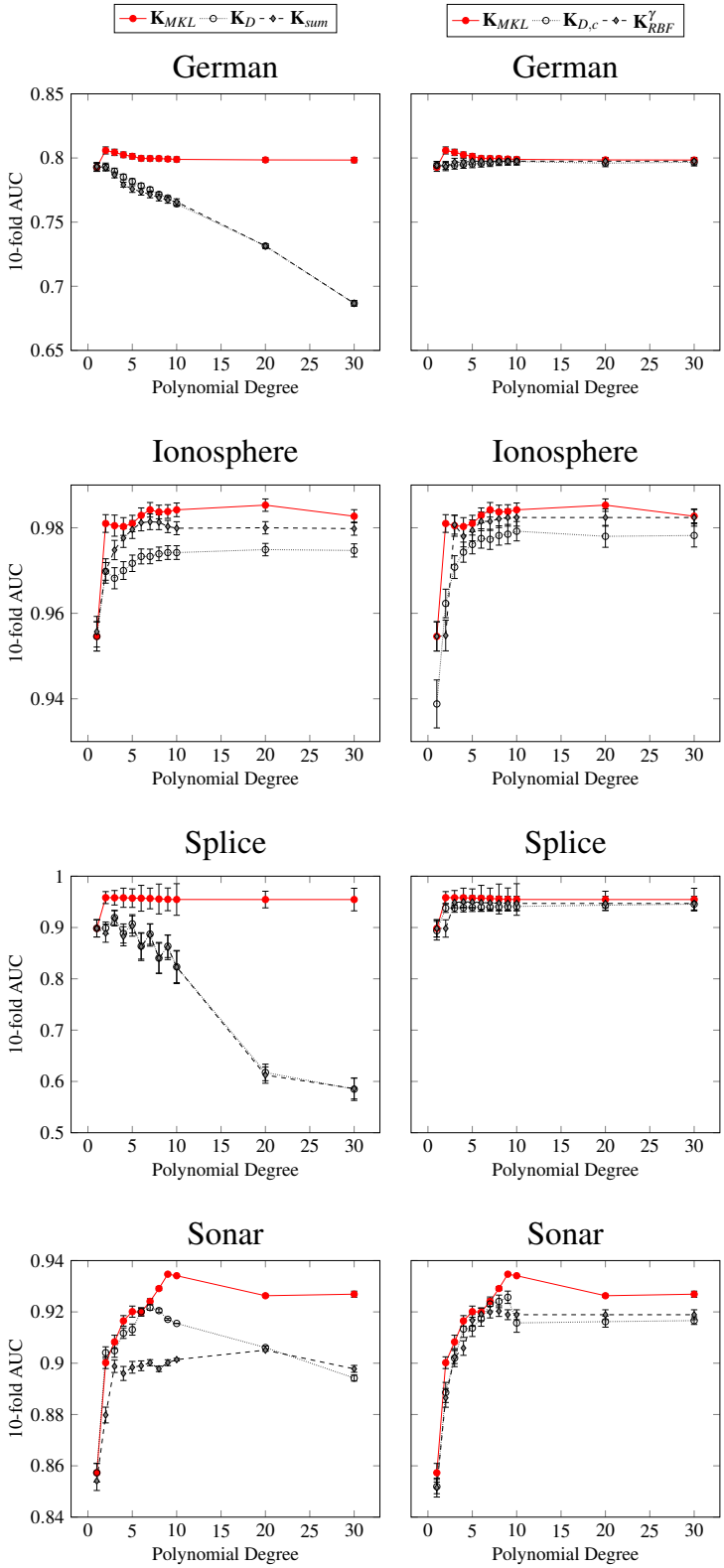


Figure 9.3: AUC with standard deviation among all the ten benchmark datasets for different values of D. A stratified nested 10-fold cross validation has been performed with the Λ hyper-parameter of EasyMKL selected from $\{0.0, 0.1, \ldots, 0.9, 1.0\}$. Our proposed solution $\mathbf{K}_{MKL}$ (solid red line) has been compared against the baselines $\mathbf{K}_D$ and $\mathbf{K}_{sum}$ and the optimistic AUC results of $\mathbf{K}_{D,c}$ and $\mathbf{K}^{\gamma}_{RBF}$ where *a posteriori* optimal $c \in \{0.5, 1, 2, 3\}$ and optimal $\gamma \in \{2^i : i = -5, -4, -3, -2, -1, 0, 1\}$ have been selected, respectively.

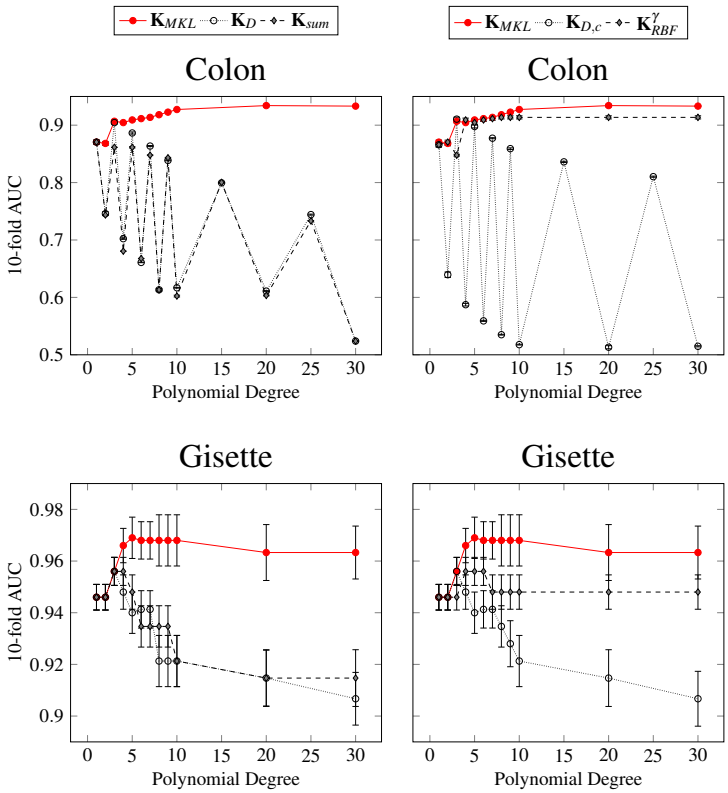


Figure 9.4: AUC with standard deviation among all the ten benchmark datasets for different values of D. A stratified nested 10-fold cross validation has been performed with the Λ hyper-parameter of EasyMKL selected from $\{0.0, 0.1, \ldots, 0.9, 1.0\}$. Our proposed solution $\mathbf{K}_{MKL}$ (solid red line) has been compared against the baselines $\mathbf{K}_D$ and $\mathbf{K}_{sum}$ and the optimistic AUC results of $\mathbf{K}_{D,c}$ and $\mathbf{K}^{\gamma}_{RBF}$ where *a posteriori* optimal $c \in \{0.5, 1, 2, 3\}$ and optimal $\gamma \in \{2^i : i = -5, -4, -3, -2, -1, 0, 1\}$ have been selected, respectively.

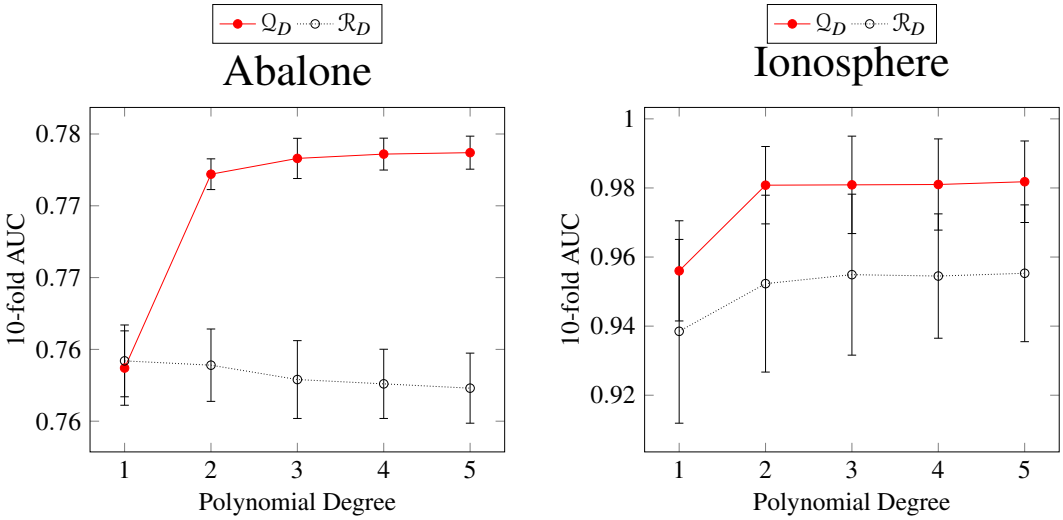


Figure 9.5: Nested 10-fold AUC with standard deviation using EasyMKL with $\mathcal{Q}_D$ and $\mathcal{R}_D$ with different values of D.

tive sets of base kernels $\mathcal{Q}_D = \{\mathbf{K}_0^{(Q)}, \ldots, \mathbf{K}_D^{(Q)}\}$ and $\mathcal{R}_D = \{\mathbf{K}_0^{(R)}, \ldots, \mathbf{K}_D^{(R)}\}$. Specifically, for the two families of base kernels, we considered the same set of features (monomials with degrees less or equal to D), but they are assigned to the base kernels in the family in a different way. When generating $\mathcal{Q}_D$, the features are assigned to the kernels according to the degree rule, that is features of degree d are assigned to the kernel $\mathbf{K}_d^{(Q)}$. On the other hand, when generating $\mathcal{R}_D$, the features are assigned randomly to one of its base kernels.

It is well known that the number of all the possible combinations with repetition of $d \in \{0, \ldots, D\}$ features, picked from a set of m features, is equal to $N_d = m^d$. Exploiting N_d we are able to define the following distribution over the D degrees:

$$\pi(d) = \frac{N_d}{\sum_{j=0}^{D} N_j}, \quad d \in \{0, \ldots, D\}. \tag{9.11}$$

Algorithm 2 gives an effective procedure to generate families of $D+1$ base kernels using the features of HPKs of degrees less or equal to a given D. Basically, the algorithm draws a random feature from the feature space and assign it to a base kernel selected with a function $\mathcal{S}$ which is a parameter of the algorithm. In particular, we invoked the algorithm using $\mathcal{S}(d) = d$ for the base kernel family $\mathcal{Q}_D$ and $\mathcal{S}(d) = \text{random}(0, \ldots, D)$ for the base kernel family $\mathcal{R}_D$. Finally, it normalizes all the base kernels generated.

Sets $\mathcal{Q}_D$ and $\mathcal{R}_D$ have been generated for different values of D, with number of steps fixed to $10,000$ for two different datasets. The stratified nested 10-fold cross validation results are reported in Figure 9.5 for the Abalone and the Ionosphere datasets.

At least empirically, these results highlight the importance of the principled structure

Algorithm 2 Approximation of *weak* kernels family.
The symbol $\bigodot$ stands for the entry-wise multiplication among vectors of the same dimension.

Require: $\mathbf{X}, D, steps, \mathcal{S} : \{0,\dots,D\} \to \{0,\dots,D\}$
Ensure: A kernel family $\mathcal{K}_D = \{\mathbf{K}_0,\dots,\mathbf{K}_D\}$.

for $s = 0$ **to** D **do**
  $\mathbf{K}_s = \mathbb{O}$
end for
for $i = 1$ **to** $steps$ **do**
  pick $d \in \{0,\dots,D\}$ according to the distribution $\pi(d)$
  if $d = 0$ **then**
    $\mathbf{H} = \mathbf{1}_L \mathbf{1}_L^\top \in \mathbb{R}^{L\times L}$
  else
    set $\mathbf{f}$ a list of random values in $[1,\dots,m]$ (with replica)
    $\mathbf{x_f} = \bigodot_{j\in\mathbf{f}} \mathbf{X}[:, j]$
    $\mathbf{H} = \mathbf{x_f} \cdot \mathbf{x_f}^T$
  end if
  set $s = \mathcal{S}(d) \in \{0,\dots,D\}$ (apply the selector)
  $\mathbf{K}_s = \mathbf{K}_s + \mathbf{H}$
end for
for $s = 0$ **to** D **do**
  Normalize the kernel matrix $\mathbf{K}_s$
end for
return $\{\mathbf{K}_0,\dots,\mathbf{K}_D\}$

imposed to $\mathcal{Q}_D$ to obtain good results with EasyMKL. As presented in the previous sections, EasyMKL exploits only the margin in its objective function and has a fixed amount of weight to distribute (i.e. the vector of the weights has a fixed norm). With a MSC family of weak kernels, EasyMKL is allowed to exclude the kernels with a spectral complexity $\mathcal{C}$ that is too high or too low to generate margin between positive and negative examples.

The idea can be explained by using a simple example. In the case of binary vectors, monomial of low degree can be considered "simple" (or "popular") features as likely they have value 1 on many examples. Monomials of high degree are considered "elaborated" (or "rare") features since they have value 1 on very few examples. The intuition is that nor the simpler features neither the more complex ones can be optimal to get a large margin for many examples. In this sense, MKL helps us to find the best combination of such groups of features (kernels).

Studying the weights assigned by EasyMKL, we noticed that the best solution that EasyMKL is able to find for the family $\mathcal{R}_D$ is near to the average kernel (i.e. all the weights are proportional to $\frac{1}{D+1}$). This means that using the EasyMKL approach with this family of weak kernels is not effective. These results highlight how a principled splitting of the information can make the difference on MKL results.

Analysis of the spectral complexity

In this section, the study we have performed about is presented $\mathbf{K}_{MKL}$, $\mathbf{K}_D$ and $\mathbf{K}_{sum}$ spectral complexities on the benchmark datasets. For all the values of D, $\mathbf{K}_{MKL}$ have shown a complexity trapped between the spectral complexity $\mathbf{K}_D$ and the one of $\mathbf{K}_{sum}$. Figure 9.6 summarizes these results.

These results confirm the theoretical finding about the monotonicity of the spectral complexity for weak kernels in $\mathcal{K}_D$ (i.e. the MSC property of $\mathcal{K}_D$).

Analysis of the weights assigned to the weak kernels

In this section, we present an analysis of the weights that EasyMKL assigns to the kernels in the *weak* family $\mathcal{K}_D$ for different values of D. Figure 9.7 reports the histograms weights for $D = 3, 5, 10$ and for two datasets: Heart and Splice.

These results show how the optimal distribution of the weights, learned by EasyMKL, is not the trivial choice of a single value of complexity but is a combination of different kernels with different complexities. In Heart, the weights are anti-correlated with respect to the degree of the *weak* kernels. However, this behavior is not true in general, in fact, for the Splice dataset, most of the weight is shared among *weak* kernels of degree in the range $[1, 5]$.

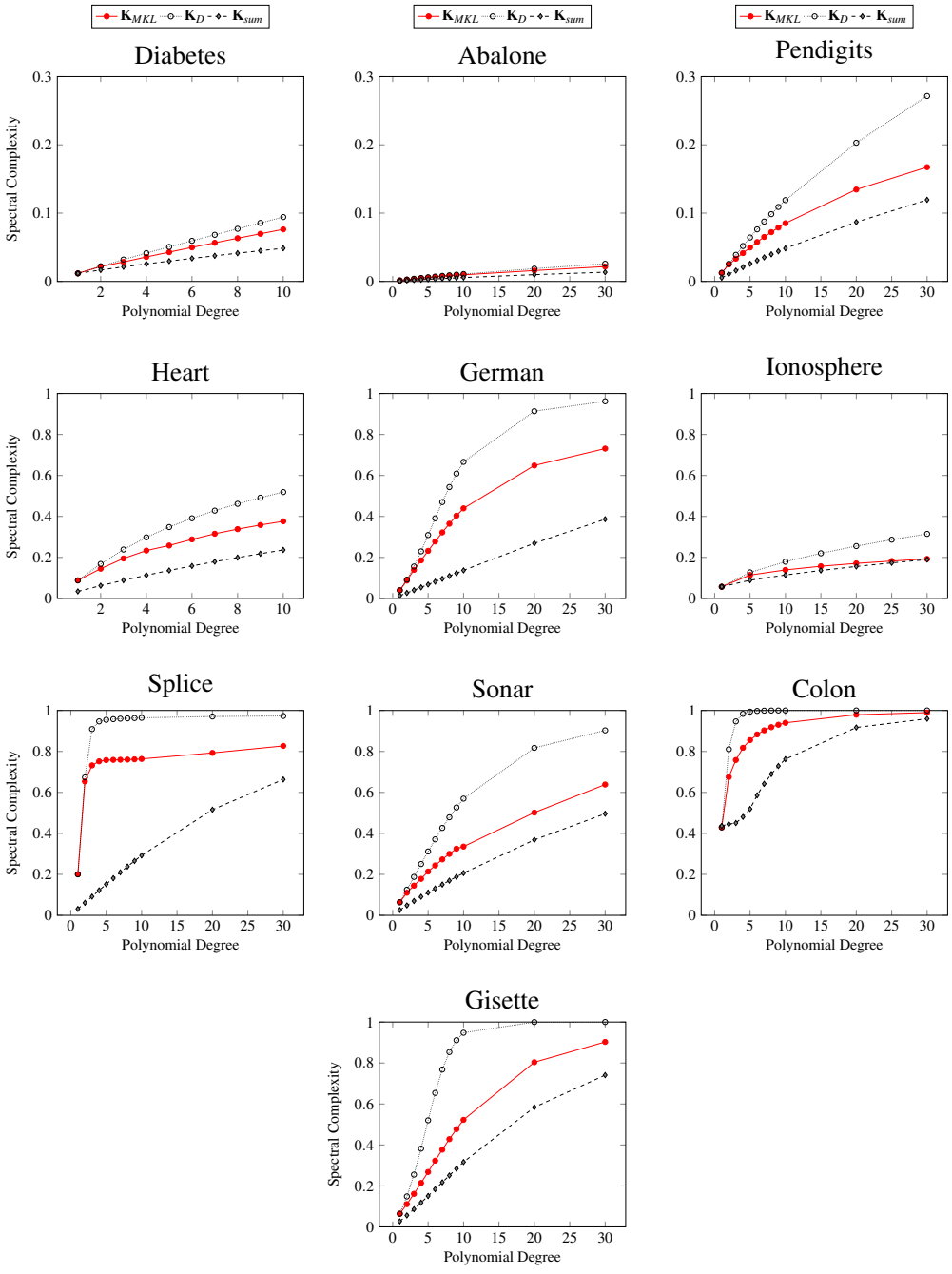


Figure 9.6: Spectral complexity evaluated for all the ten benchmark datasets for different values of D. Our proposed solution $\mathbf{K}_{MKL}$ (solid red line) has been compared to the baselines $\mathbf{K}_D$ and $\mathbf{K}_{sum}$.

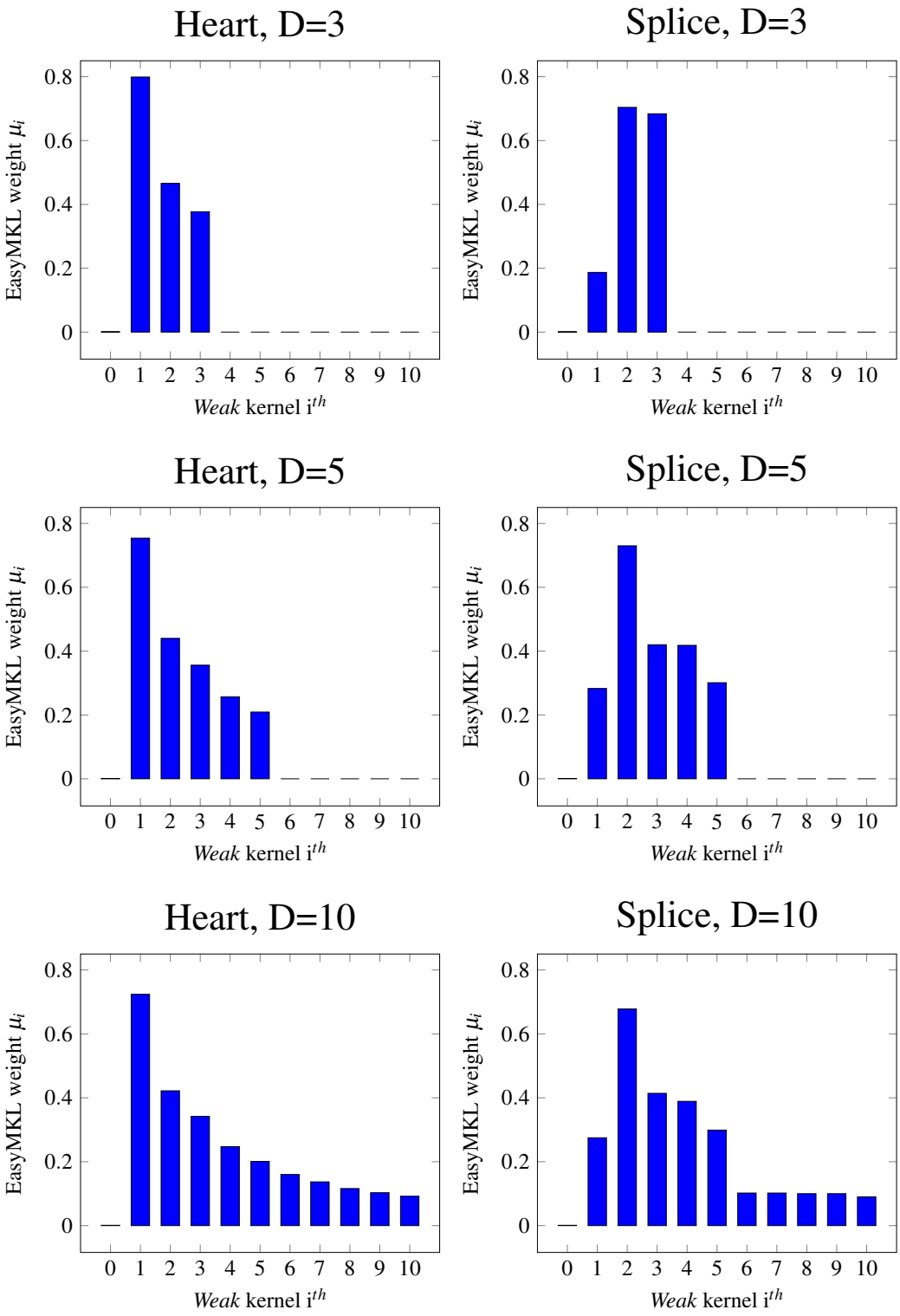


Figure 9.7: The assigned weights μ_i by using EasyMKL for the *weak* kernels in the families $\mathcal{K}_D$ for $D = 3, 5, 10$ of Heart and Splice datasets.

Dataset	$\mathbf{K}_{MKL}$	$gHKL_{1.1}$	$gHKL_{1.5}$	$gHKL_{2.0}$
Haberman	$0.716_{\pm 0.014}$	$0.617_{\pm 0.166}$	$0.518_{\pm 0.110}$	$0.556_{\pm 0.070}$
Liver	$0.689_{\pm 0.056}$	$0.565_{\pm 0.109}$	$0.583_{\pm 0.110}$	$0.623_{\pm 0.038}$
Diabetes	$0.842_{\pm 0.027}$	$0.636_{\pm 0.118}$	$0.733_{\pm 0.058}$	$0.766_{\pm 0.046}$
Australian	$0.924_{\pm 0.081}$	$0.923_{\pm 0.101}$	$0.918_{\pm 0.049}$	$0.920_{\pm 0.045}$

Table 9.1: Nested 10-fold $\text{AUC}_{\pm std}$ using EasyMKL ($\mathbf{K}_{MKL}$) with $\mathcal{K}_{10}$ as *weak* family compared to gHKL_{ρ} with $\rho \in \{1.1, 1.5, 2.0\}$.

9.4.3 A comparison with the Generalized Hierarchical Kernel Learning

The Generalized Hierarchical Kernel Learning (gHKL) is a generalization of the MKL. A graph among the features is created to obtain better combination of kernels, as depicted in Section 3.3.3. In this section, a performance comparison between EasyMKL with DPP families of *weak* kernels and the gHKL algorithm will be presented. In these experiments, we used the implementation of the $gHKL_{\rho}$ algorithm provided by the authors[1].

We performed a 10-fold cross validation for the AUC evaluation, tuning the parameter C of the SVM for $gHKL_{\rho}$ [105] with a 3-fold cross validation selecting C in $\{10^i : i = -3, \ldots, 3\}$. The same procedure has been repeated for $\rho \in \{1.1, 1.5, 2.0\}$. The number of *weak* kernels is fixed to 2^m, where m is the number of features, as in the original paper [105]. It is important to point out that, with $\rho = 2$ the HKL formulation of Bach [14] is obtained.

For our algorithm, we fixed $D = 10$ (i.e. the family of MSC *weak* kernels is $\mathcal{K}_{10}$) and performed the validation of the parameter Λ of EasyMKL by using the same methodology (3-fold cross validation) with $\Lambda \in \{0.0, 0.1, \ldots, 0.9, 1.0\}$.

In Table 9.1, the AUC results are presented. From these results, we can note how our solution ($\mathbf{K}_{MKL}$) outperforms the $gHKL_{\rho}$ method in this task.

9.5 *Weak* kernels on the ODD Graph-Kernel

Kernels for structures, including graphs, generally suffer of the diagonally dominant gram matrix issue, the effect by which the number of sub-structures, or features, shared between instances are very few with respect to those shared by an instance with itself. A parametric rule is typically used to reduce the weights of largest (more complex) sub-structures.

The particular rule which is adopted is in fact a strong external bias that may strongly affect the resulting predictive performance. Thus, in principle, the applied rule should be validated in addition to the other hyper-parameters of the kernel. Nevertheless, for the majority of graph kernels proposed in literature, the parameters of the weighting rule are

[1] `http://www.cse.iitb.ac.in/~pratik.j/ghkl/`

fixed a priori.

In the following sections, we will describe a principled way to define MSC families of *weak* kernels starting from the state-of-the-art ODD kernel framework [57].

The benefits of our approach are twofold. Firstly, with our approach, we do not need a parametric weighting rule for the graph kernel. Instead, the EasyMKL algorithm will automatically tune the weight of different kernels. Secondly, the performance of the resulting kernel empirically improves upon the base kernel. Moreover, the computational complexity of the resulting learning procedure is the same as the classical approach (consisting in the kernel computation and the application of an SVM).

9.5.1 Generating the weak kernels

When an explicit RKHS representation is available, it is possible to define a function mapping features to a limited set of natural numbers, corresponding to buckets. Then, each weak kernel is defined as the dot product that considers just the features belonging to a particular bucket.

More formally, we recall that h has been defined as a bijective function mapping features (trees in our setting) to natural numbers. Let us define a (possibly non injective) function $d : \mathcal{H} \to \Sigma$, where $\mathcal{H}$ is the RKHS and $\Sigma = \{0,1,2,\ldots,n\}$, mapping trees to a finite set of buckets. This function assigns each feature to a specific bucket. Also, d will be used to define the weak kernels, i.e. features that are mapped to the same value will be part of the same kernel. Let us define the $j-$th component of the $i-$th weak kernel in the sequence as:

$$\phi^i_j(G) = \phi_j(G)\delta(d(h^{-1}(j)),i) \tag{9.12}$$

where δ is the Kronecker's delta function.
Then we define the *i*-th weak kernel as

$$K_i(G_1,G_2) = \langle \boldsymbol{\phi}^i(G_1), \boldsymbol{\phi}^i(G_2) \rangle. \tag{9.13}$$

Let us consider the ODD_{ST} kernel defined in Section 4.1.1 and its corresponding explicit feature space representation. Every feature in this space encodes a specific labeled tree. Intuitively, the strategy we decided to employ is to generate a weak kernel corresponding to each possible height of the tree features, i.e. we group in a particular weak kernel all the trees in the RKHS having the same height.

The motivation of such a choice is the following. The features of ODD_{ST} are not independent because, as detailed in Section 4.1.1, if a tree is present in the explicit feature space representation of a graph, then all its proper subtrees will do. This relation describes a partial order relation between features, where $a \dot{<} b$ if and only if a is isomorphic to a

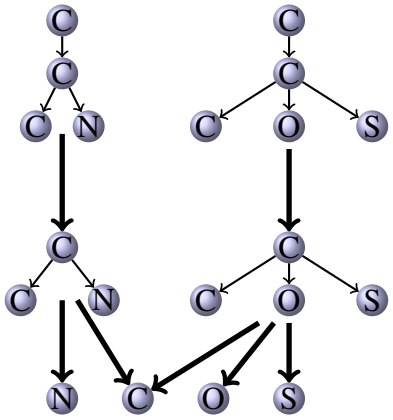


Figure 9.8: Example of hierarchy among tree features. Thick arrows encodes the "has as proper subtree" relationship.

sub-tree of b. Figure 9.8 shows an example of some features and draws the partial order relationship induced by the "has as proper subtree" relationship. A necessary condition for two features to be in the relationship is that the height of the first feature is greater than the height of the second feature. Thus, the simplest way to define a $d()$ function in such a way that two dependent features do not end up in the same bucket is to define $d(t) = height(t)$ where $height(t)$ is a function returning the height of a tree (that is the maximum distance between the root and a leaf node).

Then, the j-th weak kernel $\mathbf{K}_j$ comprehends all the tree-features with a fixed height j. Let us recall that the original ODD_{ST} kernel depends on the r parameter, determining the maximum height of the performed DAG visits, and on a λ parameter, determining the weight of the features as a function of their size. With our proposed approach, for a fixed kernel, r weak kernels will be generated, indexed with $h \in \{0,1,\ldots,r\}$. The λ parameter is no longer required, since the learning algorithm will weight the different buckets of features. This formulation is particularly convenient from a computational point of view and the weak kernels can be generated from a slight modification of the original kernel algorithm. The resulting explicit feature space representation of the i-th kernel is:

$$\phi_j^i(G) = \phi_j(G)\delta(height(h^{-1}(j)), i). \tag{9.14}$$

This feature grouping scheme is simple, but it is designed in order to behave in a very specific way. In fact, each sub-kernel will have a weight assigned by exploiting EasyMKL. This means that the features in the same bucket will share the same weight. Since all the proposed feature weighting schemes depend in some sense from the size of the features, it is reasonable to maintain this intuition in our bucketing function.

h/data	CAS $r^* = 3$	CPDB $r^* = 3$	AIDS $r^* = 8$	NCI1 $r^* = 4$	GDD $r^* = 2$
0	0.002	0.008	0.004	0.001	0.005
1	0.006	0.025	0.009	0.004	0.183
2	0.021	0.080	0.023	0.012	0.999
3	0.151	0.480	0.110	0.089	
4			0.229	0.368	
5			0.306		
6			0.375		
7			0.468		
8			0.683		
$\mathbf{K}_{sum}$	0.015	0.051	0.074	0.017	0.061
$\mathbf{K}_{MKL}$	0.095	0.215	0.275	0.165	0.281

Table 9.2: spectral complexity of the weak kernels, for each dataset and for each value of $h \leq r^*$ compared to the spectral complexity of $\mathbf{K}_{sum}$ and $\mathbf{K}_{MKL}$.

Complexity of the weak kernels

For all the weak kernels combined using EasyMKL we have evaluated the spectral complexity. For each dataset, the weak kernels evaluated are $\mathbf{K}_1, \ldots, \mathbf{K}_{r^*}$, where r^* is the best hyper-parameter r obtained in validation. The results are shown in Table 9.2 and Figure 9.9. The results highlight how our segmentation of the features induces a structure on the kernels based on their spectral complexity. In particular, the h parameter of the single weak kernel is monotonically connected to the spectral complexity. This result is correlated to the popularity[2] of the features among the graphs of the dataset. In general, sub-structures that are simpler (i.e with low r) exist among a larger number of graphs with respect to features that are more complex (i.e. with high r).

The kernel $\mathbf{K}_{sum}$ uses all the features contained in the weak kernels with the same weight and it obtains a very low spectral complexity. Table 9.2 shows that the difference between $\mathbf{K}_{sum}$ and $\mathbf{K}_{MKL}$ is large in terms of spectral complexity. Specifically, for all the datasets the spectral complexity of $\mathbf{K}_{MKL}$ is bigger with respect to the one of $\mathbf{K}_{sum}$ and is lower than the one of the weak kernel $\mathbf{K}_{r^*}$ (i.e. $R(\mathbf{K}_{r^*}) > R(\mathbf{K}_{MKL}) > R(\mathbf{K}_{sum})$). We have empirically proved that neither using all the weak kernels summed with the same weight nor using only the more complex features (contained in$\mathbf{K}_{r^*}$) is the optimal solution given a specific task.

[2]The popularity is an estimation of the probability of the existence of a specific feature in a randomly picked example from the dataset.

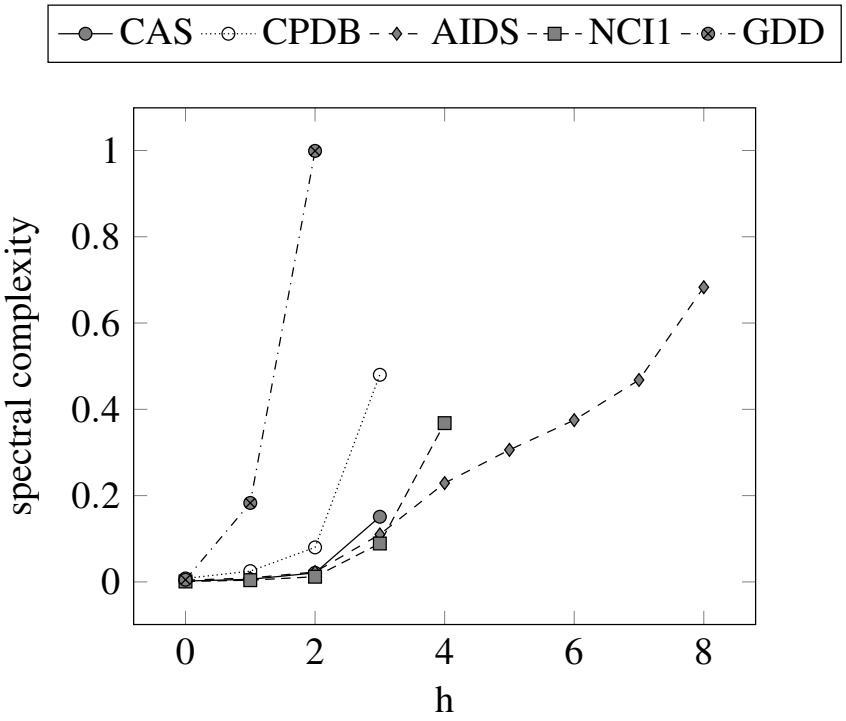


Figure 9.9: spectral complexity of the weak kernels, for each dataset and for each value of $h \leq r^*$.

9.6 Experiments

In this section, we compare our proposed ODD-MKL approach to the ODD_{ST} kernel that we considered as the base kernel. This kernel has shown state-of-the-art predictive performances [57]. Moreover, as baseline of the grouping method of the features, we created random splits of features of ODD_{ST} generating weak kernels without any hierarchical structure. We compared the MKL results obtained using this family of weak kernels with our proposed method that, on the other hand, adopts a principled way to split the features among the different weak kernels.

For the original ODD_{ST} kernel, we adopted an SVM classifier [149] given its state-of-the-art predictive performances. The C parameter has been selected in the range $\{0.001, 0.01, \ldots, 10000\}$, and the parameters of the kernel:

- $h \in \{1, 2, \ldots, 8\}$,
- $\lambda \in \{0.5, 0.6, \ldots, 1.6\}$.

As for the methods adopting the MKL approach, the Λ parameter of EasyMKL has been validated in $\{0, 0.1, 0.2, \ldots, 1.0\}$. We compared the different methods on five real-world graph datasets from bioinformatics: CAS[3], CPDB [91], AIDS [198], NCI1 [194] and GDD [63] (see Section 2.2.1)

[3] http://www.cheminformatics.org/datasets/bursi

Data/Kernel	ODD_{ST}	ODD-MKL
CAS	$0.8982_{\pm 0.0017}$	$0.9049_{\pm 0.0008}$
CPDB	$0.8442_{\pm 0.0067}$	$0.8564_{\pm 0.0056}$
AIDS	$0.8262_{\pm 0.0052}$	$0.8515_{\pm 0.0031}$
NCI1	$0.9069_{\pm 0.0010}$	$0.9144_{\pm 0.0008}$
GDD	$0.8473_{\pm 0.0038}$	$0.8498_{\pm 0.0026}$

Table 9.3: Average AUC results$_{\pm std}$ of the original ODD_{ST} kernel and the proposed ODD-MKL kernel, in nested 10-fold cross validation.

9.6.1 Results

Table 9.3 reports the AUC results of the base kernel (ODD_{ST}) and our proposed MKL approach in *nested* 10-fold cross validation, where the hyper-parameters of the different methods are selected using the training dataset only, using an inner 10-fold cross validation for each split of the 10-fold. The whole procedure has been repeated 10 times, and we reported the average and standard deviation of the 10 runs. From the table emerges that the proposed approach is able to improve the predictive performance with respect to the baseline in all the considered datasets.

In order to investigate if the proposed bucketing function makes sense, or if the performance improvements are due just to the fact that the algorithm has more degrees of freedom, we compared the proposed feature grouping method with a random method. We set, for each dataset, the best performing kernel parameter r^* (i.e. the parameter that defines the maximum height of the extracted tree). We computed a new feature bucketing function that randomly assigns each feature to one of buckets. We repeated the experiment with different random seeds in order to mitigate the fluctuations in performance due to the random assignments of the features. Table 9.4 reports the results of such an experiment. In all the datasets the random approach performs worse than out proposed method, and in 4 out of 5 datasets it performs worse than the base kernel. These results confirms that the hierarchical relationship in the feature grouping criterion is a key factor in the definition of the weak kernels to successfully apply MKL.

Data/Kernel	ODD_{ST}	ODD-MKL	ODD-MKL-random
CAS	0.8983	0.9049	0.8762
CPDB	0.8576	0.8676	0.8386
AIDS	0.8401	0.8706	0.8480
NCI1	0.9085	0.9159	0.8782
GDD	0.8426	0.8507	0.8299

Table 9.4: Average AUC results of the original ODD $_{ST}$ kernel, the proposed ODD-MKL kernel, and the random split MKL kernel in 10-fold cross validation.

9.7 Summary of the results

In this chapter, we have proposed a new point of view in the MKL framework.

Firstly, we have introduced a new general method to learn the kernel as a combination of base homogeneous polynomial kernels of different inherent complexities, exploiting their structure. Our method learns the coefficients of a dot-product polynomial kernel, a class of kernels that includes the polynomial kernels and the Gaussian kernel (RBF). We have given empirical evidence that our method is always competitive and often perform better than state-of-the-art kernels on ten benchmark datasets.

Secondly, we proposed a way to improve the predictive performance of graph kernels. We have generated a set of weak kernels from a graph kernel (ODD_{ST}) grouping the features on the basis of the associated MSC set of weak kernels, and then applied MKL to the weak kernels.

From the experimental results, we can claim that the new kernels outperform the base kernels with respect to the AUC measure. It is important to highlight that this result is reached using our principled grouping of the features injected to the weak kernels. In fact, using random bunches of features the new kernels obtained from the MKL algorithm have unsatisfying performances (that are even worse than the individual base kernels).

10. Learning the Anisotropic RBF

In this chapter, we propose to combine ideas from Distance Metrical Learning (DML), Kernel Learning (KL) and Multiple Kernel Learning (MKL). Specifically, we focus on the family of anisotropic RBF kernels, that are a generalization of the RBF kernels. Following the same idea presented in Chapter 9, we propose to use our method to create an optimal family of *weak* kernels for a MKL algorithm with fixed rule. This rule is implicitly defined by the structure of the anisotropic RBF kernel.

Part of this chapter has been published in [3].

10.1 Connection between MKL and DML

The most popular kernel is undoubtedly the RBF kernel, which is a general purpose kernel that is based on the Euclidean distance between examples. Similarly to the Euclidean distance, the RBF kernel gives an equal weight to different features and the strength of this weight depends on a single external parameter that needs to be tuned against validation data. However, it is well known that different features typically have unequal impact and importance in solving a given classification task.

Combining ideas from DML, KL and MKL, we focus on the family of anisotropic RBF kernels in the form:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{M} (\mathbf{x}_i - \mathbf{x}_j)), \tag{10.1}$$

where $\mathbf{M} = \mathbf{diag}(\boldsymbol{\beta})$ is the diagonal matrix created using the vector $\boldsymbol{\beta} \in \mathbb{R}^m$ of parameters

(one value for each feature) to learn. The method proposed extends the KOMD method (see Chapter 5) to learn an *Anisotropic RBF* (ARBF) from data maintaining the same game theoretical setting where players compete and the value of the game consists of the separation between positive and negative training data.

10.2 Extending the game to features

We propose to extend the game illustrated in Section 5.1 by considering an additional player which selects the kernel matrix $\mathbf{K}$ from the family of Anisotropic RBF kernel. The RBF kernel is defined by

$$K(\mathbf{x}_i,\mathbf{x}_j) = \exp(-\beta_0\|\mathbf{x}_i-\mathbf{x}_j\|_2^2) = \exp(-(\mathbf{x}_i-\mathbf{x}_j)^\top\beta_0\mathbf{I}(\mathbf{x}_i-\mathbf{x}_j)) \tag{10.2}$$

where $\beta_0 \in \mathbb{R}_+$ is an external parameter. The RBF kernel can also be seen as using the trivial metric $\mathbf{M} = \beta_0\mathbf{I} = \mathbf{diag}(\beta_0,...,\beta_0)$. In the anisotropic RBF we have a generalized metric $\mathbf{M} = \mathbf{diag}(\beta^{(1)},...,\beta^{(m)})$ and we can write the anisotropic RBF as:

$$K_\beta(\mathbf{x}_i,\mathbf{x}_j) = \prod_{r=1}^{m}\exp(-\beta^{(r)}(\mathbf{x}_i^{(r)}-\mathbf{x}_j^{(r)})^2), \quad \boldsymbol{\beta}\in\mathbb{R}_+^m \tag{10.3}$$

where $\mathbf{x}_i^{(r)}$ is the r^{th} feature of the i^{th} example and $\beta^{(r)} \in \mathbb{R}_+$.

This new formulation has a greater number of degrees of freedom than the classical RBF kernel. A useful observation is that $K_{\boldsymbol{\beta}}(\cdot,\cdot)$ can be seen as a element-wise product of kernels evaluated on a single feature. More formally:

$$K_{\boldsymbol{\beta}} = \bigotimes_{r=1}^{m} K_{\beta^{(r)}} \tag{10.4}$$

with $K_{\beta^{(r)}}$ the RBF kernel defined on the r^{th} feature only with parameter $\beta^{(r)}$.

From this point of view, we are looking for the best family of *weak* kernels $\{K_{\beta^{(r)}} : r = 1,\ldots,m\}$ in order to apply a MKL algorithm with a fixed heuristic rule. In this case, the rule is the entry-wise multiplication of the kernels $K_{\beta^{(r)}}$ as in Equation 10.4.

Note that one can also interpret the *Anisotropic* RBF as a standard RBF where the norm $\|\cdot\|$ is not the basic Euclidean norm, but it is a weighted 2-norm of the form $\|\mathbf{z}\|_\beta = \sqrt{\mathbf{z}^t\mathbf{M}\mathbf{z}}$ with diagonal matrix $\mathbf{M} = \mathbf{diag}(\boldsymbol{\beta})$. In fact, any symmetric positive definite matrix $\mathbf{M}$ could be used to define an admissible norm. Using matrix notation, the anisotropic RBF can be written as

$$K_\beta(\mathbf{x}_i,\mathbf{x}_j) = \exp(-(\mathbf{x}_i-\mathbf{x}_j)^t\mathbf{diag}(\boldsymbol{\beta})(\mathbf{x}_i-\mathbf{x}_j)). \tag{10.5}$$

Then, finding the best parameters for an *Anisotropic* RBF is a DML problem and we need to optimize the kernel representation by finding a trade-off between the components of $\boldsymbol{\beta}$.

Starting from the KOMD framework, we define an additional player that sets the parameters of the anisotropic RBF. This player will prefer uncorrelated features so to avoid redundancies. For this reason we define a redundancy (or correlation matrix) $\mathbf{C}$ among the features $\mathbf{f}_1, ..., \mathbf{f}_m$, defined using an RBF kernel with parameter τ and normalized with respect to the number of features. The correlation matrix $\mathbf{C} \in \mathbb{R}_+^{m \times m}$ can be generated using the following rule:

$$\mathbf{C}[i, j] = \exp(-\frac{\tau}{m}\|\mathbf{f}_i - \mathbf{f}_j\|_2^2) \ \forall i, j = 1, ..., m. \tag{10.6}$$

Finally, we propose to use the following regularized optimization problem as objective for the player $\boldsymbol{\beta}$:

$$\max_{\boldsymbol{\beta} \in \mathbb{R}_+^m} \ \mathcal{Q}(\boldsymbol{\beta}, \boldsymbol{\gamma}) - \mu \mathcal{C}(\boldsymbol{\beta}) \tag{10.7}$$

where $\mathcal{Q}(\boldsymbol{\beta}, \boldsymbol{\gamma}) = \boldsymbol{\gamma}^\top \mathbf{Y}\mathbf{K}(\boldsymbol{\beta})\mathbf{Y}\boldsymbol{\gamma}$ and $\mathcal{C}(\boldsymbol{\beta}) = \frac{1}{2}\boldsymbol{\beta}^t \mathbf{C}\boldsymbol{\beta}$.

Note that, the proposed type of regularizer differs significantly from the usual trace regularizer used in kernel learning. As already noted, the trace regularizer does not represent the notion of complexity of a kernel in terms of the space of functions that can be generated using that kernel. For example, all RBF kernels have the same trace independently from the RBF parameter weighting the distance between examples, while the complexity of the resulting kernels can be dramatically different. On the other side, the correlation of the features on the parameters $\boldsymbol{\beta}$ that we propose, well fits the idea that good features are more useful if they represent different points of view of the examples.

Summarizing, the extended game we propose has value $\mathcal{Q}(\boldsymbol{\beta}, \boldsymbol{\gamma})$ and the two players individually aim at optimizing their strategies according to the following optimization problems:

$$\mathrm{P}_{\boldsymbol{\gamma}}: \quad \min_{\boldsymbol{\gamma} \in \Gamma} \quad (1 - \Lambda)\mathcal{Q}(\boldsymbol{\beta}, \boldsymbol{\gamma}) + \Lambda||\boldsymbol{\gamma}||^2 \tag{10.8}$$

$$\mathrm{P}_{\boldsymbol{\beta}}: \quad \max_{\boldsymbol{\beta} \in \mathbb{R}_+^d} \quad \mathcal{Q}(\boldsymbol{\beta}, \boldsymbol{\gamma}) - \mu \mathcal{C}(\boldsymbol{\beta}) \tag{10.9}$$

In the following, we give the simple alternating algorithm we used to solve the multi-objective problem given above.

Algorithm 3 ARBF algorithm

Find the best β_0 and Λ with KOMD validation
for t=1,...,T **do**
  Set $\boldsymbol{\beta} = \boldsymbol{\beta}_{t-1}$ and generate a solution $\boldsymbol{\gamma}_t$ optimizing the problem in Eq. 10.8
  Set $\boldsymbol{\gamma} = \boldsymbol{\gamma}_t$ and generate a solution $\boldsymbol{\beta}_t$ optimizing the problem in Eq. 10.9
end for

The first step, i.e. the optimization of $\boldsymbol{\gamma}_t$, is performed by solving the Equation 10.8 and it is equal to solve a KOMD optimization problem using the kernel $\mathbf{K}(\boldsymbol{\beta}_{t-1})$.

10.2.1 Gradient based optimization for $P_{\boldsymbol{\beta}}$

The function $\mathcal{Q}(\boldsymbol{\beta})$ in Eq. 10.9 has the r^{th} component of the gradient equal to:

$$\frac{\partial \mathcal{Q}(\boldsymbol{\beta})}{\partial \beta^{(r)}} = -\sum_{i,j} y_i y_j \gamma_i \gamma_j K_\beta(\mathbf{x}_i, \mathbf{x}_j)(\mathbf{x}_i^{(r)} - \mathbf{x}_j^{(r)})^2 = -\boldsymbol{\gamma}^\top \mathbf{Y}(\mathbf{D}_r \otimes \mathbf{K}_\beta)\mathbf{Y}\boldsymbol{\gamma}, \tag{10.10}$$

where $\mathbf{D}_r \in \mathbb{R}^{n \times n}$ is the symmetric matrix of pairwise squared differences of the r^{th} feature, that is, $\mathbf{D}_r[i,j] = (\mathbf{x}_i^{(r)} - \mathbf{x}_j^{(r)})^2$. Then, the partial derivative of Eq. 10.9 with respect to $\beta^{(r)}$ will be:

$$\frac{\partial \mathcal{Q}(\boldsymbol{\beta})}{\partial \beta^{(r)}} - \mu \mathbf{C}[r,:]\boldsymbol{\beta}, \tag{10.11}$$

where $\mathbf{C}[r,:]$ is the r^{th} row of $\mathbf{C}$.

10.2.2 Concavity of $P_{\boldsymbol{\beta}}$

It is simple to evaluate that the Hessian matrix $\mathbf{H}$ of the function $\mathcal{Q}(\boldsymbol{\beta},\boldsymbol{\gamma}) - \mu\mathcal{C}(\boldsymbol{\beta})$ can be written as:

$$\mathbf{H}[i,j] = \gamma^T \mathbf{Y}(\mathbf{D}_i \otimes \mathbf{D}_j \otimes \mathbf{K}_\beta)\mathbf{Y}\gamma - \mu \mathbf{C}[i,j]. \tag{10.12}$$

Then, we can claim that with a sufficiently large value of μ and τ the function $\mathcal{Q}(\boldsymbol{\beta},\boldsymbol{\gamma}) - \mu\mathcal{C}(\boldsymbol{\beta})$ is concave. In fact, the following proposition holds:

Proposition 10.2.1 Let be $\mathbf{H}$ the Hessian matrix of the function $\mathcal{Q}(\boldsymbol{\beta},\boldsymbol{\gamma}) - \mu\mathcal{C}(\boldsymbol{\beta})$, then $\exists \Upsilon, \Psi \in \mathbb{R}^+$ such that $\forall \mu \geq \Upsilon, \tau \geq \Psi$, the matrix $\mathbf{H}$ is negative definite.

Proof. We have that a diagonal entry of the matrix $\mathbf{H}$ is in the form

$$\mathbf{H}[i,i] = \gamma^T \mathbf{Y}(\mathbf{D}_i \otimes \mathbf{D}_i \otimes \mathbf{K}_\beta)\mathbf{Y}\gamma - \mu, \tag{10.13}$$

observing that $\mathbf{C}[i,i] = 1$ for definition. Now, we will prove that the matrix $-\mathbf{H}$ is positive definite. Firstly, we can find the bounds for the values outside the diagonal of $-\mathbf{H}$ using the eigenvalues $g_1^{i,j} \geq g_2^{i,j} \geq \cdots \geq g_m^{i,j}$ of the matrix $\mathbf{D}_i \otimes \mathbf{D}_j \otimes \mathbf{K}_\beta$. Due to the

particular structure of this matrix, we have that $g_m^{i,j} = -g_1^{i,j} \leq 0$. Also, we can define $g_h = \max\{g_h^{i,j} : i,j = 1,\ldots,m\} \ \forall h = 1,\ldots,m$. Then, the following bounds hold:

$$-\mathbf{H}[i,j] = -\gamma^T \mathbf{Y}(\mathbf{D}_i \otimes \mathbf{D}_j \otimes \mathbf{K}_\beta)\mathbf{Y}\gamma + \mu\mathbf{C}[i,j] \leq 2g_1 + \mu\mathbf{C}[i,j], \tag{10.14}$$

$$-\mathbf{H}[i,j] \geq 2g_m = -2g_1. \tag{10.15}$$

Secondly, we can define the bounds for the diagonals entry of $-\mathbf{H}$:

$$-\mathbf{H}[j,j] \leq 2g_1 + \mu, \tag{10.16}$$

$$-\mathbf{H}[j,j] \geq 2g_m + \mu = -2g_1 + \mu. \tag{10.17}$$

We are interested in finding a value Υ such that $\forall \mu \geq \Upsilon$ the matrix $-\mathbf{H}$ is diagonally dominant. The definition of a matrix diagonally dominant in this case is equivalent to:

$$\sum_{\substack{i=1 \\ i\neq j}}^{m} (2g_1 + \mu\mathbf{C}[i,j]) \leq |-2g_1 + \mu| \ \ \forall j = 1,\ldots,m. \tag{10.18}$$

$$-\mathbf{H}[j,j] \geq 0 \ \ \forall j = 1,\ldots,m. \tag{10.19}$$

Now, we can fix $\mu \geq 2g_1$ in order to satisfy the condition 10.19 and then we have that:

$$|-2g_1 + \mu| = \mu - 2g_1 \ \ \forall j = 1,\ldots,m. \tag{10.20}$$

So, the Equation 10.18 is equivalent to the following:

$$g_1(2m+2) \leq \mu(1 - \sum_{\substack{i=1 \\ i\neq j}}^{m} \mathbf{C}[i,j]) \ \ \forall j = 1,\ldots,m. \tag{10.21}$$

We have now to observe that $(1 - \sum_{i\neq j}^{m} \mathbf{C}[i,j]) \leq 1$ but is also possible to fix a τ such that $(1 - \sum_{i\neq j}^{m} \mathbf{C}[i,j]) > 0$ for every $j = 1,\ldots,m$. In fact, this is true if and only if

$$\sum_{\substack{i=1 \\ i\neq j}}^{m} \mathbf{C}[i,j] < 1 \iff \sum_{\substack{i=1 \\ i\neq j}}^{m} \exp(-\frac{\tau}{m}\|\mathbf{f}_i - \mathbf{f}_j\|_2^2) < 1 \iff \sum_{\substack{i=1 \\ i\neq j}}^{m} \left(\exp(-\frac{1}{m}\|\mathbf{f}_i - \mathbf{f}_j\|_2^2)\right)^{\tau} < 1. \tag{10.22}$$

Excluding the case where $\exists i \neq j : \|\mathbf{f}_i - \mathbf{f}_j\|_2^2 = 0$ (i.e we have two times the same feature) and due to the monotonicity of the inequality with respect to τ, it is possible to evaluate numerically a Ψ such that the inequality holds if $\tau \geq \Psi$. We are now ready to solve the two conditions (Equations 10.16 and 10.17) of the diagonally dominant matrices with respect

to μ:

$$\mu \geq \frac{(2m+2)}{1-\sum_{i=1}^{m}\mathbf{C}[i,j]}g_1 \geq 2g_1 =, \ \forall j = 1,\ldots,m. \tag{10.23}$$

Then, we can define the bound Υ for μ as:

$$\Upsilon := \frac{(2m+2)}{1-\max_{j=1}^{m}\sum_{i=1}^{m}\mathbf{C}[i,j]}g_1. \tag{10.24}$$

Finally, Υ is such that $\forall \mu \in \Upsilon$, $-\mathbf{H}$ is diagonally dominant. $-\mathbf{H}$ is also a real symmetric matrix with only positive values as diagonal entries. These conditions are sufficient to claim that the matrix $-\mathbf{H}$ is positive definite.

■

10.2.3 Reducing the problem P_β to an unconstrained optimization problem

It is well known that solving a constrained optimization problem with gradient based optimization techniques is particularly difficult. For this, we reduced the problem to an unconstrained one by performing a simple change of variables, that is $\beta^{(r)} = e^{-\alpha^{(r)}}$. Computing the gradient with respect to variables $\alpha^{(r)}$ we obtain

$$\frac{\partial \mathcal{Q}(\alpha)}{\partial \alpha^{(r)}} = \frac{\partial \mathcal{Q}(\boldsymbol{\beta})}{\partial \beta^{(r)}}\frac{\partial \beta^{(r)}(\alpha)}{\partial \alpha^{(r)}} = (\boldsymbol{\gamma}^\top \mathbf{Y}(\mathbf{D}_r \otimes \mathbf{K}_\beta)\mathbf{Y}\boldsymbol{\gamma})e^{-\alpha^{(r)}}$$

$$\frac{\partial \mathcal{C}(\alpha)}{\partial \alpha^{(r)}} = \frac{\partial \mathcal{C}(\boldsymbol{\beta})}{\partial \beta^{(r)}}\frac{\partial \beta^{(r)}(\alpha)}{\partial \alpha^{(r)}} = \mathbf{C}[r,:]\boldsymbol{\beta}e^{-\alpha^{(r)}}$$

which leads to the following update

$$\beta^{(r)} \leftarrow e^{-\alpha^{(r)}-\mu(\frac{\partial \mathcal{Q}(\alpha)}{\partial \alpha^{(r)}}-\frac{\partial \mathcal{C}(\alpha)}{\partial \alpha^{(r)}})} = \beta^{(r)}\Delta_{\beta^{(r)}}, \tag{10.25}$$

where we set $\Delta_{\beta^{(r)}} = e^{-\mu(\frac{\partial \mathcal{Q}(\alpha)}{\partial \alpha^{(r)}}-\frac{\partial \mathcal{C}(\alpha)}{\partial \alpha^{(r)}})}$.

The simple update above leads to an easy update for the kernel as in the following,

$$\mathbf{K}_{\boldsymbol{\beta}} \leftarrow \mathbf{K}_{\boldsymbol{\beta}} \otimes \exp((1-\Delta_{\beta^{(r)}})\mathbf{D}_r), \tag{10.26}$$

where $\exp(\mathbf{M})$ denotes the element-wise exponential of a matrix $\mathbf{M}$.

10.3 Experiments and Results

We have performed the evaluation of our algorithm against six benchmark datasets of varying size, typology and complexity, and we have compared our performances with the same experiments performed using other techniques. The datasets used are *Splice*,

Dataset	*KOMD*	*K-Raw*	*K-LMNN*	*K-GB-LMNN*	$ARBF_{20}$	$ARBF_{50}$
Australian	$93.2_{\pm 1.5}$	$79.2_{\pm 4.8}$	$79.1_{\pm 5.3}$	$92.4_{\pm 9.2}$	$93.8_{\pm 2.1}$	$\mathbf{94.1}_{\pm 1.6}$
German	$79.5_{\pm 2.2}$	$66.3_{\pm 2.6}$	$65.5_{\pm 3.0}$	$78.9_{\pm 5.6}$	$80.5_{\pm 4.1}$	$\mathbf{80.7}_{\pm 2.5}$
Splice	$93.7_{\pm 1.5}$	$68.1_{\pm 4.7}$	$79.7_{\pm 3.5}$	$\mathbf{95.1}_{\pm 2.8}$	$94.2_{\pm 1.5}$	$\mathbf{95.1}_{\pm 1.4}$
Heart	$90.6_{\pm 3.4}$	$76.6_{\pm 9.5}$	$74.1_{\pm 11.2}$	$92.6_{\pm 7.5}$	$91.1_{\pm 6.0}$	$\mathbf{93.8}_{\pm 3.7}$
Diabetes	$84.0_{\pm 1.9}$	$72.0_{\pm 5.5}$	$70.9_{\pm 5.8}$	$86.3_{\pm 4.6}$	$86.9_{\pm 3.2}$	$\mathbf{87.1}_{\pm 3.1}$
Ionosphere	$97.5_{\pm 1.4}$	$88.4_{\pm 3.8}$	$89.3_{\pm 4.3}$	$97.3_{\pm 3.8}$	$97.7_{\pm 3.5}$	$\mathbf{98.0}_{\pm 3.5}$

Table 10.1: AUC % (average$_{\pm std}$) obtained against 6 datasets with N_e examples and N_f features.

Ionosphere, and *Diabetes*, *German*, *Australian* and *Heart* (see Section 2.1.2 for further details). The datasets have all the features scaled to the interval $[-1,1]$. For each dataset, we constructed several splits containing 70% of the examples for the training set, 10% of the examples for the validation set and used the remaining 20% of the examples as the test set.

We compared our algorithm **ARBF** at different number of steps T, against the following baselines and state-of-the-art techniques:

- **KOMD**: in this case, model selection has been used to find the best parameters $\Lambda \in \{0, 0.1, 0.5, 0.9\}$ and $\beta_0 \in \{0.01, 0.1, 0.5, 1.0\}$. A KOMD with standard RBF (shape parameter β_0) has been trained.
- **K-Raw**: this is kNN without learning any new metric, with validation and model selection to find the best k.
- **K-LMNN** and **K-GB-LMNN**: we used the implementation made by the authors[1] and we performed a model selection in order to find the best k for kNN.

Concerning our method, KOMD validation has been used to obtain the initial parameters (β_0 and Λ, see Algorithm 3). The parameters $\mu \in \{1, 10, 100\}$ and $\tau \in \{1, 10, 100, 1000\}$ as been selected by model selection. For each technique a ranking over the examples in the test set is obtained (a function from the test set to $\mathbb{R}$). AUC metric is used to measure the performance of such a ranking function. We evaluated AUC metric for each data set with different techniques and we have obtained the results in Table 10.1 (using $T = 20$ and $T = 50$, called respectively $\mathbf{ARBF}_{20}$ and $\mathbf{ARBF}_{50}$) and the convergence curves in Figure 10.1 with the AUC values for each iteration of our algorithm up to $T = 150$.

According to these results, our method obtains the best performance in all the six datasets and significantly improve on the baseline (KOMD) and other state-of-the-art techniques.

[1] `http://www.cse.wustl.edu/~kilian/code/code`

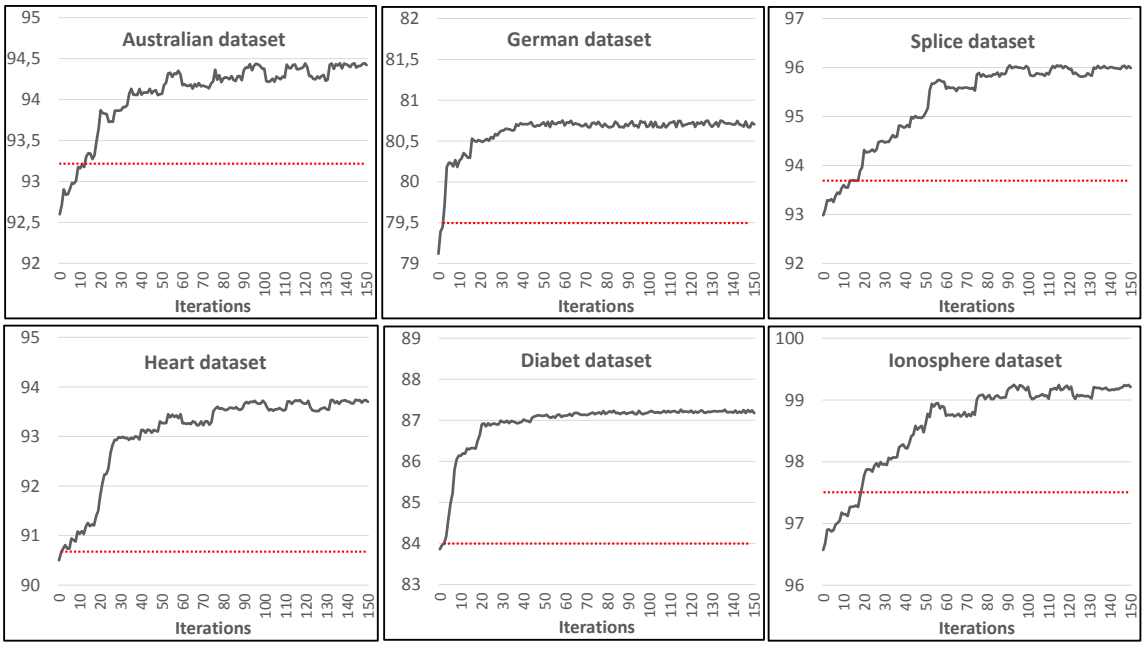


Figure 10.1: AUC % values for each iteration of ARBF compared to the KOMD baseline (red dots).

III Real World Applications

Summary

Dealing with real world problems is the real challenge for the machine learning experts. In this part, three different applications will be presented, where choosing a correct representation is the key in order to obtain an improvement with respect the state-of-the-art solutions.

In Chapter 11, the problem tackled is the segmentation of the Magnetic Resonance Imaging (MRI). MRI allows the acquisition of high-resolution images of the brain. The diagnosis of various brain illnesses is supported by the distinguished analysis of the different kind of brain tissues, which imply their segmentation and classification. Brain MRI is organized in volumes composed by millions of voxels (at least 65.536 per slice, for at least 50 slice), hence the problem of the labeling of the brain tissue classes in the composition of atlases and ground truth references [153], which are needed for the training and the validation of machine-learning methods employed for brain segmentation. We propose a stacking classification scheme that does not require any other anatomical *a priori* information to identify the 3 classes, Gray Matter (GM), White Matter (WM) and Cerebro-Spinal Fluid (CSF). We employed two different MR sequences: FLuid Attenuated Inversion Recovery (FLAIR) and Double Inversion Recovery (DIR). The former highlights both gray matter (GM) and white matter (WM), the latter highlights GM alone. Features are extracted by means of a local multi-scale texture analysis, computed for each pixel of the DIR and FLAIR sequences. The 9 textures considered are average, standard deviation, kurtosis, entropy, contrast, correlation, energy, homogeneity, and skewness, evaluated on a neighborhood of 3x3, 5x5, and 7x7 pixels. Two stacked classifiers were created exploiting the *a priori* knowledge about DIR and FLAIR images. The results highlight an improvement in classification performance with respect to using all the features in a state-of-the-art single classifier. The proper use of *a priori* information was further developed and we noted that the better performance depends on the *a priori* decision to use the two different images in a hierarchical manner with a specific order.

The task tackled in Chapter 12 is concerning a serious game, called *ClimbTheWorld*, which uses a machine learning based technique to recognize and count stairsteps and aims at persuading people to use stairs instead of elevators or escalators. The widespread positioning of smartphones in the users' pockets allows to sense their movements thus monitoring the amount of physical activity they do. But, it also gives the possibility to use these devices to persuade people to change their behaviors. We perform a fine-grained analysis by exploiting smartphone sensors to recognize single stairsteps by exploiting various techniques, i.e.:

- The smoothing and the standardization of the raw signals;
- The segmentation of the data by using a data-dependent window;

- The creation of the features using an *ad hoc* analysis (i.e. feature engineering);
- The standardization of the generated features;
- The feature selections by exploiting a MKL technique.

Finally, energy consumption is widely investigated to avoid exhausting smartphone battery.

Finally, Chapter 13 is concerning Multitask Learning (MTL). Past research on MTL has focused mainly on devising adequate regularizers and less on its scalability. We present a new method to scale up a MTL method based on regularization functionals similar to those used in support vector machines is presented. It builds upon the alternating direction method of multipliers (ADMM) to decouple different tasks in the objective function. This allows for an optimization process in which different tasks are independently solved in parallel and subsequently corrected to pool information from other tasks. This new method converges and outperforms the state-of-the-art methods. In fact, experiments on large URL classification data show that MTL can obtain superior performance over learning a common model or independent task learning both in typical and cold start situations.

Parts of these chapters have been published in [5, 1].

11. MRI Segmentation

11.1 Motivation and importance of the MRI Segmentation

The diagnosis of various brain illnesses, is supported by the distinguished analysis of the White Matter (WM), Gray Matter (GM) and Cerebro-Spinal Fluid (CSF). In this work we present a semi-supervised method to segment WM, GM, and CSF from MRI data that combines DIR and FLAIR scans, without exploiting any anatomical a priori information, and with the specific objective of preserving the lesions belonging to their correct tissue.

There exist widely available and commonly used brain tissue segmentation software, such as the segmentation tool in SPM [11] and FAST in FSL [178], which use both intensity and a priori anatomic information. However, having been designed for general use, they are not necessarily optimized for specific pulse sequences or for application to images from patients with a specific disease. For example, as observed in [144, 73], when used to segment MR images of MS patients, these tools occur in misclassification of MS lesions as GM due to overlapping intensities, which then requires time-consuming manual editing and introduces operator variability into the measurements.

Hence, manual delineation remains the *gold standard* procedure in studies where brain segmentation of MR data sets is required, especially when dealing with specific populations (e.g. [80, 173, 184, 84, 74, 166]). However, it is expert dependent, observer demanding and time consuming, and essentially not transferable. Automated techniques are necessary to overcome these obstacles, especially when large cohorts of data sets are involved [80]. Moreover, given the growing interest in translational studies in neuroscience, the need for building annotated *gold standard* segmentation and atlases on non human data ([138, 41])

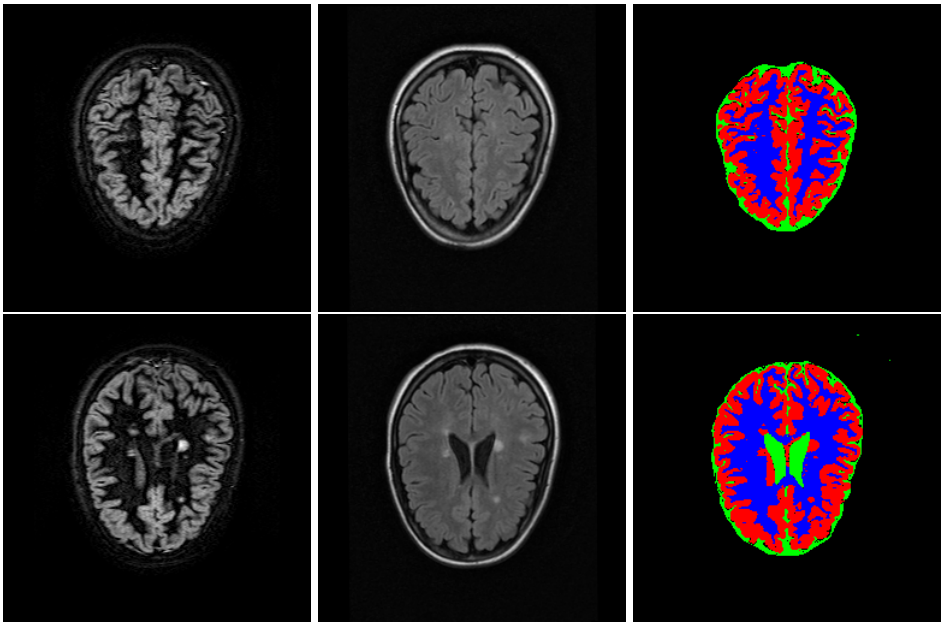


Figure 11.1: (Leftmost column) two slices of the DIR sequence; (central column) the corresponding FLAIR slices; (rightmost column) manual ground truth segmentations provided by the experts

has further stressed the demand for automatic techniques or fast annotation methods.

We propose a supervised classification method that exploits the texture information of the brain tissue provided by the two sequences FLAIR and DIR [152] (11.1). The former is characterized by the suppression of CFS and by the consequent enhancement of both GM and WM, which are however difficult to be distinguished one from the other; the latter has two inversion recoveries that allow suppressing the contribution of both CSF and WM, thus enhancing GM. We thus avoid using T1-w sequence, which, even if characterized by high spatial resolution, proved inadequate for tissue segmentation when brain lesions are present. In addition, the method does not need population-derived location-based priors, registration to template space, or explicit bias field modeling.

11.2 Material

Twenty-four slices (256×256), from $z = 20$ to $z = 44$, from both DIR and FLAIR sequences acquired on a patient affected by MS [191], have been taken into account. In each slide, the three classes GM, WM and CSF were manually labeled.

11.2.1 Feature Extraction

The rationale of the feature extraction approach is to use the peculiar texture characteristic of a pixel neighborhood in order to obtain information about the pixel tissue class [152]. Image texture analysis has been subject of intense study and has been employed in a variety

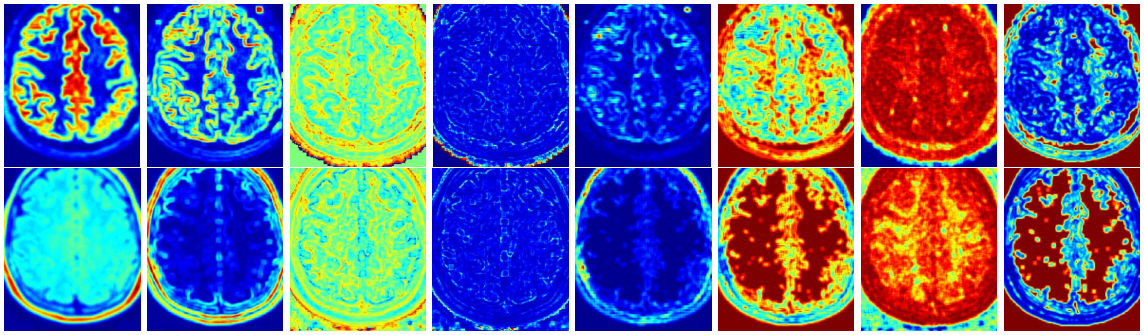


Figure 11.2: *Upper row:* 8 textures extracted from the DIR slice. *Lower row:* 8 textures extracted from the FLAIR slice. *Columns, from left to right:* mean, standard deviation, skewness, kurtosis, contrast, homogeneity, entropy, energy. The scale employed is 5 (5×5 pixel block analysis). Original DIR and FLAIR slices are the one in Fig. 11.1; the slice is at $z = 33$.

of applications; however, there is no general agreement upon definition of texture. For our specific application, we assume that a region in an image has a constant texture if a set of local statistics or other local properties of the picture function are almost constant.

Depending on the number of pixels defining the local feature, the statistical methods can be respectively classified as 1^{st}-order, 2^{nd}-order and higher-order statistics. 1^{st}-order statistics measure the likelihood of observing a specific gray value at a random location in the image (hence directly computable from the image histogram). 2^{nd}-order statistics measure the likelihood of observing a specific pair of gray values in a randomly placed dipole of pixels (computable from the gray level co-occurrence matrices (GLCM) [87]). Method proposed in [193] employed three 1st-order statistics: skewness, median and median absolute deviation, which, on T1-w images, are approximately independent of bias field and of scanner gain. In order to increase the discriminability of the classes, and at the same time couple at best with the double source of information at disposal (i.e., the DIR and the FLAIR sequences), we opted to employ as features four 1^{st}-order statistics and five 2^{nd}-order statistics. The 1^{st}-order statistics considered in this work are mean, standard deviation, skewness, and kurtosis, while the 2^{nd}-order ones are contrast, correlation, homogeneity, entropy and energy.

First-order Statistics

Equations 11.1-11.4 show the mathematical formulations of the 1st-order statistics employed in this work: one ordinary moment and three central moments (two of which are standardized). These measures are computed on an image block $\mathbf{I}$ with dimension $N \times N$,

so the total number of pixels considered is N^2.

$$\mu = \mathbb{E}[\mathbf{I}] = \frac{1}{N^2}\sum_{x,y}\mathbf{I}[x,y] \tag{11.1}$$

$$\sigma = \sqrt{\mathbb{E}[(\mathbf{I}-\mu)^2]} = \sqrt{\frac{1}{N^2}\sum_{x,y}(\mathbf{I}[x,y]-\mu)^2} \tag{11.2}$$

$$\gamma_1 = \mathbb{E}\left[\left(\frac{\mathbf{I}-\mu}{\sigma}\right)^3\right] = \frac{1}{N^2\sigma^3}\sum_{x,y}(\mathbf{I}[x,y]-\mu)^3 \tag{11.3}$$

$$\gamma_2 = \mathbb{E}\left[\left(\frac{\mathbf{I}-\mu}{\sigma}\right)^4\right] - 3 = \frac{1}{N^2\sigma^4}\sum_{x,y}(\mathbf{I}[x,y]-\mu)^4 - 3 \tag{11.4}$$

We chose this set of values because it can accurately characterize a probability distribution in all its properties: while the 1^{st} moment, expected value or mean (Eq. 11.1), describes the central location of the data, central moments relate only to the spread and shape of the distribution. The standard deviation, i.e., the square of the 2^{nd} central moment (Eq. 11.2), measures the dispersion of the data from the average. The skewness (Eq. 11.3) and the kurtosis (Eq. 11.4) measure respectively the asymmetry and the peakedness (or the heaviness of the tails) of the distribution. These two last statistics are based on the standardization (with respect to the standard deviation) of the 3^{rd} and the 4^{th} central moment, and hence are scale invariant and dimensionless.

Second-order Statistics

In order to compute the 2^{nd}-order statistics, GLCM has to be defined (Eq.11.5). If G is the number of gray levels of the image $\mathbf{I}[x,y]$, the GLCM $\mathbf{C}_d$ for a displacement vector $\mathbf{d} = (d_x, d_y)$ is a square matrix $G \times G$ so that the entry $\mathbf{C}_d[i,j]$ is the number of occurrences in $\mathbf{I}$ of the pair of gray levels i and j that are a distance $\mathbf{d}$ apart.

$$\mathbf{C}_d[i,j] = \sum_{x,y}\begin{cases} 1, \textit{ if } \begin{cases} \mathbf{I}[x,y] = i, \\ \mathbf{I}[x+d_x, y+d_y] = j \end{cases} \\ 0, \textit{ otherwise} \end{cases} \tag{11.5}$$

As it can be appreciated in Eq. 11.6-11.9, 2^{nd}-order statistics can be thought as weighted sums of the GLCM cell contents, where the weights express the relative importance of the cell values. In that sense, it is worth noting that the diagonal elements of $\mathbf{C}$ represent all pixel pairs with no grey level difference; the farther away from the diagonal, the greater

the difference between pixel grey levels.

$$Contrast = \sum_{i,j} \mathbf{C}[i,j](i-j)^2 \tag{11.6}$$

$$Homogeneity = \sum_{i,j} \mathbf{C}[i,j]\frac{1}{1+|i-j|} \tag{11.7}$$

$$Entropy = -\sum_{i,j} \mathbf{C}[i,j]\ln C(i-j) \tag{11.8}$$

$$Energy = \sum_{i,j} \mathbf{C}[i,j]^2 \tag{11.9}$$

The contrast (i.e., the sum of squares variance, Eq. 11.6) exponentially weights $\mathbf{C}$ as $|i-j|$ increases, while in the homogeneity definition (Eq. 11.7) the weights decrease exponentially away from the diagonal. Entropy (Eq. 11.8) and energy (Eq. 11.9) are measure related to orderliness (regularity or commonness) of the data: the weights are constructed related to how many times a given pair occurs. Since $\mathbf{C}[i,j]$ are already a measure of commonness of occurrence, it makes that they are used as weights for themselves.

We extract the local texture information at 3 different scales from blocks of $N \times N$ pixel, with N = 3, 5, and 7. For each 2^{nd}-order texture, 4 GLCMs are constructed, with $\mathbf{d} = (d_x, d_y) \in \{(0,1),(1,1),(1,0),(1,-1)\}$. Then, to make the textures invariant to rotation, the obtained matrices are averaged over the 4 angles. Since the feature extraction is performed on both DIR and FLAIR images, the final feature vector associated to each pixel is composed by 56 values (2 original sequence pixel values, plus 9 textures $\times$ 3 scales $\times$ 2 sequences).

11.2.2 Classification

State-of-the-art algorithms typically cast the multi-class problem of classifying CSF, GM and WM with a *one against all* technique. The classifier system consists of three binary classifiers. The classifier h_c is trained with all the available labeled examples giving a positive label to examples of class$c$ and a negative label to examples of other classes. All the features for each image (DIR and FLAIR) are used to build these three classifiers. The prediction is made by comparing the scores of the three classifiers and predicting the class whose corresponding classifier maximizes this score.

However, once considering the *a priori* knowledge we have about our specific problem, thinking in this way can be counter intuitive. In fact, using the approach described above, we are not exploiting the fact that different types of features (DIR and FLAIR) contain diverse information and each one is naturally tailored to a more specific task. FLAIR based features highlight both gray matter (GM) and white matter (WM), then they can be

Algorithm	Accuracy
One-Against-All (OAA)	$0.688_{\pm 0.005}$
Stacked All Features (SAF)	$0.689_{\pm 0.005}$
Stacked Disjoint Features (SDF)	$\mathbf{0.704_{\pm 0.005}}$

Table 11.1: Accuracy$_{\pm std}$ results for our method, i.e. Stacked Disjoint Features (SDF), compared to the Stacked All Features (SAF) and to the One-Against-All (OAA)

more useful to discriminate CSF and not-CSF (GM or WM). On the other side, DIR based features highlight GM alone and thus can be more useful to discriminate between GM and WM when we already know that a particular voxel is not of class CSF.

Here, we consider a *two-step stacked* system that creates, in a first step, a binary classifier h_{CSF} using the FLAIR image features only to select the subset of voxel corresponding to CSF. In a second step, it creates a binary classifier h_{WM} that selects the WM from the not-CSF part of voxel. The remaining part of voxels are classified as GM.

11.2.3 Experimental Setting

We compared three classification settings using a dataset consisting of ten manually labeled slices. Specifically,

1. One-Against-All (OAA, [7]), where each binary classifier is trained with all features (DIR and FLAIR)
2. Stacked All Features (SAF, [45]), where stacking is performed as described above and all the features are used in both levels
3. Stacked Disjoint Features (SDF), where stacking is performed as described above. FLAIR features used in the first level only and DIR features used in the second level.

KOMD (see Chapter 5) with RBF kernel ($\gamma = 0.01$) has been used for all different settings. On each experiment:

1. We randomly select few (5) labeled examples for each class (CSF, GM and WM).
2. We train all the binary classifiers using these labeled data as training data.
3. We classify all the unlabeled data.

We have repeated the steps above for 1000 times to increase the significance of our experiments and we calculated the average accuracy and standard deviation.

11.3 Results

Results for all the three settings are summarized in Table 11.1.

We can see that our two-step stacking algorithm has a significantly better accuracy than state-of-the-art methods. We have also demonstrated that a significant improvement can be

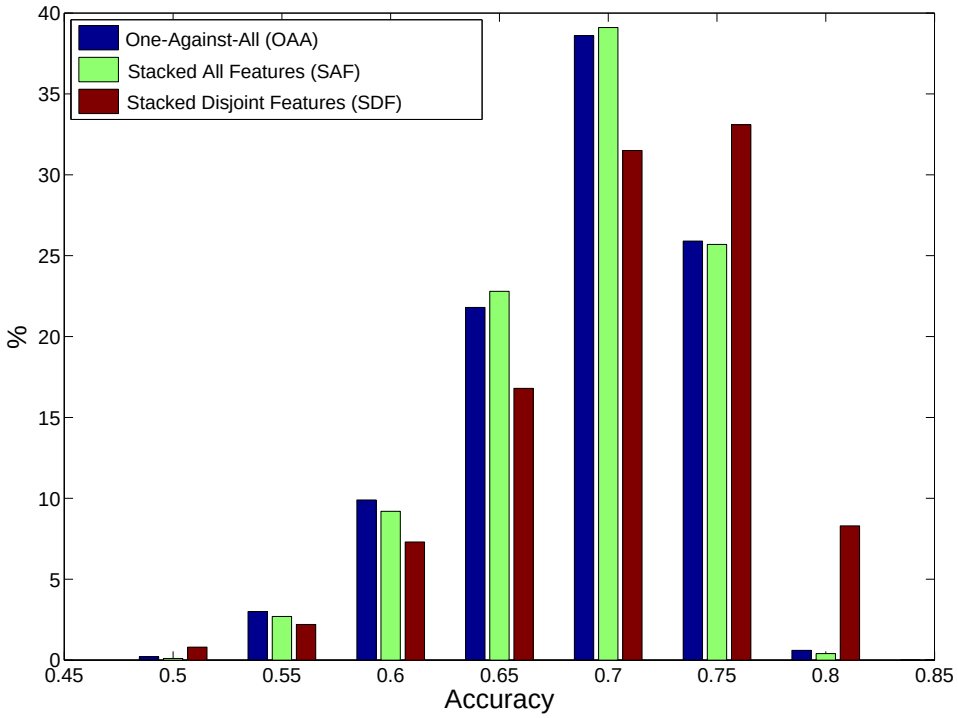


Figure 11.3: Histogram of accuracy for our method, i.e. Stacked Disjoint Features (SDF), compared to the Stacked All Features (SAF) and to the One-Against-All (OAA)

obtained by using *a priori* knowledge on the task at hand. Moreover, the baseline (OAA) method requires the training of three binary classifiers and all the features, while our two-step algorithm needs only two binary classifiers, each one working with a halved number features. The proposed approach provided both better results and better computational performance.

11.4 Discussion and Conclusion

An experiment was performed to prove the importance of a reasoned choice of *a priori* information. We have reversed the hierarchical order of the two classifiers in our two-step algorithm and we obtained a significant decrease in performance. In the future we plan to improve our two-step algorithm in two principal ways: by exploiting the physical topology of the problem and by improving the *active learning* structure of the algorithm. For the first task we can create a graph representation of the brain that contains the topological information of physical neighborhood. From the first experiments we can see an improvement in the results using these topological information on each slice. As far as the latter task is concerned, it is worth mentioning that in the proposed algorithm, an error in the first pixel selection step is propagated directly in the second step. As for

the two-step structure of the algorithm the second classifier starts with a minimum error that is greater than or equal to the error from the first classifier. In order to improve the overall performance, we will study a first-step active learning algorithm that selects the best pixels for the labeling. From the histogram in Figure 11.3, we can see that there are more outstanding performances (with accuracy over 0.8) in our two-step algorithm than the others so we plan to implement an active learning algorithm that chooses in an unsupervised manner the best pixels eligible for the training set. Hopefully, in this way we will increase even further the performance of our two-step algorithm.

12. Climb the world

ClimbTheWorld is a serious game which uses a machine learning based technique to recognize and count stairsteps in real-time. It aims at persuading people to use stairs instead of elevators or escalators. We perform a fine-grained analysis by exploiting smartphone sensors to recognize single stairsteps. Energy consumption is widely investigated to avoid exhausting smartphone battery.

12.1 Motivation

The aim of *ClimbTheWorld* is to persuade people to choose stairs instead of escalators, thus increasing their physical activity, using *gamification*, i.e., providing a way to have fun while climbing stairs. The whole design process of the game followed the Fogg Behaviour

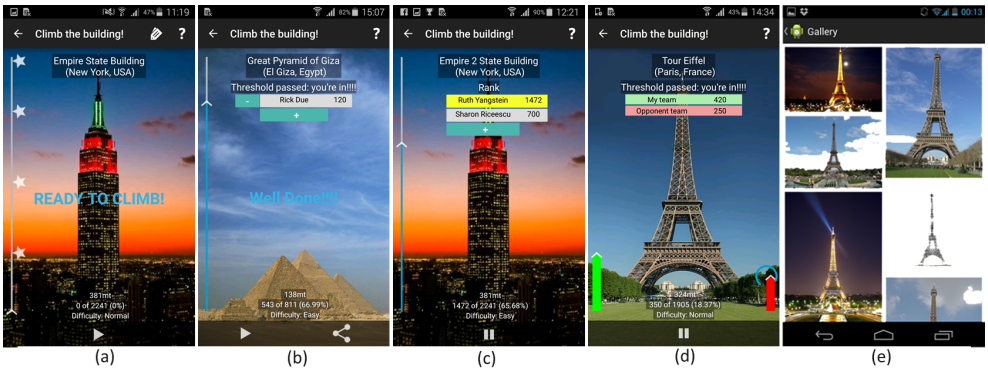


Figure 12.1: Five screenshots of *ClimbTheWorld*

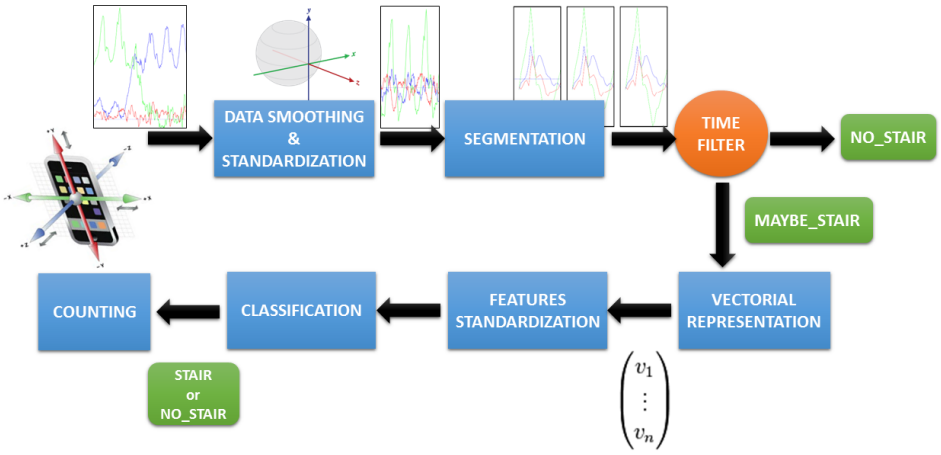


Figure 12.2: Pipeline overview of the system

Model (FBM) [69] to improve game's persuasiveness. FBM shows that three elements must converge at the same moment for a behaviour to occur: *Motivation*, *Ability* and *Triggers*. Since the game is not intended for impaired people, *Ability* is not a problem since our target audience is able to climb the stairs. But it is also true that, even if climbing the stairs can be considered an easy activity, it can become tiring or even frustrating if the goal is too far away. For this reason, we add *sub-goals* to *ClimbTheWorld*, denoted by the stars in Figure 12.1(a), to encourage users to never give up.

In the same way, to avoid discouraging the users, the game proposes different difficulty levels: easier levels correspond to a lower number of stairsteps necessary to reach the top of the building. Each stairstep in real life corresponds to one (or more) stairstep in the game: higher difficulty means less stairsteps in the game. Once the user reaches the top of the building, a slideshow of pictures is displayed, showing the view from the top of the building. Different difficulty levels also bring different quality and number of provided photos. Figure 12.1(e) shows a screenshot of the gallery of the Eiffel Tower.

12.2 Pipeline overview

In this section, we give a broad overview of our system for recognition and counting of stairsteps whose modules will be discussed in detail in the following sections.

The application's pipeline is shown in Figure 12.2. When active, the application constantly acquires data from smartphone sensors to identify whether the user is climbing or descending stairs. Since it is very important to provide real time feedback to increase user engagement, we do not rely on a server that analyzes data and provides classification output, but everything is performed by the smartphone. In this way, we avoid problems of network availability and delay. Moreover, the use of a server can create a bottleneck

in the system. On the other side, if all the computational tasks reside on the smartphone, then energy consumption have to be considered and deeply investigated to avoid wasting energy and drain smartphone battery.

The first step of the pipeline is data acquisition: the smartphone acquires data about movements of the device through its accelerometer and rotation sensor.

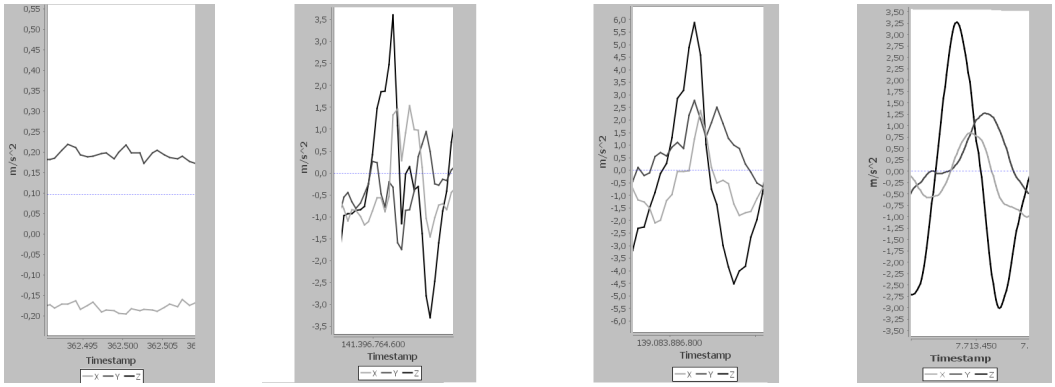
The second step of the pipeline, called "data-smoothing & standardization", initially cleans data by reducing sensor noise and variability. This operation helps to increase the accuracy of the classification. Then data is standardized, since data acquired by sensors can change a lot depending on how the user carries the smartphone as we will discuss in Section 12.4, and we do not impose to the user a smartphone position. To overcome this problem, we translate smoothed data to a fixed coordinate system, using information from the rotation sensor. In this way, the same user activity data have the same pattern independently from the orientation of the smartphone.

The segmentation step splits the standardized data into consecutive segments or windows. In this module, a window segmentation technique based on data analysis (i.e., without fixing size of a window) is used to reduce the computational cost, thus reducing energy consumption, and improve the accuracy. The proposed method allows to consider time as a parameter. In this way, windows that do not fall into a predefined time duration, i.e., the time necessary to complete a stairstep, can be automatically labeled as not a stairstep without forwarding it to the next step (thus, reducing energy consumption).

After segmentation, the extraction and standardization of the features for the classifier takes place.

We cast the stairstep recognition task in a classification setting where we have two possible labels ("STAIR" and "NO_STAIR") and the examples consist of vector representations of segmented windows. Note that, the training of the model is performed offline only once (one single model shared by all different users), while the recognition phase have to be made in *real-time*. This implies that the classification outputs must be promptly provided to the user without delays. We must note here that a simple high-pass filter or a peak detection algorithm would not be sufficient in our context. In fact, as it is shown in Figure 12.3b and in Figure 12.3c, a stairstep and a step have almost the same shape, that is, a data peak on one axis. For this reason, we need a more complex method able to distinguish between this two different activities.

In the following sections, we provide a deep analysis of the main modules compounding the complete system. Finally, we present the results of the classification algorithms in a simulation of the system and an analysis of issues related to energy consumption.



(a) Noise that affects accelerometer data

(b) Example of a step

(c) Example of a stairstep

(d) Example of a stairstep using data smoothing

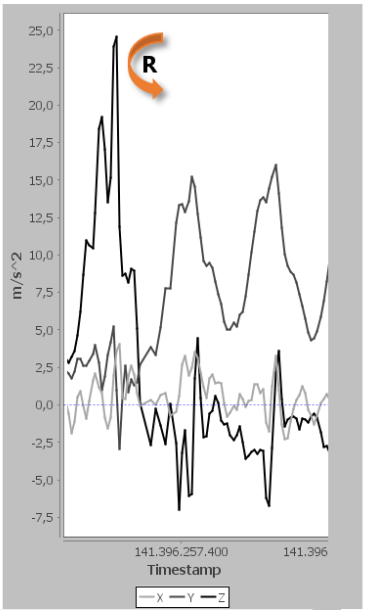
Figure 12.3: Several examples of data acquired from the accelerometer

12.3 Data smoothing

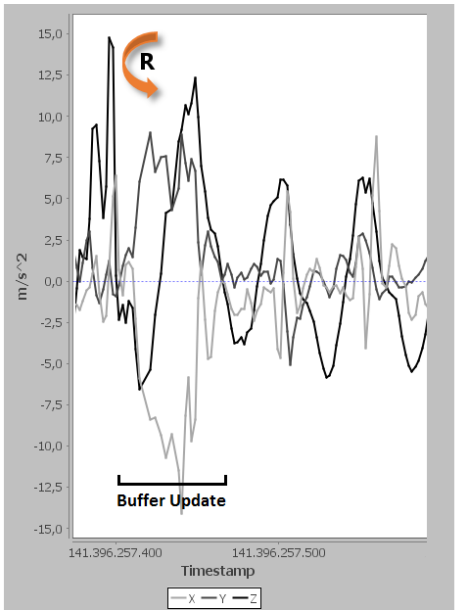
One of the biggest problem that affects data retrieved from the accelerometer and any other sensor of a smartphone acquiring real-time information from the environment is the noise that affects these data. This noise is caused, for example, by sensor precision and sensibility, or by the external environment itself. Let us consider, for example, data from the accelerometer. If we put the smartphone on a table with the screen facing out the sky and we record data from the accelerometer, what we expect to register is a series of vectors equal to $v = (0.0, 0.0, 9.8)$, where X and Y axes should record values equals to 0.0 m/s^2, since there is no movement, and with the gravity force that influences only the Z axis, whose value should be equal to 9.8 m/s^2.

What actually happens is shown in Figure 12.3a. Each accelerometer axis is affected by both background noise and sensor sensibility that make the signal very unstable. The presence of this noise makes the classification task more difficult as data is affected by this background error that disturbs the recording of the real movement that we want to measure. The idea behind the data smoothing step is to reduce the effect of background noise and hence to have cleaner data.

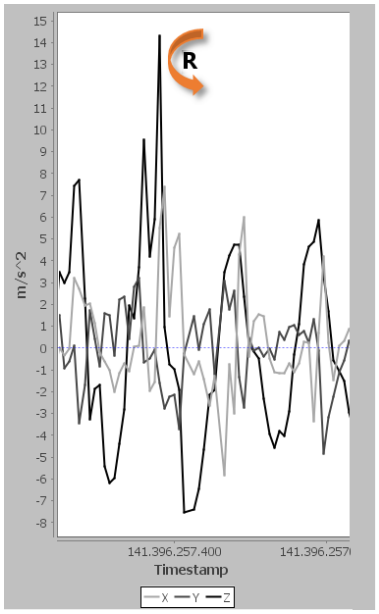
The idea is simple. We use a buffer which stores the accelerometer records related to the last 300ms. The time length of the buffer is short enough to keep all useful information for the classification task (e.g. it is shorter than the time length of a stairstep). Let us suppose that a new vector value $\mathbf{a} = (a_x, a_y, a_z)$ of acceleration data is received from the accelerometer sensor. If the buffer is not full, i.e. its time length is lower than 300ms, then we simply store the vector $\mathbf{a}$ into the buffer and nothing more happens. If the buffer is full, we calculate the average vector $\mathbf{m} = (m_x, m_y, m_z)$ of every readings stored in the buffer and the vector $\mathbf{m}$ is sent forward to the data standardization step. Then, we store the original



(a) Raw data acquired from the accelerometer: same activity but different axis values after rotation of the smartphone



(b) Our method applied to a walking activity while rotating the smartphone



(c) Native linear acceleration plus rotation method

Figure 12.4: The comparison among the raw signal (a) and the signal from the methods for the orientation-independence: our method (b), Linear (c). The instant in which the rotation takes place is denoted by the letter "R". Each line represents data acquired with reference to one axis X, Y and Y

vector **a** into the buffer. In particular, we use a cyclic buffer, so the first element of the buffer is deleted when a new record is added to the buffer. In this way, the background noise is reduced since variations of a single reading is smoothed from all the other readings.

The result of this technique in a stairsteps pattern is shown in Figure 12.3, where it is possible to see the same pattern of a stairstep without (Figure 12.3c) and with (Figure 12.3d) the data smoothing step. As we can see, the signal is much less disturbed and clearer after the data smoothing step, making the learning task, and hence the classification task, easier and more accurate.

12.4 Data standardization

One of the main issue it is necessary to address when working with activity recognition using data from smartphone sensors like the accelerometer, is smartphone orientation. This problem is related to the fact that data from motion sensor changes depending on how the smartphone is carried on by the user. Consider for example a walking activity as depicted in Figure 12.4a. When the smartphone is rotated to a different position from the initial one (the "R" arrow), the data completely changes. In particular, we can observe how accelerometer data completely switches data of the three axis. Consequently, the learning task of the classifier becomes extremely difficult. One of the possible solution to this problem is to force users to keep the smartphone in a particular position. In fact, some earlier works followed this approach, but this is clearly a strong limitation, since imposing a fixed smartphone position reduces pervasiveness of our application and its persuasive power. Another solution would be to train the classifier with every possible orientation of the smartphone. This is clearly an unpractical solution, since there are infinite possible orientations and thus training a classifier of this type becomes infeasible.

One method to solve the problem above was previously proposed by Mizell [141]. The idea is the following: given a sampling interval, the gravity component $\mathbf{g} = (g_x, g_y, g_z) \in \mathbb{R}^3$ on each axis can be estimated by averaging over data read on each axis. When an accelerometer produces the original signal $\mathbf{a} = (a_x, a_y, a_z) \in \mathbb{R}^3$, it is possible to calculate the so called *dynamic component* of **a** as $\mathbf{d} = (a_x - g_x, a_y - g_y, a_z - g_z)$ where the influence of the gravity is eliminated.

Here, we propose a new method to increase the precision of the transformation and preserving the orientation-invariance property. How this method works and how it changes the signal acquired by the accelerometer is shown in Figure 12.4b. The fixed target coordinate system is the following: the X axis is defined as the one tangential to the ground pointing approximately toward East. The Y axis is tangential to the ground pointing toward the geomagnetic North. Finally, the Z axis is orthogonal to the ground plane and

points toward the sky[1]. To implement our method, we need the rotation vector sensor in addition to the accelerometer of the smartphone, and a buffer used to estimate the gravity component acting on each axis.

Firstly, we remove the gravity component from the accelerometer signal using a buffer which stores data acquired during the last 500ms, and averaging over the axes. The vector $\mathbf{g} = (g_x, g_y, g_z)$ is used to compute the dynamic component vector $\mathbf{d} = (a_x - g_x, a_y - g_y, a_z - g_z)$ where $\mathbf{a} = (a_x, a_y, a_z)$ is the original accelerometer reading. Once the vector $\mathbf{d}$ has been computed, it is rotated to the fixed coordinate system using information coming from the rotation vector sensor. This sensor provides information about the current orientation of the device with a three-component vector, where each component represents a rotation angle around an axis. Now, we can apply a three dimensional rotation to the vector $\mathbf{d} = (d_x, d_y, d_z)$ to obtain a new vector $\mathbf{d}' = (d'_x, d'_y, d'_z)$ representing the real movement of the user with respect to our target coordinate system. As we can see in Figure 12.4b, the signal remains affected from the rotation for a very short interval of time only which is bounded by the buffer size.

Android natively supports the automatic gravity removal using the linear acceleration sensor that directly retrieves the dynamic component vector $\mathbf{d}$. Starting from this vector and using the rotation data as described above, linear acceleration readings can be rotated to our fixed coordinate system. An illustration of the rotation-sensor based method applied on the Android native solution is provided in Figure 12.4c. As we will see in Section 12.7.1 and in Section 12.7.2, our method performs better that the native solution both in terms of precision and energy consumption.

A positive aspect of our approach is that, using a buffer to calculate and remove gravity, data from the accelerometer can be represented as a smoothed line which permits us to avoid the problem of sensitivity and calibration of different smartphone sensors, which varies on different models. With our approach this error is smoothed. We made several tests with a Samsung Galaxy i9250, acquiring data from the accelerometer and then applying our approach, and from the linear sensor. The smartphone was placed with the screen pointing to the sky. Data acquired at $125Hz$ for 5 minutes, and the readings averaged on each axis. Since the smartphone was not moving, the theoretical result is $A_t = (0.0, 0.0, 0.0)$. As mentioned before, this is only a theoretical result since every sensor has its own noise. The final average values obtained were $A_l = (-0.04869, 0.01913, 0.62103)$ using the linear sensor and $A_o = (0.03083, -0.01564, 0.42016)$ using our method. As we can see, even if we were not able to completely remove the noise from the accelerometer sensor, its influence on the final data is reduced.

[1] `http://developer.android.com/guide/topics/sensors/sensors_overview.html`

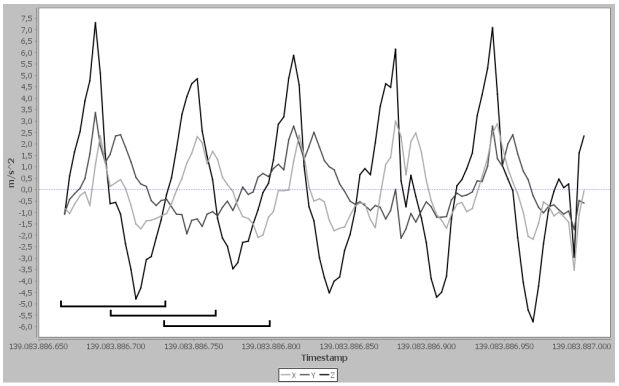


Figure 12.5: Sliding windows using time

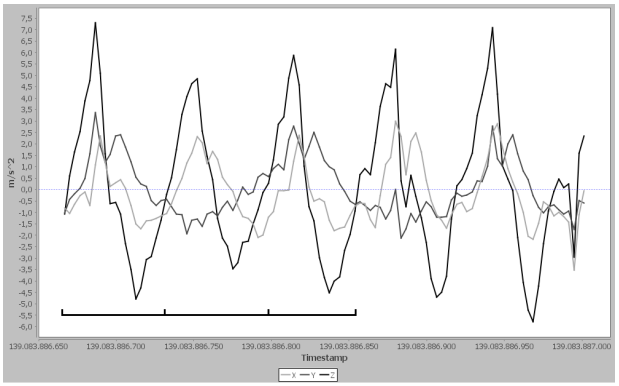


Figure 12.6: Data-dependent sliding windows

Figure 12.7: The comparison between the segmentation in windows of possible stairsteps obtained from the time-driven sliding windows (a) and the data-dependent sliding windows (b)

12.5 Segmentation with data-dependent window

Once data is standardized to a fixed coordinate system, then the data segmentation step is used to divide data into time segments, i.e. *windows*, from which we capture the features that the algorithm uses to understand if the user is climbing stairs or not.

The standard approach for data segmentation is based on *sliding windows*, that is a set of readings of a fixed interval of time, usually a time length sufficiently long to contain the activity one wants to recognize. Since the activity could start in any moment inside a *sliding window*, an improvement can be obtained by using the *overlapping sliding windows* in order to increase the recognition probability. In this case, even if time duration remains the same, the *sliding windows* overlaps, typically on about 50% of the total time duration. Figure 12.5 gives an example of a segmentation using sliding windows which overlaps on 50% of the time length.

The approaches described above have several drawbacks. First of all, since time duration is fixed, there is no user adaptation, especially for short activities, e. g., the time

necessary to perform a stairstep may vary between a child and an old man. Another issue is that, since data pattern inside a window is not considered, windows are always considered for classification, even when it is clear that they are not the target activity. Therefore, this approaches waste energy, since battery is consumed for activities and analysis which are not strictly needed. Moreover, the usage of overlapping sliding windows requires to consider the same data twice (for two different windows).

Our approach focuses on data patterns (and not on time) in order to decide the starting point and the ending point of a sliding window. Since the data pattern of a stairstep is generally the same, i.e., a local maximum followed by a negative local minimum for the Z-axis, while the X-axis and the Y-axis get a much smaller variation, we suggest to use this pattern to divide data coming from sensors into windows. We must note here that, thanks to the previous orientation-independence step, this pattern is not affected by smartphone orientation. An example of how windows are built is provided in Figure 12.6.

Since time length of the windows is no more fixed but becomes variable, it can be used for energy consumption purposes. Analyzing the training set with stairsteps, we found out that, on average, people need a time span between 300ms and 2 seconds to complete a stairstep. Using this information, all the windows that do not fall into this time length interval can be automatically discarded and considered as not stairsteps, even when they follow the data pattern. In this case, we can omit the real classification step of our pipeline, hence saving energy. Finally, our approach adapts a classification task to the user variability, since the window are appropriately re-sized to contain a stairstep without the need of a previous step of calibration. For a deeper analysis of our approach see [1].

12.6 Representation and features standardization

In the representation phase, the dynamic window resulting from the segmentation phase is transformed into a vector of real values representing the information obtained by the device sensors contained in that particular time window. This mapping will permit a natural application of standard machine learning methods to our task of recognizing whether the user is climbing stairs or not. The way data are represented is crucial for the effectiveness of a learning algorithm. Specifically, a good representation method will be able to maintain the information which is really relevant for the task and reduce the noise in the data as much as possible.

In the following we formally describe our choices for the representation module. In particular, let the frequency of sampling be fixed. Then, each window obtained by the segmentation step will consist of a sequence of n vectors, $\mathbf{v}_i = (x_i, y_i, z_i) \in \mathbb{R}^3$, $i \in \{1, .., n\}$, each one consisting of the standardized values of acceleration with respect to the three axes.

We also consider two additional computed values that, in our opinion, can carry precious information. Firstly, we consider the norm of the vector $\mathbf{v}_i$ which well represents the global amount of activity being performed and, secondly, the average value of horizontal accelerations, namely $\frac{x_i+y_i}{2}$ which combines the horizontal activities in our fixed coordinate system in a single value.

More formally, for any single standardized vector $\mathbf{v}_i$ of a dynamic window, a new vector $\mathbf{s}_i \in \mathbb{R}^5$ can be built as follows:

$$\mathbf{v}_i = (x_i, y_i, z_i) \mapsto \mathbf{s}_i = (x_i, y_i, z_i, \|\mathbf{v}_i\|_2, \frac{x_i + y_i}{2})$$

Now, the entire sequence of observations in a window can be represented in the matrix $\mathbf{S} \in \mathbb{R}^{5\times n}$ where the vectors $\mathbf{s}_i$ are accommodated in columns. Finally, the representation of the sequence corresponding to the entire dynamic window is obtained by applying a simple transformation $\Phi : \mathbb{R}^{5\times n} \to \mathbb{R}^{79}$, by evaluation of different statistical estimates over the rows of $\mathbf{S}$.

Specifically, the 79 dimensions of $\Phi(\mathbf{S})$ are created evaluating standard statistics (i.e. average, standard deviation, variance and difference between minimal and maximal values) which are very common in time series analysis or evaluating the ratio among the statistical features above and the correlations from different sources [175, 151].

Table 12.1 presents a detailed description of the entire set of features we decided to compute in order to represent a sequence of acceleration measures in a dynamic window. The first set of 20 features considers standard statistics computed over the rows $\mathbf{S}_j$ of the sequence matrix $\mathbf{S}$. The next group of 40 features considers the ratio of the same statistics computed on different rows. An additional group of 9 features (61-69) takes in account other important ratio statistics (among vertical and horizontal activities and the norm of the total activity), and two other features, the Magnitude Area (MA) and the Signal Magnitude Area (SMA) of $\{\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3\}$. Finally, the last 10 features correspond to the value of the correlation between pairs of rows. For reason of computational complexity and energy consumption we did not use the features in the Fast Fourier Transform (FFT) family.

In order to fairly compare our orientation-independence supporting method with other methods presented earlier in this chapter, we used a similar representation mapping. In particular, exactly the same mapping is used for the Linear method. Concerning the Mizell method, that returns two measurements vectors instead of one, we have considered the natural extension of the function Φ applied to the two vectors independently and concatenating the resulting vectors, thus obtaining a new vector of dimension $79 \times 2 = 158$.

Finally, it is well-known that the classification algorithms are badly influenced by features with different orders of magnitude. For this reason, we have rescaled all the features from the dynamic windows used to train the classifiers between $[-1, 1]$ with a

Features No.	Description
$1-5$	$Ave(\mathbf{S}_j),\ j \in \{1,\ldots,5\}$
$6-10$	$Std(\mathbf{S}_j),\ j \in \{1,\ldots,5\}$
$11-15$	$Var(\mathbf{S}_j),\ j \in \{1,\ldots,5\}$
$16-20$	$Max(\mathbf{S}_j) - Min(\mathbf{S}_j),\ j \in \{1,\ldots,5\}$
$21-30$	$\frac{Ave(\mathbf{S}_j)}{Ave(\mathbf{S}_h)},\ j,h \in \{1,\ldots,5\}, j > h$
$31-40$	$\frac{Std(\mathbf{S}_j)}{Std(\mathbf{S}_h)},\ j,h \in \{1,\ldots,5\}, j > h$
$41-50$	$\frac{Var(\mathbf{S}_j)}{Var(\mathbf{S}_h)},\ j,h \in \{1,\ldots,5\}, j > h$
$51-60$	$\frac{Max(\mathbf{S}_j)-Min(\mathbf{S}_j)}{Max(\mathbf{S}_h)-Min(\mathbf{S}_h)},\ j,h \in \{1,\ldots,5\}, j > h$
61	$\frac{Max(\mathbf{S}_3)}{Max(\mathbf{S}_5)}$
62	$\lvert\frac{Min(\mathbf{S}_3)}{Min(\mathbf{S}_5)}\rvert$
63	$\sqrt{Ave(\mathbf{S}_1)^2 + Ave(\mathbf{S}_2)^2 + Ave(\mathbf{S}_3)^2}$
64	$Ave(\lvert\mathbf{S}_1\rvert + \lvert\mathbf{S}_2\rvert + \lvert\mathbf{S}_3\rvert)$
65	$\frac{Std(s_1)^2}{Std(s_4)}$
66	$\frac{Std(s_2)^2}{Std(s_4)}$
67	$\frac{Std(s_3)^2}{Std(s_4)}$
68	$\frac{(Std(s_1)+Std(s_2))^2}{Std(s_3)}$
69	$\frac{(Std(s_1)+Std(s_2))^2}{Std(s_4)}$
$70-79$	$Corr(\mathbf{S}_j,\mathbf{S}_h),\ j,h \in \{1,\ldots,5\}, j > h$

Table 12.1: The 79 features extracted from dynamic windows

linear transformation, in order to improve the classification results from the classifiers. The linear transformations (that are fixed and different for each feature) are applied to all the features before the classification takes place in order to improve the classification performance.

12.7 Experiments and results

In this section, we present experimental results we have obtained considering classification performances, energy consumption and users evaluation.

12.7.1 Classification setting and results

Our dataset was acquired through direct data collection from 6 different users using different smartphones. This data is recorded keeping the devices consistent with the body movement[2] (e.g. hand held, in a backpack or in a handbag) and without other constraints

[2]In this chapter, we consider a device movement consistent with the body movement whenever the device movement is negligible relatively to the body's barycenter

Frequency 20Hz												
	Mizell rotation			**Linear rotation**			**Our rotation method**			**Smoothed + Our rotation method**		
	Prec	Rec	$F_{1\pm std}$	Prec	Rec	$F_{1\pm std}$	Prec	Rec	$F_{1\pm std}$	Prec	Rec	$F_{1\pm std}$
DT	0.81	0.84	$0.825_{\pm 0.012}$	0.67	0.70	$0.685_{\pm 0.024}$	0.77	0.79	$0.780_{\pm 0.015}$	0.73	0.71	$0.720_{\pm 0.010}$
KNN	0.80	0.77	$0.785_{\pm 0.010}$	0.87	0.77	$0.817_{\pm 0.024}$	0.80	0.75	$0.774_{\pm 0.012}$	0.81	0.81	$0.810_{\pm 0.009}$
KOMD	0.83	0.84	$0.835_{\pm 0.011}$	0.83	0.83	$0.830_{\pm 0.018}$	0.84	0.85	$0.845_{\pm 0.008}$	**0.84**	**0.90**	$\mathbf{0.869_{\pm 0.007}}$

Frequency 30Hz												
	Mizell rotation			**Linear rotation**			**Our rotation method**			**Smoothed + Our rotation method**		
	Prec	Rec	$F_{1\pm std}$	Prec	Rec	$F_{1\pm std}$	Prec	Rec	$F_{1\pm std}$	Prec	Rec	$F_{1\pm std}$
DT	0.82	0.86	$0.840_{\pm 0.012}$	0.80	0.78	$0.790_{\pm 0.024}$	0.82	0.73	$0.772_{\pm 0.015}$	0.73	0.71	$0.720_{\pm 0.010}$
KNN	0.83	0.84	$0.835_{\pm 0.010}$	0.88	0.61	$0.721_{\pm 0.024}$	0.90	0.78	$0.836_{\pm 0.011}$	0.87	0.85	$0.860_{\pm 0.009}$
KOMD	0.89	0.85	$0.870_{\pm 0.010}$	0.86	0.83	$0.845_{\pm 0.018}$	0.90	0.88	$0.890_{\pm 0.008}$	**0.91**	**0.91**	$\mathbf{0.910_{\pm 0.007}}$

Frequency 50Hz												
	Mizell rotation			**Linear rotation**			**Our rotation method**			**Smoothed + Our rotation method**		
	Prec	Rec	$F_{1\pm std}$	Prec	Rec	$F_{1\pm std}$	Prec	Rec	$F_{1\pm std}$	Prec	Rec	$F_{1\pm std}$
DT	0.70	0.77	$0.733_{\pm 0.012}$	0.82	0.89	$0.854_{\pm 0.023}$	0.81	0.75	$0.779_{\pm 0.015}$	0.86	0.83	$0.845_{\pm 0.009}$
KNN	0.82	0.85	$0.835_{\pm 0.009}$	0.81	0.84	$0.825_{\pm 0.024}$	0.90	0.78	$0.836_{\pm 0.011}$	0.82	0.85	$0.835_{\pm 0.009}$
KOMD	0.85	0.88	$0.865_{\pm 0.010}$	0.79	0.86	$0.824_{\pm 0.018}$	0.91	0.90	$0.905_{\pm 0.008}$	**0.92**	**0.92**	$\mathbf{0.920_{\pm 0.006}}$

Table 12.2: Experimental results of Precision (Prec), Recall (Rec) and $F_{1\pm std}$ score for the algorithms and methods used with frequency of sampling of **20**Hz, **30**Hz and **50**Hz. The result highlighted has the largest value of $\mathbf{F_1}$

with respect to where or how to keep the device when the data was collected. All the methods presented in Section 12.6 have been used to compute a vector of features for each dynamic window contained in the whole dataset. Data has been then manually labeled in order to use it with the supervised learning algorithms.

We use algorithms from three different families to tackle this task:

- *Decision Trees (DT)* generated using the C4.5 algorithm,
- *K-nearest neighbors (KNN)* and
- *Kernel Optimization of the Margin Distribution (KOMD)* that is described in Chapter 5, with RBF as the kernel.

For the KOMD algorithm, we used our own implementation, while for the decision trees and the k-nearest neighbors we used the Weka [85] implementation.

Raw data was sampled with at three different frequencies, namely 50Hz, 30Hz and 20Hz, with the aim to explore the relation between performance and energy consumption and finding the best trade-off. Higher frequencies correspond to more accurate information given to the system, but obviously, they also correspond to an higher energy consumption for the device.

In our application, we face with the binary classification task of stairstep recognition. The distribution of different labels in this case is very imbalanced since we have far more negative examples than positive ones. For this, we evaluated *Precision* and *Recall*, instead of accuracy, to obtain a more proper performance estimation and comparison of different experimental settings. Then, we used the F_β*-score* (with $\beta = 1.0$) to combine recall and precision in a single aggregated effectiveness score.

Starting from the original datasets with manually assigned labels for each dynamic window, the experiments are performed using a nested stratified 10-fold cross validation.

Firstly, the dataset has been divided in 10 partitions (each partition with the same distribution of stairsteps). The best parameters are selected for each algorithm using the classical 10-fold cross validation among the windows contained in 9 out of the 10 partitions (i.e. the training set). Finally, the machine learning algorithms are trained to generate a model using the best parameters and the classification statistics are evaluated among the windows contained in the remaining partition, used as test set. This procedure is repeated selecting all the 10 partitions as test set and evaluating the average of the performance. The obtained results over the test set are summarized in Table 12.2 for different rotation methods: Linear, Mizell, our method without the smoothing of the data and our method with the smoothing of the data.

These results show that smoothing of data improves our proposed method for the support to the orientation-independence. This technique, when combined with the KOMD classification algorithm, obtains the best performances against any other combination of methods and algorithms. The results are satisfactory considering the high difficulty of the classification task and given the small training sample obtained on each single dynamic window. We are also interested in finding the best trade-off between energy consumption and performances. On this respect, we can see that the best compromise is obtained at 30Hz, as we can notice that the difference in performances between 50Hz and 30Hz is not statistically significant considering the standard deviation. Moreover, our method with KOMD shows an higher F_1 score at 30Hz than any other classifier at 50Hz.

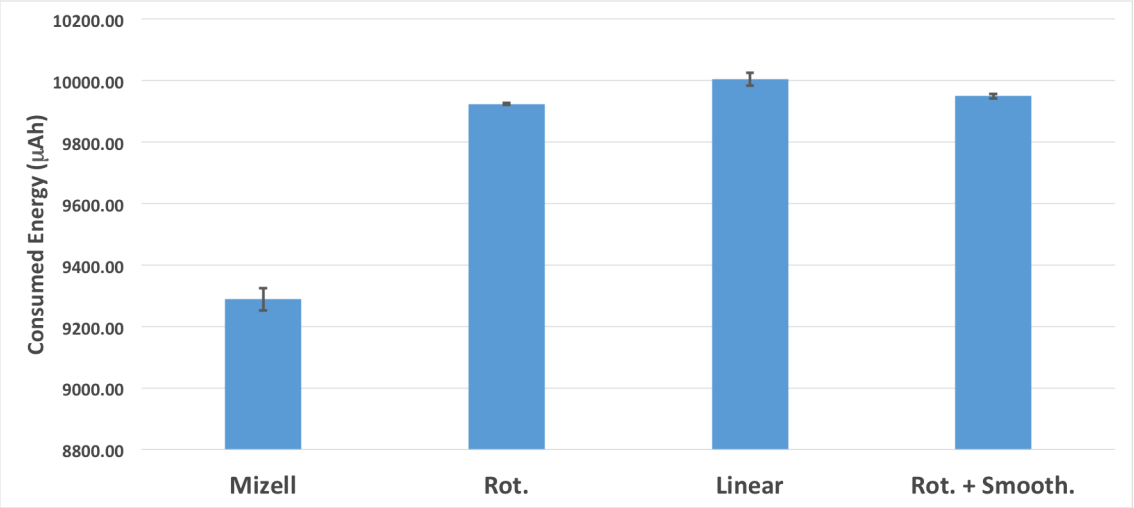
12.7.2 Energy consumption results

When dealing with smartphone devices and applications, energy consumption is one of the key aspects to consider. One of the most important aspect for mobile devices users is battery lifetime, meaning that it is not possible to develop an application that wastes a lot of energy and asks the user to recharge smartphone more than once a day. This kind of situation usually takes the user to remove the application.

A deep analysis of each step of the pipeline is a fundamental requirement during design and development. In our particular case, we wanted to study how resources are used and which are the computationally expensive tasks. Furthermore, we wanted to understand how data frequency acquisition from the accelerometer and the rotation sensor affects energy consumption and in which measure, how much energy the data-smoothing step requires and how well the data segmentation technique performs with respect to the *sliding window* technique.

To measure the energy consumption, we used the Monsoon Power Monitor[3]. Differently form other approaches like background application running on the smartphone

[3] `http://www.msoon.com/LabEquipment/PowerMonitor/`



Energy consumption (μAh)			
Mizell	**Linear**	**Rotation method**	**Smoothing and rotation method**
$9289_{\pm 36}$	$10003_{\pm 21}$	$9923_{\pm 3}$	$9949_{\pm 8}$

Figure 12.8: Energy consumption of the four different methods

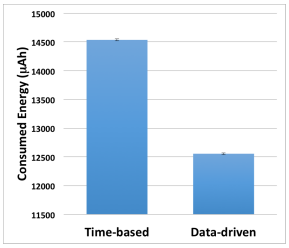
(see [140, 147]) that could introduce an unknown and not fixed overhead that could affect measured data, the Monsoon Power Monitor is an external device that directly measures the "requested" energy from the smartphone to the battery. In this way, no overhead is introduced.

Our tests have been performed using a Samsung Galaxy i9250 with a 1750mAh battery. Even if this smartphone is not up to date, we considered this not a real problem since we do not want to consider absolute values of energy consumption, but we want to compare energy consumption of different methods to determine which is the less expensive.

Figure 12.8 reports the energy consumption measurements. Each experiment lasted two minutes and we repeated each test three times for each method. We tested our method using the selected frequency of 30Hz.

The first evidence we can note from our results is that the Mizell method is the less consuming one. This is not surprising since this method gets data from the accelerometer sensor only, thus requiring less energy. On the other side, this method shows a significantly poorer performance (see Table 12.2).

Comparing our method, which uses both the rotation sensor and the accelerometer, and the linear method, which use the rotation sensor and the linear sensor, we can note that the usage of the accelerometer is less demanding than the usage of the linear sensor. These results along with the ones presented in Table 12.2 demonstrate that our solution is able to improve both the accuracy in terms of stairstep recognition and the energy consumption with respect to the linear solution. Moreover, when adding the data smoothing phase, the



	Segmentation approach	
	Time-based window	**Data-driven window**
Energy consumption (μAh)	$14535_{\pm 17}$	$12554_{\pm 14}$

Figure 12.9: Energy consumption comparison between time-based and data-driven windows

performance in accuracy are further improved without affecting the energy consumption (only about +0,26%).

Another key aspect of our pipeline is the use of data-driven windows that, differently from standard time-based windows, uses the data pattern to determine how to segment data coming from sensors. To compare the two methods in terms of energy consumption, we tested both of them acquiring data at 30Hz and using KOMD as the classification algorithm. For time-based windows, a time length of 500ms and overlapping at 50% were used. Final results are reported in Figure 12.9.

As we can see from our data, data-driven windows allow us to save more energy (more than 15%) with respect to the time-based sliding windows. This is because in the time-based approach every windows have to be classified, while, in our method, only the ones that contains the correct pattern are taken into consideration. Moreover, the time filter let us further reduce the number of analyzed windows, and cannot be applied to standard windows. Clearly, less windows analyzed by the classifier means less energy consumption.

12.7.3 Irrelevant features

A further analysis has been performed to investigate on which features are more useful for our specific task. The motivation for this is three-fold. On one hand we wanted to understand what kind of information is sufficient to recognize stairsteps. On the other hand, removing useless features allows a classifier to learn more quickly, that is, it needs fewer examples to obtain the same performance in classification [145]. Finally, having fewer features to compute can be beneficial from the point of view of energy consumption.

We applied the MKL approach for feature selection, using EasyMKL (see Chapter 7). Pursuing the connection between feature selection and MKL, the *weak* kernels in our case are obtained from single features.

Specifically, we have generated a family of 79 *weak* linear $1-$rank kernels $\mathcal{F} = \{\mathbf{K}_1, \ldots, \mathbf{K}_{79}\}$ by using the 79 features (i.e. $\mathbf{K}_i = \mathbf{X}[:,i]^T\mathbf{X}[:,i]$). Then, we applied EasyMKL to find the optimal combination of these kernels. The weight μ_i, and hence the importance, of the feature i^{th} is given by the value obtained in the linear combination

returned by EasyMKL. From this point of view, given in input the whole dataset, EasyMKL returns an order of the features (from the best to the worst) for our specific classification task.

Our experiments have shown that we can achieve a performance that is very similar to the one obtained using the whole set of 79 features (F_1 score of $0.905_{\pm 0.009}$) by keeping only the first 49 most important features. Specifically, the following features: 69, 31, 13, 48, 49, 17, 47, 24, 76, 1, 27, 73, 7, 64, 35, 39, 47, 72, 53, 67, 59, 66, 55, 33, 45, 25, 37, 23, 21 and 57 have been discarded (the indexes refer to the Table 12.1 and they are sorted in order of importance).

In Figure 12.10, we depicted the weight assigned to each feature by EasyMKL, sorted in order of importance. The distribution of the weights highlights that a set of the evaluated features are useless for this task (i.e. their corresponding weights are zero). On the other hard, the most important features play a key role in the solution of the classification problem and obtain an high weight.

A second experiment has been performed. We have trained KOMD several times, initially only using the best feature and sequentially adding, at each step, the next feature in the ranking. In Figure 12.11, the F_1 results are presented for this experiment. An anti-correlation is pointed out by analyzing the behavior of the model performance compared to the ranking of the weights, highlighting the efficacy and the correctness of our approach.

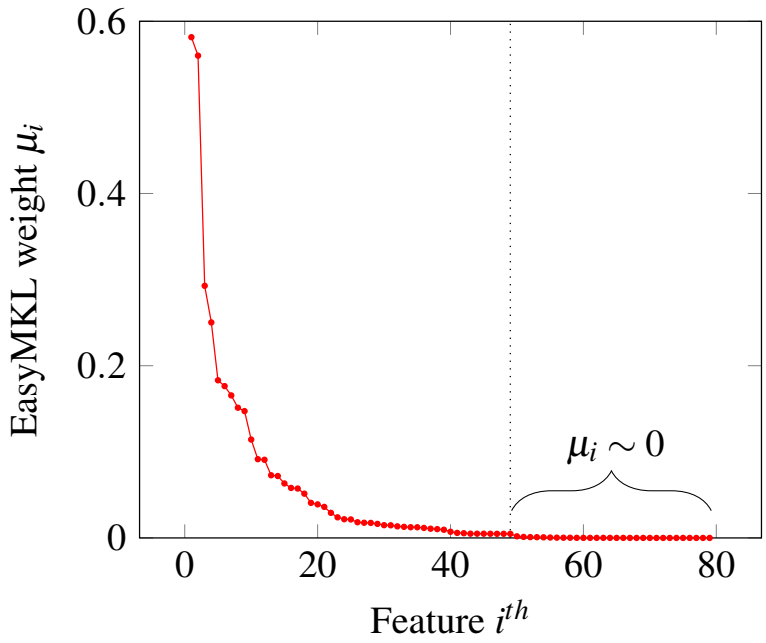


Figure 12.10: The weights μ_i, $i = 1,\ldots,79$ of the linear $1-$rank kernels, given by EasyMKL, sorted in order of importance. The vertical dotted line represents the selected number of considered features (i.e. 49) used to generate the final model of our application

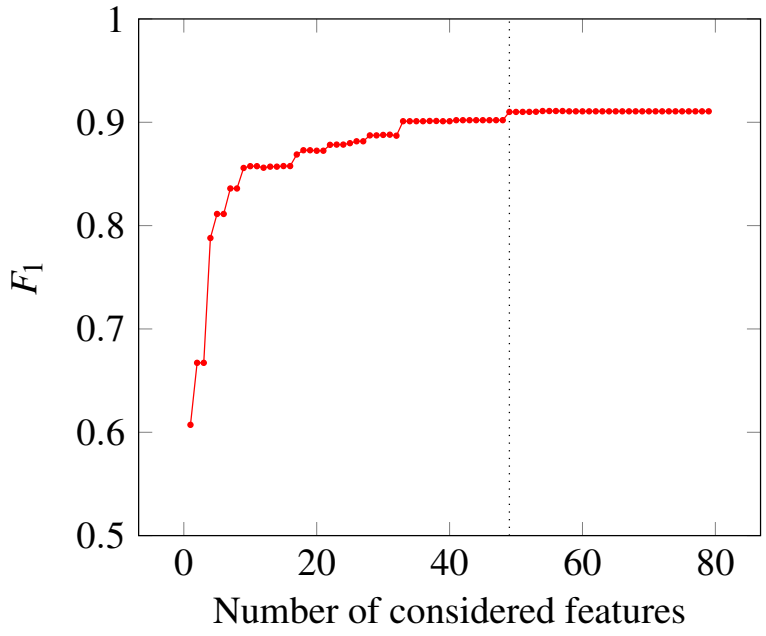


Figure 12.11: F_1 using a different number of features by adding at each step the next best feature in the EasyMKL ranking. The vertical dotted line represents the selected number of considered features (i.e. 49) used to generate the final model of our application

13. Distributed Variance Regularized MTL

A key goal of this chapter is to equip one of the most widespread MTL methods presented in [68] with a parallel optimization procedure. For this purpose, we employ the alternating direction method of multipliers (ADMM), see for example [66, 36] and references therein. We focus on MTL methods which involve the variance of the task weight vectors as the regularizer. We show that the optimization process can be efficiently implemented in a distributed setting, in that the different tasks can first be independently solved in parallel and subsequently corrected to pool information from other tasks. We report on numerical experiments, which indicate that the method works well and converges in few distributed iterations. We empirically observe that the number of iterations is nearly independent of the number of tasks, yielding a computational gain of $O(T)$ over standard solvers. We also present experiments on a large URL classification dataset, which is challenging both in terms of volume of data points, dimensionality and number of different tasks. Our results confirm that MTL can obtain superior performance over either learning a common model or independent task learning.

13.1 The importance of the Multitask Learning

Multitask Learning (MTL) is nowadays an established area of machine learning that has shown its benefits in many applications. MTL aims at simultaneously learning models for multiple related tasks by inducing some knowledge transfer between them. A typical MTL algorithm introduces a regularization term or prior that imposes an adequate shared bias between the learning tasks. This idea was inspired by research on transfer learning

in psychology, which brought up the hypothesis that the abilities acquired while learning one task (e.g. to walk, to recognize cars, etc.) presumably apply when learning a similar task (to run, to recognize trucks, etc.) [176, 185] and was first coined inside the machine learning community in the 90's [42, 185]. The notion of what "task relatedness" means in practice is still not completely clear but theoretical studies support the intuition that training simultaneously from different related tasks is advantageous when compared to single task learning [22] [26] [136]. The benefits usually include more efficient use of small data sets in new related tasks and improved generalization bounds when learning simultaneously.

Recent successful studies on large datasets suggest that the use of models whose training process can be scaled up in terms of the size of the database is key to obtain good results. In large scale learning scenarios, the reduction of the estimation error that can be achieved by leveraging bigger datasets compensates the bias introduced by the use of simpler models such as linear classifiers [35]. This trend has produced several algorithms that put the focus on scaling up the training of simple linear models such as support vector machines to PB of data, see [168] and references therein. Despite this success, the MTL community has targeted the formulation of different regularizers that foster the correct information sharing between tasks in specific situations, overlooking their scalability. Nevertheless, MTL formulations apply very naturally to many large scale scenarios where we expect heterogeneous (although related) regimes to be present in the data.

Scalability of MTL can be challenged from two different angles. Recent applied studies on MTL [31] [61] [19] [100], tackle scenarios such as preference learning, ranking, advertising targeting and content classification. Those studies mainly focus on the scalability when the number of examples increases. But scalability issues can also be encountered when the number of tasks grows. Think for example about learning: i) the preferences of a client on a per client basis, ii) the categorization of webpages on a per host basis or iii) the relevance of a query-document pair for a specific combination of user aspects like region, sex, age, etc. A common feature of these applications is that: (a) many tasks suffer a lack of data to learn from, and (b) the number of free parameters to learn grows with the number of tasks. The first issue has been one of the main motivations of multitask learning, namely to leverage data from similar tasks in order to improve accuracy. On the other hand, the second challenge could impose a limitation on the learning process if the number of tasks grows big since it means a larger amount of data to transfer on the network, stored and managed by the learning algorithms. Thus, if we combine MTL with parallel optimization, we would obtain a practical process that is expected to obtain higher accuracy.

13.2 Multitask Learning Framework

When we refer to multitask learning (MTL) we mean the following situation. We have T learning tasks and we assume that all data from these tasks lie in the same space $\mathbb{R}^d \times Y$, where $Y = \{-1, 1\}$ for binary classification and $Y = \mathbb{R}$ for regression. Associated with each task t we have m_t data points

$$(\mathbf{x}_{1t}, y_{1t}), (\mathbf{x}_{2t}, y_{2t}), \ldots, (\mathbf{x}_{m_t t}, y_{m_t t}) \tag{13.1}$$

sampled from a distribution P_t on $X \times Y$. We assume that this distribution P_t is different for each task but that different P_t are related. The goal is to learn T functions $h_1, h_2, \ldots, h_T$ such that the average error $\frac{1}{T}\sum_{i=1}^{T} \mathbb{E}_{(x,y)\sim P_t}[\ell(y, h_t(x))]$ is low for a prescribed loss function ℓ. Note that when $T = 1$, this framework includes the single task learning problem as a specific case.

In [68] an intuitive formulation for relatedness between tasks inspired by a hierarchical bayesian perspective was presented. This formulation assumes that the hypothesis class of each individual task is formed by the set of linear predictors (classifiers) whose parameter vectors are related by the equations

$$\mathbf{w}_t = \mathbf{v}_t + \mathbf{w}_0, \quad t = 1, \ldots, T. \tag{13.2}$$

The vector $\mathbf{v}_t$ models the bias for task t and the vector $\mathbf{w}_0$ represents a common (mean) classifier between the tasks. Within this setting, the tasks are related when they are similar to each other, in the sense that the vectors $\mathbf{v}_t$ have small norms compared to the norm of the common vector $\mathbf{w}_0$. Note also that this setting may be useful in a transfer learning setting, in which we do not have data for a new task but we can still make predictions using the common average $\mathbf{w}_0$.

In the above setup, an optimal $\mathbf{w}_0$ and a set of optimal $\mathbf{v}_t$ for each task is found by solving the following optimization problem which is an extension of linear SVMs for a single task (which corresponds to the case $T = 1$), namely

$$\min_{\mathbf{w}_0, \mathbf{v}_t} \left\{ \sum_{t=1}^{T} f_t(\mathbf{w}_0 + \mathbf{v}_t) + \frac{\lambda_1}{T} \sum_{t=1}^{T} \|\mathbf{v}_t\|_2^2 + \lambda_2 \|\mathbf{w}_0\|_2^2 \right\} \tag{13.3}$$

where $f_t(\cdot) = \sum_{i=1}^{m_t} \ell(y_{it}, \langle \cdot, \mathbf{x}_{it} \rangle)$, the empirical error for the task t.

Define the variance of the vectors $\mathbf{w}_1, \ldots, \mathbf{w}_T$ as

$$\mathrm{Var}(\mathbf{w}_1, \ldots, \mathbf{w}_T) = \frac{1}{T} \sum_{t=1}^{T} \left\| \mathbf{w}_t - \frac{1}{T} \sum_{s=1}^{T} \mathbf{w}_s \right\|_2^2.$$

It can be shown [68, Lemma 2.2] that problem (13.3) is equivalent to the problem

$$\min_{\mathbf{w}_t} \left\{ \sum_{t=1}^{T} \left(f_t(\mathbf{w}_t) + \rho_1 \|\mathbf{w}_t\|_2^2 \right) + \rho_2 T \ \mathrm{Var}(\mathbf{w}_1, \ldots, \mathbf{w}_T) \right\} \tag{13.4}$$

where the hyperparameters are linked by the equations

$$\rho_1 = \frac{1}{T} \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2}, \quad \rho_2 = \frac{1}{T} \frac{\lambda_1^2}{\lambda_1 + \lambda_2}.$$

This connection makes it apparent that the regularization term encourages a small magnitude of the vectors $\mathbf{w}_t$ (large margin of each SVM) while simultaneously controlling their variance.

13.3 Distributed MTL via ADMM

The utility of problem (13.4) is that its objective function – unlike that in problem (13.3) – is almost separable across the different tasks. Namely, the only term in the objective function in (13.4) that prevents decoupling is the last summand, which makes the gradients for different tasks dependent. If we could remove this dependency, both the data and weight vectors for different tasks could be maintained in different nodes of a computer cluster without the need of centralizing any information and so making the method scalable. For this purpose we use an optimization strategy based on the alternating direction method of multipliers (ADMM), see for example [36, 66] and references therein. This method solve the general convex optimization problem

$$\begin{aligned} \underset{\mathbf{w},\mathbf{z}}{\text{minimize}} \quad & f(\mathbf{w}) + g(\mathbf{z}) \\ \text{subject to} \quad & M\mathbf{w} = \mathbf{z} \end{aligned} \tag{13.5}$$

where f and g are two convex functions. Different algorithms have been proposed to tackle this optimization with different convergence characteristics, see, for example, [36, 143, 90, 95]. In this chapter, we employ the ADMM algorithm outlined in [66]. Although potentially faster algorithms exists, such as [77], their convergence analysis require stronger assumptions which are not meet in our problem.

We define the augmented Langrangian function L_η at $\mathbf{w}, \mathbf{z}, \mathbf{y}$ as

$$L_\eta(\mathbf{w}, \mathbf{z}, \mathbf{y}) = f(\mathbf{w}) + g(\mathbf{z}) + \langle \mathbf{y}, M\mathbf{w} - \mathbf{z} \rangle + \frac{\eta}{2} \|M\mathbf{w} + \mathbf{z}\|_2^2$$

where η is a positive parameter. Each ADMM step requires the following computations

$$\begin{aligned}\mathbf{w}^k &= \arg\min_{\mathbf{x}} L_\eta(\mathbf{w}, \mathbf{z}^{k-1}, \mathbf{y}^{k-1}) \\ \mathbf{z}^k &= \arg\min_{\mathbf{z}} L_\eta(\mathbf{w}^k, \mathbf{z}, \mathbf{y}^{k-1}) \\ \mathbf{y}^k &= \mathbf{y}^{k-1} + \eta(M\mathbf{w}^k - \mathbf{z}^k)\end{aligned} \tag{13.6}$$

where k is a positive integer and $\mathbf{z}^0$, $\mathbf{y}^0$ are some starting points. We call each round of (13.6) an ADMM iteration. This process is repeated until convergence.

A convenient identification between problem (13.4) and the ADMM objective function (13.5) shows that the multitask objective can be efficiently optimized through this strategy. Namely, problem (13.4) is of the form (13.5) for the choice $M = I$ and

$$\begin{aligned}f(\mathbf{w}) &= \sum_{t=1}^{T} f_t(\mathbf{w}_t) + \frac{\rho_1}{2}\sum_{t=1}^{T}\|\mathbf{w}_t\|_2^2 \\ g(\mathbf{z}) &= \rho_2 T \ \mathrm{Var}(\mathbf{z}_1, \ldots, \mathbf{z}_T) + \frac{\rho_1}{2}\|\mathbf{z}\|_2^2\end{aligned}$$

where we set $\mathbf{w}$ to be the concatenation of the weight vectors $\mathbf{w}_t$ for all the tasks, and we force $\mathbf{z} = \mathbf{w}$ in order to make $f(\mathbf{w}) + g(\mathbf{z})$ equal to the original objective in the feasible region.

The augmented Lagrangian for this specific case is

$$\begin{aligned}L_\eta(\mathbf{w}, \mathbf{z}, \mathbf{y}) &= \sum_{t=1}^{T} f_t(\mathbf{w}_t) + \frac{\rho_1}{2}\sum_{t=1}^{T}(\|\mathbf{w}_t\|_2^2 + \|\mathbf{z}_t\|_2^2) \\ &+\rho_2 T \ \mathrm{Var}(\mathbf{z}_1, \ldots, \mathbf{z}_T) + \langle \mathbf{y}, \mathbf{w} - \mathbf{z}\rangle + \frac{\eta}{2}\|\mathbf{w} - \mathbf{z}\|_2^2.\end{aligned}$$

Using this expression, the first three updating equations in the ADMM optimization strategy (13.6) become

$$\begin{aligned}\mathbf{w}^k &= \arg\min_{\mathbf{w}} \left\{ \sum_{t=1}^{T} f_t(\mathbf{w}_t) + \sum_{t=1}^{T}\frac{\rho_1}{2}\|\mathbf{w}_t\|_2^2 + \langle \mathbf{y}_t^{k-1}, \mathbf{w}_t\rangle + \frac{\eta}{2}\|\mathbf{w}_t - \mathbf{z}_t^{k-1}\|_2^2 \right\} \\ \mathbf{z}^k &= \arg\min_{\mathbf{z}} \left\{ \rho_2 T \ \mathrm{Var}(\mathbf{z}_1, \ldots, \mathbf{z}_T) + \frac{\rho_1}{2}\|\mathbf{z}\|_2^2 - \langle \mathbf{y}^{k-1}, \mathbf{z}\rangle + \frac{\eta}{2}\|\mathbf{w}^{k-1} - \mathbf{z}\|_2^2 \right\} \\ \mathbf{y}^k &= \mathbf{y}^{k-1} + \eta(\mathbf{w}^k - \mathbf{z}^k).\end{aligned} \tag{13.7}$$

We are left to analyze how to solve the first two optimization steps to obtain $\mathbf{w}^k$ and $\mathbf{z}^k$. It is noticeable that in the first of these steps the optimization over the tasks is completely decoupled, that is the component vectors $\mathbf{w}_t^k$ can be computed independently of each other – we discuss how to do this in Section 13.5. Thus, the update of each task's weight vector

can be run in parallel with no communication involved once data is distributed by tasks. As we shall see in Section 13.4.2 the second optimization step, in which the only information sharing between the tasks occurs, can also be carried out with minimal communication just by averaging the vectors for the different tasks, hence leading to an scalable strategy.

13.4 Algorithm

In the previous section we showed how the MTL problem in equation (13.3) or (13.4) can be solved by the iterative scheme (13.6). In this section, we analyse the minimization problems for $\mathbf{w}^k$ and $\mathbf{z}^k$, noting that both can be computed in a distributed fashion.

13.4.1 Optimization of each individual task

The update formula for the weights $\mathbf{w}$ in (13.7) can be implemented with different methods. The optimization of the weights completely decouple across the tasks w_t and each can be compute by solving the problem

$$\min_{\mathbf{w}_t} \left\{ f_t(\mathbf{w}_t) + \frac{\rho_1 + \eta}{2} \|\mathbf{w}_t\|_2^2 + \langle \mathbf{y}_t - \eta \mathbf{z}_t, \mathbf{w}_t \rangle \right\}. \tag{13.8}$$

One natural approach is to use (sub)gradient descent, possibly in a stochastic setting when the number of datapoints of a task is large. In this chapter we consider the case that ℓ is the hinge loss, namely $\ell(y, y') = h(yy')$, where $h(\cdot) = \max(0, 1 - \cdot)$. In Section 13.5 we detail a stochastic gradient descent method to solve problem.

13.4.2 Optimization of the auxiliary variables

The update step for the auxiliary variable $\mathbf{z}^k$ is more advantageous computationally since we can work out a closed formula. The objective function is given by

$$\rho_2 \sum_{t=1}^{T} \left\| \mathbf{z}_t - \frac{1}{T} \sum_{s=1}^{T} \mathbf{z}_s \right\|_2^2 + \frac{\rho_1}{2} \|\mathbf{z}\|_2^2 + \langle \mathbf{y}^k, \mathbf{w}^k - \mathbf{z} \rangle + \frac{\eta}{2} \|\mathbf{w}^k - \mathbf{z}\|_2^2$$

for fixed $\mathbf{w}^k$ and $\mathbf{y}^k$. Removing constant terms that do not depend on $\mathbf{z}$, this can further be rewritten as

$$F(\mathbf{z}) = \left\langle \mathbf{z}, \left(\rho_2 D^T D + \frac{\rho_1}{2} I \right) \mathbf{z} \right\rangle - \langle \mathbf{y}^k, \mathbf{z} \rangle + \frac{\eta}{2} \left(\|\mathbf{z}\|_2^2 - 2 \langle \mathbf{w}^k, \mathbf{z} \rangle \right)$$

where matrix $D \in \mathbb{R}^{Td \times Td}$ is given in block form as

$$D = \begin{bmatrix} D_M & D_0 & D_0 & \cdots & D_0 \\ D_0 & D_M & D_0 & \cdots & D_0 \\ D_0 & D_0 & D_M & \cdots & D_0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ D_0 & D_0 & D_0 & \cdots & D_M \end{bmatrix}, \tag{13.9}$$

being d the number dimensions of a block and T the number of blocks. Finally, $D_M = \frac{T-1}{T} I_{d\times d}$ and $D_0 = -\frac{1}{T} I_{d\times d}$, where $I_{d\times d}$ is the $d \times d$ identity matrix.

If we take the derivative of the above objective function we can see that the optimal solution can be found by solving the system of linear equations

$$(2\rho_2 D^T D + (\rho_1 + \eta) I)\mathbf{z}^k = (\mathbf{y}^k + \eta \mathbf{w}^k).$$

The following proposition shows that the inverse $E = (2\rho_2 D^T D + (\rho_1 + \eta) I)^{-1}$ has a convenient closed analytical formula. where E_M and E_0 are the $d \times d$ matrices

Proposition 13.4.1 Let $E = (2\rho_2 D^T D + (\rho_1 + \eta) I)^{-1}$, where matrix $D \in \mathbb{R}^{Td \times Td}$ is defined equation (13.9). The matrix E has the following structure

$$E = \begin{bmatrix} E_M & E_0 & E_0 & \cdots & E_0 \\ E_0 & E_M & E_0 & \cdots & E_0 \\ E_0 & E_0 & E_M & \cdots & E_0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ E_0 & E_0 & E_0 & \cdots & E_M \end{bmatrix} \in \mathbb{R}^{Td \times Td}$$

where E_M and E_0 are:

$$E_M = \frac{1}{T}\left(\frac{1}{\eta + \rho_2} + \frac{T-1}{\eta + 2\rho_1 + \rho_2}\right) I_{d\times d},$$
$$E_0 = \frac{2\rho_1}{T(\eta + \rho_2)(\eta + 2\rho_1 + \rho_2)} I_{d\times d}.$$

Proof. We have to prove that $(2\rho_2 D^T D + (\rho_1 + \eta) I)E = E(2\rho_2 D^T D + (\rho_1 + \eta) I) = I \in \mathbb{R}^{Td \times Td}$. The first observation of the proof is the particular structure of the matrix D that is an idempotent symmetric matrix and then $D^T D = D^2 = D$. So, the matrix $2\rho_2 D^T D + (\rho_1 +$

$\eta)I$ is also equal to $G = 2\rho_2 D + (\rho_1 + \eta)I$ and has the following structure in $\mathbb{R}^{Td \times Td}$

$$\begin{bmatrix} G_M & G_0 & \cdots & G_0 \\ G_0 & G_M & \cdots & G_0 \\ \cdots & \cdots & \cdots & \cdots \\ G_0 & G_0 & \cdots & G_M \end{bmatrix} \tag{13.10}$$

where $G_0 = 2\rho_2 D_0$, $G_M = 2\rho_2 D_M + (\rho_1 + \eta)I_{d\times d}$, where $I_{d\times d}$ is the identity matrix in d dimensions. Then, we can evaluate the block product between E and G. This product generates a block matrix where each block on the diagonal is

$$(T-1)E_0 2\rho_2 D_0 + E_M(2\rho_2 D_M + (\rho_1 + \eta)I_{d\times d}) = I_{d\times d},$$

and the off-diagonal block are zero, since

$$(T-2)E_0 2\rho_2 D_0 + E_0(2\rho_2 D_M + (\rho_1 + \eta)I_{d\times d}) + E_M 2\rho_2 D_0 = 0_d.$$

■

This formula reveals that the optimization over $\mathbf{z}_1, \ldots, \mathbf{z}_T$ can also be run in parallel. First, we can reduce the vectors from all the tasks to a single vector $\mathbf{w} = E_0 \sum_{t=1}^{T} \mathbf{w}_t$. Then, each vector can be updated in parallel by using again E_M and E_0. Similarly to the optimization of $\mathbf{w}_t$, this operation can be readily done in a framework such as MapReduce to speed up computations with a very reduced need of broadcasting information.

13.4.3 Convergence

We comment on the convergence properties of our method. Our observations are a direct consequence of the general analysis in [66]. Specifically, [66, Theorem 8] applies to our problem, with (using their notation) $\rho_k = 1$ for every $k \in \mathbb{N}$, their $\mathbf{p}$ equal to our $\mathbf{y}$ and their λ equal our η. The theorem requires that the matrix M is full rank, f and g are closed proper convex functions and the sum of the errors of the inner optimization problems is finite. All these hypothesis are meet in our case. In particular, the optimization over $\mathbf{z}$ is performed exactly as discussed in Section 13.4.2. The optimization over $\mathbf{w}$ are the parallel SVMs which we can solve to arbitrary precision using for example gradient descent. As we will see in our numerical experiments only few iterations are sufficient to reach a good suboptimal solution and gradient descent may be replaced by its stochastic version (discussed below) without affecting the good convergence of the algorithm.

13.5 SGD Optimization of each individual task

When we have to solve problems with a large number of points, it is not computationally feasible to solve the optimization of each individual task with batch algorithms. In this section, we observe that it is possible to exploit a Stochastic Gradient Descent (SGD) strategy for our proposed method. Firstly, we show that the optimal solution of the optimization problem is contained inside a convex ball. From this results, we are able to satisfy the hypothesis of convergence of the SGD technique for strongly convex functions [156, 168]. Specifically, we prove the boundedness of the directions followed in the SGD optimization of $\mathbf{w}$, in each outer step $k \geq 0$ of the ADMM strategy.

Algorithm 4 depicts the optimization of the weight vectors $\mathbf{w}_t$ using the SGD for strongly convex functions. The optimization for each task presented in Equation 13.8 is equivalent to the problem $\min_{\mathbf{w}_t} F_t(\mathbf{w}_t)$, where

$$F_t(\mathbf{w}_t) = L_t(\mathbf{w}_t) + \frac{\gamma_t}{2}\|\mathbf{w}_t\|_2^2,$$

$$L_t(\mathbf{w}_t) = \frac{1}{m_t}\left(\sum_{i=1}^{m_t} h(y_{it}\langle \mathbf{w}_t, \mathbf{x}_{it}\rangle) + \langle \mathbf{y}_t - \eta \mathbf{z}_t, \mathbf{w}_t\rangle\right)$$

and $\gamma_t = (\rho_1 + \eta)/m_t$. In order to be able to apply SGD we need to sample a direction $\mathbf{u}_t$ at each step q such that $\mathbb{E}[\mathbf{u}_t|\mathbf{w}_t]$ is a subgradient of F_t at $\mathbf{w}_t$. If at each step we sample randomly and with replacement a pattern$\mathbf{x}_{it}$ from the training set, the following sequence of directions complies with the above restriction

$$\mathbf{u}_t = \begin{cases} \gamma_t \mathbf{w}_t + \frac{1}{m}(\mathbf{y}_t - \eta \mathbf{z}_t) & \text{if } y_{it}\langle \mathbf{w}_t, \mathbf{x}_{it}\rangle \geq 1, \\ -y_{it}\mathbf{x}_{it} + \gamma_t \mathbf{w}_t + \frac{1}{m}(\mathbf{y}_t - \eta \mathbf{z}_t) & \text{otherwise.} \end{cases}$$

In the following proposition, we show that the optimal solution of the original optimization problem resides inside a convex ball.

Proposition 13.5.1 If $\mathbf{w}^*$ is the optimal solution of problem (13.4) then $\|\mathbf{w}^*\|_2 \leq \sqrt{\frac{TM}{2\lambda_1}}$, where $M = \sum_{t=1}^{T} m_t$.

Proof. Let $\mu = \frac{T\lambda_2}{\lambda_1}$. We make the change of variables $\mathbf{w} = \left(\sqrt{\mu}\mathbf{w}_0, \mathbf{v_1}, \ldots, \mathbf{v}_T\right) \in \mathbb{R}^{(T+1)d}$ and introduce the map $\phi : \mathbb{R}^d \times \{1, \ldots, T\} \mapsto \mathbb{R}^{(T+1)d}$, defined as

$$\phi(\mathbf{x}, t) = \left(\frac{\mathbf{x}}{\sqrt{\mu}}, 0, \ldots, 0, \mathbf{x}, 0, \ldots, 0\right). \tag{13.11}$$

Following [68] we rewrite problem (13.3) as a standard SVM problem in the extended

input space for the feature map (13.11), namely

$$\min_{\mathbf{w}} \left\{ C \sum_{i=1}^{M} h(y_i \langle \mathbf{w}, \phi(\mathbf{x}_i, t_i) \rangle) + \frac{1}{2} \|\mathbf{w}\|_2^2 \right\}$$

where $C = \frac{T}{2\lambda_1}$ and $M = \sum_{t=1}^{T} m_t$. Since at the optimum the gap between the primal and dual objective vanishes, there exists $\alpha^* \in [0, C]^M$ such that

$$\frac{1}{2}\|\mathbf{w}^*\|_2^2 + C \sum_{i=1}^{M} h(y_i \langle \mathbf{w}, \phi(\mathbf{x}_i, t_i) \rangle) = \sum_{i=1}^{M} \alpha_i^* - \frac{1}{2}\|\mathbf{w}^*\|_2^2.$$

Using the fact that $\sum_{i=1}^{M} h(y_i \langle \mathbf{w}, \phi(\mathbf{x}_i, t_i) \rangle) \geq 0$ we conclude that $\|\mathbf{w}^*\|_2^2 \leq \sum_{i=1}^{M} \alpha_i^* \leq MC = \frac{TM}{2\lambda_1} = \beta$. ■

This proposition thus proves that in Algorithm 4 it is safe to restrict the set of feasible solutions to the ball $\mathbb{B} := \{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\|_2^2 \leq \beta\}$ with $\beta = \max(\sqrt{TM/(2\lambda_1)}, \sqrt{M/(2\lambda_1)})$ and it justifies the projection step at line 7 of the algorithm.

Algorithm 4 Optimization of each individual task

Require: Dataset $\{(\mathbf{x}_{it}, y_{it}), i = 1, \ldots, m_t\}$ for task t, parameters ρ_1, γ_t, η, $\mathbf{z}_t$, $\mathbf{y}_t$, T, M
Ensure: Optimal weight vector $\mathbf{w}_t$

1: $\mathbf{w}_t^0 = \mathbf{0}$
2: $\gamma_t = \frac{\rho_1 + \eta}{m_t}$
3: **for** $q = 1$ **to** n **do**
4: $\quad \delta = \frac{1}{q\gamma_t}$
5: $\quad$ Choose a random pattern $(\mathbf{x}_{it}, y_{it})$ from the set
6: $\quad \mathbf{w}_t^{q+\frac{1}{2}} = \mathbf{w}_t^q - \delta \mathbf{u}_t$
7: $\quad \mathbf{w}_t^{q+1} = \min\left(1, \beta \frac{1}{\|\mathbf{w}_t^{q+\frac{1}{2}}\|_2}\right) \mathbf{w}_t^{q+\frac{1}{2}}$
8: **end for**
9: return $\hat{\mathbf{w}}_t = \frac{2}{n} \sum_{q=\frac{n}{2}+1}^{n} \mathbf{w}_t^q$

Now, we analyze the convergence of Algorithm 1 and we base our proof on a recent result presented in [156, 168] which states (adapting their result to our notation) that the aforementioned strategy converges with the rate $\mathbb{E}[F_t(\hat{\mathbf{w}}_t)] - F_t(\mathbf{w}_t) \leq \frac{\rho^2}{2\gamma_t n}$ after n iterations, provided when $F_t(\mathbf{w})$ is γ_t-strongly convex and $\mathbb{E}[\|\mathbf{u}_t\|^2] \leq \rho^2$ for some finite constant ρ. Clearly, the objective function F_t is strongly convex with constant $\gamma_t = \frac{\rho_1 + \eta}{m_t}$, see [168, Lemma 13.5]. We are left to prove that the directions $\mathbf{u}_t$ followed in the SGD optimization

of $\mathbf{w}$ in each outer step $k \geq 0$ of the ADMM strategy are bounded, which we state in the following theorem.

Theorem 13.5.2 If all the examples $\mathbf{x}_{it}$ satisfy $\|\mathbf{x}_{it}\|_2^2 \leq R$ then for every iteration $k \in \mathbb{N}$ of the outer ADMM scheme and every inner iteration q, there exists $\rho > 0$ such that the directions of the SGD in Algorithm 1 are bounded in average, $\mathbb{E}[\|\mathbf{u}_t^k\|_2^2] \leq \rho^2 < +\infty$ for every $t = 1, \ldots, T$.

Proof. At each outer step k, the direction for the task t is:

$$\mathbf{u}_t^k = \begin{cases} \gamma_t \mathbf{w}_t^k + \frac{1}{m}(\mathbf{y}_t^k - \eta \mathbf{z}_t^k) & \text{if } \ell_{it}^k = 0, \\ -y_{it}\mathbf{x}_{it} + \gamma_t \mathbf{w}_t^k + \frac{1}{m}(\mathbf{y}_t^k - \eta \mathbf{z}_t^k) & \text{if } \ell_{it}^k > 0 \end{cases}$$

where we use the shorthand $\ell_{it}^k = y_{it}\langle \mathbf{w}_t, \mathbf{x}_{it} \rangle$.

We are interested in finding a bound for the quantity $\|\mathbf{u}_t^k\|_2^2$. From the definition we have that

$$\|\mathbf{u}_t^k\|_2^2 \leq \|y_{it}\mathbf{x}_{it}\|_2^2 + \gamma_t^2 \|\mathbf{w}_t^k\|_2^2 + \frac{1}{m^2}(\|\mathbf{y}_t^k\|_2^2 + \eta^2 \|\mathbf{z}_t^k\|_2^2).$$

We have already shown that $\|\mathbf{w}_t^k\|_2^2 \leq \beta$ and from the hypothesis we have that $\|y_{it}\mathbf{x}_{it}\|_2^2 \leq R$. Now, we are interested in finding a finite bound for the quantity $\|\mathbf{y}_t^k\|_2^2 + \eta^2 \|\mathbf{z}_t^k\|_2^2$. From the initialization of the algorithm, we have that $\|\mathbf{y}_t^1\|_2^2 + \eta^2 \|\mathbf{z}_t^1\|_2^2 < +\infty$. In fact, $\|\mathbf{y}_t^1\|_2^2 < +\infty$ and $\|\mathbf{z}_t^1\|_2^2 < +\infty$. We can use the induction for the value $\|\mathbf{y}_t^k\|_2^2 + \eta^2 \|\mathbf{z}_t^k\|_2^2$ changing the step k. We can exploit the hypothesis that

$$\|\mathbf{y}_t^i\|_2^2 + \eta^2 \|\mathbf{z}_t^i\|_2^2 < +\infty, \ \ \forall i < k$$

and then $\|\mathbf{y}_t^i\|_2^2 < +\infty$ and $\eta^2 \|\mathbf{z}_t^i\|_2^2 < +\infty, \ \ \forall i < k$.

By definition, the following inequalities hold

$$\begin{aligned} \|\mathbf{y}_t^k\|_2^2 &\leq \|\mathbf{y}_t^{k-1}\|_2^2 + \|\eta \mathbf{w}_t^{k-1}\|_2^2 + \|\eta \mathbf{z}_t^{k-1}\|_2^2 \leq \|\mathbf{y}_t^{k-1}\|_2^2 + \eta^2 \beta + \eta^2 \|\mathbf{z}_t^{k-1}\|_2^2 \\ &\leq \|\mathbf{y}_t^1\|_2^2 + \sum_{i=2}^{k-1} \eta^2 (\beta + \|\mathbf{z}_t^i\|_2^2) =: \Phi_t^k. \end{aligned}$$

By the induction hypothesis the value $\Phi_t^k < +\infty$. Also, we have that

$$\|\mathbf{z}_t^k\|_2^2 \leq \|E\|_2^2 (\|\mathbf{y}_t^k\|_2^2 + \eta^2 \|\mathbf{w}_t^k\|_2^2),$$

where $E = (2\rho_2 D^T D + (\rho_1 + \eta) I)^{-1}$ is symmetric and then $\|E\|_2^2$ is its spectral radius

$r_E \in \mathbb{R}^+$. Then, we can claim that

$$\|\mathbf{z}_t^k\|_2^2 \leq r_E(\|\mathbf{y}_t^k\|_2^2 + \eta^2\beta) \leq r_E(\Phi_t^k + \eta^2\beta) =: \Psi_t^k < +\infty.$$

Finally, the following bound holds:

$$\|\mathbf{u}_t^k\|_2^2 \leq R + \gamma_t^2\beta + \frac{1}{m^2}(\Phi_t^k + \eta^2\Psi_t^k) := \rho^2 < +\infty.$$

■

13.6 Experimental results

In this section present numerical experiments which highlight the computational efficiency of our algorithm, and report on the advantage offered by the MTL strategy in formula (13.3) in a challenging url classification dataset. Our implementation is available at `https://github.com/torito1984/MTLADMM`.

13.6.1 Artificial data

In the first experiment we generated an artificial dataset which is captured by the model (13.2) and, in addition, introduces a set of irrelevant features.

The number of relevant and irrelevant features was set to 8 and 100, respectively. The dataset is made of 400 different binary classification tasks, 5 of which have 1000 patterns and the rest 16 patterns. To generate the data we first sample the components $w_{01}, \ldots, w_{0d}$ of the mean vector $\mathbf{w}_0$ iid form a zero mean Gaussian with variance $\sigma = 0.25$. Then, we randomly pick 5 of the relevant features and create a vector $\mathbf{w}_t$ for each task by adding a Gaussian perturbation with zero mean and standard deviation $\sigma_i = 2w_{0i}$. Next for each task, we generate a balanced set of points on each side of the classification hyperplane. To generate a point we: (1) pick a sign s with a 50% chance, (2) starting from the origin, move in a random direction $s\mathbf{d}$ where $\mathbf{d}$ is the normal vector of the classification hyperplane and $p \in N(5, 0.1)$ (this generates a well behaved dataset with some random margin violations), (3) finally we choose a random direction in the subspace of dimension $d-1$ parallel to the classification hyperplane and make a jump with magnitude sampled from $N(0, 20)$. Since we are generating patterns with a low margin covering many different directions in the parallel subspace, only the tasks with 1000 patterns have enough data to achieve high test accuracy by their own. For the rest of the tasks – which account for around 55% of the dataset – the number of points is insufficient to achieve a reasonable solution and they should benefit from an MTL approach. We will see that a similar situation arises in the real dataset we cover hereunder.

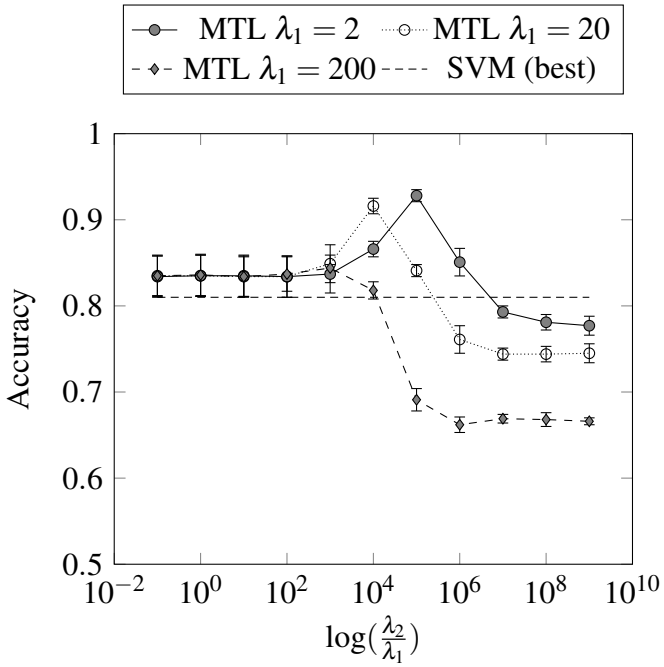


Figure 13.1: Classification performance for the artificial classification dataset

Returning to formula (13.3), it is instructive to observe that, for a fixed $\lambda_1 > 0$, if we let $\lambda_2 \to \infty$, the optimization problem is equivalent to training separate linear SVMs on each task. In this situation and in high dimension, the method will overfit the dataset. On the other hand, for a fixed $\lambda_2 > 0$, if we let $\lambda_1 \to \infty$, the optimization problem is equivalent to training a single linear SVMs on the full dataset, which may underfit the data. In practical situations, we expect the optimal hyperparameters to lay in between these two situations, and obtaining lower accuracy for both extremes.

Classification performance

Figure 13.1 depicts the average test accuracy for a 10-fold CV in the aforementioned dataset. Solid lines depict the accuracy of the proposed algorithm for different values of λ_1 and λ_2. As expected, the optimal hyperparameters lay in between both extremes for λ_2/λ_1 and overfitting arises for when $\lambda_2 \gg \lambda_1$. The dotted line depicts the accuracy of a single SVM trained with a state of the art optimization algorithms for linear SVMs in [96]. We can observe that, with appropriate hyperparameters, MTL is superior to single task learning thanks to the information sharing.

Convergence

One question that may arise is how many ADMM iterations are needed to achieve a good result in practice. This would be the major bottleneck of the proposed method since ADMM iterations are sequential. We ran experiments with the same data generation strategy and varying the dimensionality. The number of relevant features was kept at 10%

of the dimensionality.

First, in Figure 13.2, it can be observed that a higher dimension slows down the convergence to the optimum in the first iterations, an order of tens of iterations is enough to obtain a good accuracy. Second, we compared a MatLab implementation of the standard MTL called MALSAR[1] in performance and computational complexity varying the number of different tasks in the set $\{20, 40, 60, 80, 100, 200, 400, 800, 1200\}$. We keep the sample sizes per task fixed and measure MALSAR running time. For our method, we are interested in finding the number of outer ADMM iterations needed to converge to the same accuracy of MALSAR with a tolerance of 10^{-4}.

In Table 13.1, a comparison of the computational complexities is presented. We collected the CPU time required by the standard MTL implementation in order to find the optimal solution, varying the number of tasks. We compared these values with respect to the number of outer ADMM steps that our algorithm required to reach the same solution. For this purpose, we introduce the variable$\alpha = \frac{\lambda_2}{\lambda_1+\lambda_2}$. This variable imposes the quantity of shared information among the tasks. In fact, when α is equal to zero, we are training a single SVM for all the tasks, on the other hand, with $\alpha = 1$ we are training a different SVM for each single task.

In comparison we observed that the CPU time of MALSAR varies quadratically with number of tasks. For example for alpha=0.5 the CPU was of 3.3, 4.8,..., secs. for T=20,40,...,1200

From these results, we are able to claim that the standard MTL implementation has a quadratic complexity with respect to the number of tasks, whereas our ADMM implementation is able to reach the same optimal solution in a fixed number of outer steps.

Tasks	MALSAR time (s)	ADMM steps			
	$\alpha = 0.5$	$\alpha = 0.0$	$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 1.0$
20	3.3	21	16	10	10
40	4.8	21	16	11	11
60	6.4	21	16	10	10
80	8.3	21	16	10	10
100	10.7	21	16	10	10
200	33.1	21	16	11	11
400	95.8	21	16	11	11
800	342.5	21	16	11	11
1200	698.3	21	16	10	10

Table 13.1: CPU time of MALSAR algorithm and number of outer steps of our ADMM algorithm in order to reach the same accuracy of MALSAR, with different amounts of tasks. The variable α is equal to $\alpha = \frac{\lambda_2}{\lambda_1+\lambda_2}$

[1]MALSAR code: http://www.public.asu.edu/ jye02/Software/MALSAR

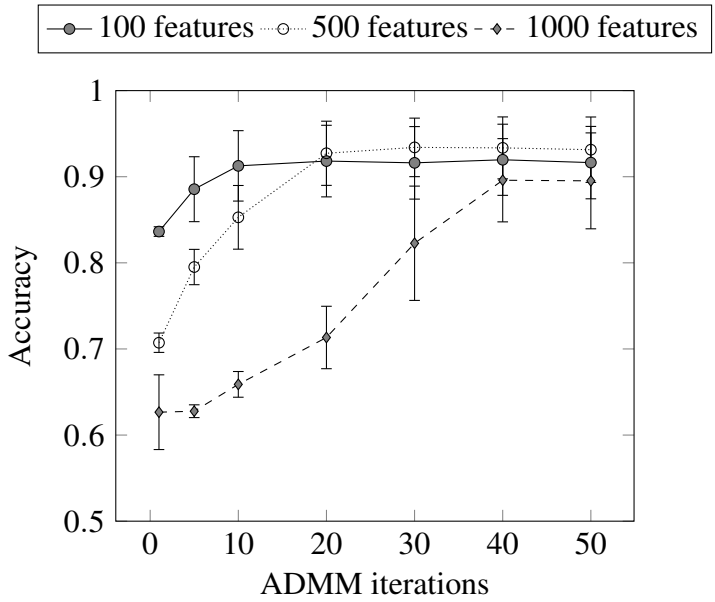


Figure 13.2: Performance of ADMM with different dimensionality

13.6.2 URL classification

The next dataset was extracted from a production environment in an online advertising firm. For online targeting, a data provider supplies data to an advertiser indicating which users are interested in specific product classes. In order to determine the interest of each user, web pages are first classified into interest categories (eg 'fashion', 'photography' etc). Thereafter the browsing behavior of users can be used to predict their interests. One way of achieving it is to predict the category of a page from the contents of a url. We can tackle this task using a bag-of-words strategy [108]. Modern URLs carry enough information to identify the page contents thanks to Search Engine Optimization strategies. Before being able to apply this, words should be detected and segmented into meaningful n-grams. This process can be easily achieved though a Viterbi-like strategy. The details of the algorithm used can be found in [165, Ch. 4]. It calculates the most likely segmentation of a string containing text with no spaces based on a Viterbi-like probability maximization process and the prior probabilities of different n-grams extracted from the 1 Trillion words dataset gathered by Google [37]. Once the URL is segmented, a bag-of-words representation is built.

The dataset treated is a collection of URLs extracted from the client base at Skimlinks labeled by humans as related to fashion or not. The dataset is a compound of over 500,000 URLs extracted from 4,326 different client hosts embedded in a 150,000 dimension bag of words vector space. Thus, we have a binary classification task for a set of different tasks (clients/hosts). We hypothesize that different clients may use different vocabularies and ways of building URLs, but that the underlying set of n-grams that indicate fashion should

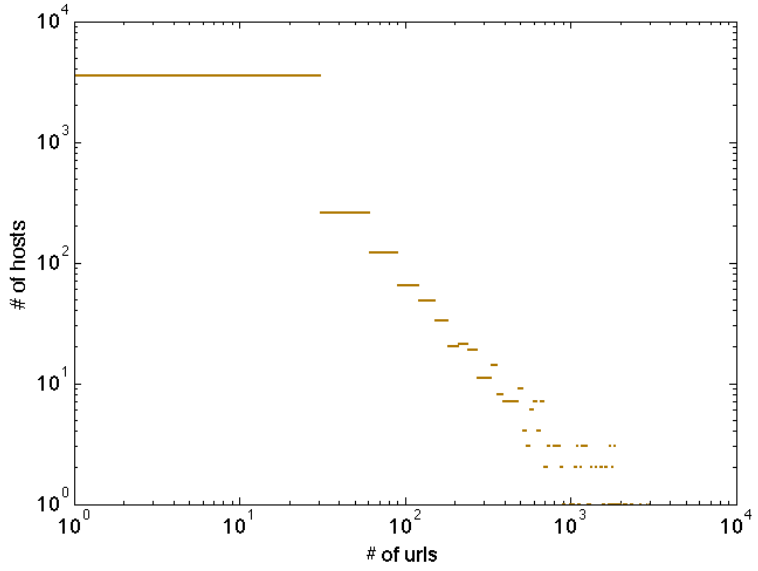


Figure 13.3: # of patterns per task (client) in the URL classification dataset.

be shared between all of them. In addition, we can see in Figure 13.3 that the number of URLs extracted from each client is highly skewed, so most tasks do not have enough data to build a reliable model in such a high dimensional space. This problem's scenario is similar to the artificial case previously treated and so we expect that sharing information between tasks could be beneficial.

Table 13.2 summarizes the results for the categorization dataset when compared to its single task and individual tasks counterparts. As it can be observed in the third column, there is an expected gain of 6% accuracy when compared with a single SVM model trained for all the the tasks and of 3% when compared to an SVM trained individually for each task. When comparing these results with the distribution of patterns per task, we can observe that there is enough variety between tasks that hinders SVM capacity to adapt to all of them. On the other hand, for less populated tasks there is not enough data to achieve a high accuracy and so all the information is not exploited. The capacity to transfer information embedded in MTL seems to balance the right amount of regularization and information sharing, giving the best results.

One of the questions in systems where an individualised response is needed is what to do in a cold start situation, i.e. what response to provide when we do not have data for some individual task. Single model training would advocate to use the same model as for all the tasks, whereas if we choose to have individualised models, we would lack the model for that case. It seems natural to think that the average model $\mathbf{w}_0$ in formula (13.2), which constitutes the bias for any task in the context, would be the right answer. In the previous dataset we discarded 29753 patterns from unpopulated task. So these tasks would

Method	Validation	Train	Test
Linear SVM single task	$0.780_{\pm 0.014}$	$0.813_{\pm 0.021}$	$0.768_{\pm 0.083}$
Linear SVM individual tasks	$0.793_{\pm 0.001}$	$0.926_{\pm 0.027}$	$0.811_{\pm 0.001}$
ADMM_5	$0.801_{\pm 0.001}$	$0.918_{\pm 0.007}$	$0.814_{\pm 0.001}$
ADMM_{30}	$0.817_{\pm 0.001}$	$0.887_{\pm 0.003}$	$0.836_{\pm 0.001}$
ADMM_{50}	$0.817_{\pm 0.001}$	$0.891_{\pm 0.003}$	$0.835_{\pm 0.001}$

Table 13.2: Classification Accuracy$_{\pm std}$ for Skimlinks dataset.

be a sample of “cold start" of tasks we do not have data on. The accuracy of $\mathbf{w}_0$ and of the single task SVM previously obtained are 82% and 80% respectively. When unified with the results in Table 13.2, we can see that MTL shows a good adaptation to individual tasks and, at the same time, a good prior bias is also discovered.

IV

Conclusions

14. Conclusions

In this thesis we have focused on the representation in machine learning problems and proposed new theories and algorithms to learn better representations exploiting the structure of the feature space of kernels.

The main contribution of the thesis is a new perspective on MKL. MKL algorithms combine predefined kernels. The effectiveness of traditional approaches are questionable. In fact, even if theoretically well founded, the empirical results on real problems do not always confirm this theoretical findings. For instance, in many task they performed more or less like very simple baselines, e.g. the simple average of kernels. In this thesis, we have investigated on principled criteria to use when defining the set of base kernels in such a way to improve the MKL combination effectiveness. Specifically, criteria like the structure of the feature space and the orthogonality of the features have been studied.

We firstly introduced a new MKL algorithm in Chapter 7, called EasyMKL. This algorithm is able to combine thousands of *weak* kernels in a very efficient way, i.e. using a fixed amount of memory and with linear computational complexity in time. Given its nice scalability, EasyMKL can be used in a setting where one wants to combine a large number of *weak* kernels with a fine-grained (and possibly diverse) information. Having many small pieces of information allows EasyMKL to recompose the information with the best combination possible by maximizing the obtained margin. With this new methodology, our MKL algorithm outperformed both the (strong) average baseline and all the state-of-the-art MKL methods it was compared with. Finally, EasyMKL also seems very robust with respect to the noise introduced by features which are not informative and works well even

in the classical MKL framework, where a small number of *weak* kernels is used.

The EasyMKL algorithm, like the majority of state-of-the-art MKL methods, optimizes the margin and does not take into account directly the complexity (or expressiveness) of the combined kernels. In this thesis, a new measure for the complexity of a kernel function called spectral complexity is also proposed in Chapter 8. The spectral complexity is an easy to evaluate measure of expressiveness and we proved its strict connection with the *empirical Rademacher complexity* and hence, with the generalization error. Intuitively, kernel functions built on simpler and more popular features will have lower complexity while kernel functions built on more elaborated and rare features will have higher complexity. Typically, classifiers defined on too simple features are not able to discriminate correctly the class and, similarly, too complex features cannot guarantee a large margin for many examples in the training set. This intuition has motivated the idea of a MKL combination of weak kernels of increasing complexity.

For this, a property on sets of (weak) kernels (MSC property) has been defined. This property guarantees a hierarchical structure among the features spaces of different *weak* kernels. The idea is then to exploit the structure of the set of *weak* kernels in order to improve the MKL performance.

In Chapter 9, experiments have been performed to highlight the importance of the hierarchical structure guaranteed by the MSC property.

In a first setting, weak kernel optimization has been applied to dot-product kernels, that is kernels in the form $K(\mathbf{x},\mathbf{z}) = f(\mathbf{x}\cdot\mathbf{z})$. We have shown that popular kernels, like RBF and Polynomials, are indeed contained in this wide class of kernels and can be represented as a dot-product polynomial with fixed polynomial coefficients. The proposed approach is able to learn a kernel in the space of dot-product polynomials. We have shown empirical evidence that our approach outperforms the baselines, sometimes significantly, against ten benchmark datasets. We have also demonstrated that building *weak* kernels by exploiting the structure of the features and their dependencies, makes the generated kernel improved upon alternatives which do not exploit this structure.

In a second setting, the same methodology has been applied to the ODD graph-kernel, a state-of-the-art kernel for graphs. In this case, a MSC set of *weak* kernels has been defined exploiting the definition of the kernel. In particular, its tree structured features have been grouped based on their height. This leads to a set of week kernels of increasing spectral complexity (MSC). Also in this case, our experiments have confirmed that this methodology improves the results of EasyMKL and outperforms the original ODD graph-kernel.

These findings open a new scenario in MKL beyond the combination of a large number of *weak* kernels. The idea is to perform a pre-training step consisting of the optimization

of the set of weak kernels to combine. The entire processing can be summarized in the following three steps:

1. Define a good (and possibly large) family of *weak* kernels starting from the raw information;
2. Combine all the *weak* kernels by exploiting MKL algorithms in order to fix the final representation;
3. Create a model using a kernel machine.

Schematically, the methodology can be summarized as in the following:

$$\mathbf{x} \in \mathcal{X} \rightarrow \underbrace{\{\mathbf{K}_1, \ldots, \mathbf{K}_R\}_{learned}}_{1} \rightarrow \underbrace{\mathbf{K}_{learned} = \sum_{i=1}^{R} \mu_i \mathbf{K}_i \sim \boldsymbol{\phi}_{\mathbf{K}}(\mathbf{x})}_{2} \rightarrow \underbrace{\mathbf{w}_{learned} \cdot \boldsymbol{\phi}_{\mathbf{K}}(\mathbf{x})}_{3}.$$

It is important to note that in the first step of this procedure ($\mathbf{x} \in \mathcal{X} \rightarrow \{\mathbf{K}_1, \ldots, \mathbf{K}_R\}_{learned}$) only the hierarchical structure implicitly defined in the RKHS is exploited.

Pursuing the same idea, a new algorithm has been presented in Chapter 10 to learn the parameters of an anisotropic RBF kernel. This algorithm can be seen as a MKL algorithm where the rule for the combination of the *weak* kernels is the entry-wise product of the *weak* matrices. We considered each *weak* kernel as a $1-$feature RBF (i.e. RBF kernel that considers only 1 feature) and optimized the parameter of this family of RBFs. Exploiting this methodology, an optimal family of *weak* kernels has been created. The obtained results are very good compared to the state-of-the-art DML methods in term of AUC. Moreover, our algorithm can be used also for an explicit feature weighting/selection by using the weights assigned by EasyMKL to the $1-$feature RBFs.

In the final chapters, our expertise in representation learning has been challenged by real world tasks: noise or large scale datasets and real-time applications. Specifically, solutions for three different real world applications have been described. In Chapter 11, a biomedical application is considered, concerning the segmentation of brain MRI volumes. A two-steps model is applied in order to exploit the domain knowledge *a priori*. Using this model, the different underlying factors of variation for this task are manually detected. We have shown that this two-step model improves on the previous state-of-the-art results. In Chapter 12, an applications for smartphones has been presented, called *Climb The World*. The machine learning goal of this application was the counting in real-time of the number of stair-steps performed by a user by exploiting the sensors of the device. An extensive work of feature engineering and feature selection has been performed. EasyMKL has been used for this task with nice results under the constraint of battery consumption, imposed by the device. We achieved the goal of a very good real-time classification of the stairsteps and, at the best of our knowledge, this was the first algorithm able to solve this

very difficult task. Finally, in Chapter 13, a new Multitask Learning (MTL) algorithm has been presented dealing with large scale datasets by using the alternating direction method of multipliers (ADMM). In the MTL context, this algorithm outperforms the previous state-of-the-art methods used in the classification of large scale datasets.

15. Future work

In this thesis, we have proposed several ideas trying to unify different fields of machine learning under the same challenging task of the representation learning.

Based on our analysis, the idea to be followed should be the creation of a novel theoretical framework that could include and highlight the connection between feature learning, kernel learning, manifold learning and the analysis of the complexity as different shades of the same task. As highlighted in Part II, the unification among these (apparently) different fields becomes clear in several cases:

- Comparing the MKL ℓ_1 algorithms with respect to the Group Lasso feature selection methods;
- Considering the MKL methods applied to $1-$rank kernels and their connection to the feature weighting. We can consider the $1-$rank kernels as linear kernels evaluated by exploiting only one feature[1]. Also, the $1-$rank kernels naturally appear in the *spectral embedding* decomposition (see Section 3.3.2);
- Exploiting the complexity as a measure to create optimal *weak* families of kernels for the MKL problem.

Starting from our experience with kernel and feature learning algorithms, the priority milestones should be:

Kernel approximation. A problem that often afflicts the kernel methods is the difficulty of their scaling with big data. The representations given by kernels (or combinations

[1] Corinna Cortes, *Can learning kernels help performance?* `http://static.googleusercontent.com/media/research.google.com/it//pubs/archive/36227.pdf`

of kernels) are very powerful but useless when they have to tackle huge amounts of data. A possible solution in literature is to efficiently approximate the original kernel [174]. Another possible solution is to investigate a greedy (sparse) method to select the explicit features of the kernel. These randomly generated features (e.g. the features of the RBF kernels [54]) have the property to approximate very quickly the original kernel and obtaining the linearity of the final model.

Optimal *weak* kernels families. As proven in Chapter 9, Multiple Kernel Learning results are improved by providing an optimal family of *weak* kernels. A generalization of the Monotonically Spectral-Complex (see Chapter 8) can be studied in order to exploit the connection to the hierarchical structure of the features. Furthermore, new methods can be developed to automatically generate MSC families, not only starting from DPP kernels. For example, the *Recommender System* framework represents an optimal application for this development. In this context, the users represent the features and the items are the examples. By applying the MSC idea, we can divide the users in bunches of different spectral complexity injecting a new sense in the definition of expressiveness: users are less complex if their preferences are more popular.

MKL and features learning. The MKL algorithms paradigm can be exploited to combine a very large set of weak kernels aiming at boosting their accuracy, as shown in Chapter 7 with EasyMKL. In this way, the final combination will represent a weighted combination of the different subsets of features and the method performs a data-driven feature learning/selection/weighting. EasyMKL can be modified in order to combine a *weak* family generated by considering only one feature for each kernel, as in Chapter 12. The theoretical and practical applications of this paradigms have to be deeply investigated.

Distance metric learning and kernel learning. In Chapter 10, a method has been presented with a connection with these two research fields. It is well known that the performance of a learning algorithm (nearest-neighbors classifiers, kernel algorithms etc.) mostly depends on the metric used [195][2]. The idea is to develop these concepts in order to create new algorithms that are able to inject automatically the optimal metric information directly in the kernel.

Real world applications. In the last few years, several datasets with high dimensionality have become publicly available on the Internet. This fact has brought an interesting challenge to the research community, since for the machine learning methods it is difficult to deal with a high number of input. In fact, in real world application we

[2] Barbara Hammer, *Metric Learning and Model Interpretability*, `https://lecture2go.uni-hamburg.de/veranstaltungen/-/v/16627`

have to deal with a big and complex amount of data and the correct selection of the information is crucial in order to avoid an exponential growth of the computational time.

In summary, the final goal is to make new algorithms that should be automatically able to generate the optimal representation of the data (given a specific task, a specific space or a set of tasks) avoiding the arbitrary choice from the users that is sub-optimal and time consuming, especially when dealing with big data.

Bibliography

[1] Fabio Aiolli, Matteo Ciman, Michele Donini, and Ombretta Gaggi. Climbtheworld: Real-time stairstep counting to increase physical activity. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pages 218–227. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014.

[2] Fabio Aiolli and Michele Donini. Easy multiple kernel learning. In*22th European Symposium on Artificial Neural Networks, ESANN 2014, Bruges, Belgium, April 23-25, 2014*, 2014.

[3] Fabio Aiolli and Michele Donini. Learning anisotropic RBF kernels. In *Artificial Neural Networks and Machine Learning - ICANN 2014 - 24th International Conference on Artificial Neural Networks, Hamburg, Germany, September 15-19, 2014. Proceedings*, pages 515–522, 2014.

[4] Fabio Aiolli and Michele Donini. Easymkl: a scalable multiple kernel learning algorithm. *Neurocomputing*, 169:215–224, 2015.

[5] Fabio Aiolli, Michele Donini, Enea Poletti, and Enrico Grisan. Stacked models for efficient annotation of brain tissues in MR volumes. *IFMBE Proceedings*, 41:261–264, 2014.

[6] Fabio Aiolli, Giovanni Da San Martino, and Alessandro Sperduti. A kernel method for the optimization of the margin distribution. In *ICANN (1)*, pages 305–314, 2008.

[7] Erin L. Allwein, Robert E. Schapire, and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000.

[8] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2014.

[9] Aouatif Amine, Ali El Akadi, Mohammed Rziza, and Driss Aboutajdine. Ga-svm and mutual information based frequency feature selection for face recognition. *GSCM-LRIT, Faculty of Sciences, Mohammed V University, BP*, 1014, 2009.

[10] Davide Anguita, Alessandro Ghio, L. Oneto, and Sandro Ridella. A Deep Connection Between the Vapnik-Chervonenkis Entropy and the Rademacher Complexity. *IEEE Transactions on Neural Networks and Learning Systems*, (12):2202–2211.

[11] J. Ashburner and K.J. Friston. Unified segmentation. *NeuroImage*, 26(3):839–851, 2005. cited By (since 1996) 1330.

[12] E Ataer-Cansizoglu, J Kalpathy-Cramer, S You, K Keck, D Erdogmus, MF Chiang, et al. Analysis of underlying causes of inter-expert disagreement in retinopathy of prematurity diagnosis. *Methods Inf Med*, 54(1):93–102, 2015.

[13] Francis Bach. Exploring Large Feature Spaces with Hierarchical Multiple Kernel Learning. (2).

[14] Francis Bach. Hierarchical kernel learning. *Nips*, (May):1–18.

[15] Francis Bach, Julien Mairal, and Jean Ponce. Convex sparse matrix factorizations. *arXiv preprint arXiv:0812.1869*, 2008.

[16] Francis R. Bach. Exploring large feature spaces with hierarchical multiple kernel learning. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 105–112. Curran Associates, Inc., 2009.

[17] Francis R Bach, Gert RG Lanckriet, and Michael I Jordan. Multiple kernel learning, conic duality, and the smo algorithm. In *Proceedings of the twenty-first international conference on Machine learning*, page 6. ACM, 2004.

[18] K. Bache and M. Lichman. Uci machine learning repository, 2013.

[19] Jing Bai, Ke Zhou, Guirong Xue, Hongyuan Zha, Gordon Sun, Belle Tseng, Zhaohui Zheng, and Yi Chang. Multi-task learning for learning to rank in web search. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1549–1552. ACM, 2009.

[20] Peter L. Bartlett, Olivier Bousquet, and Shahar Mendelson. Local Rademacher complexities. *The Annals of Statistics*, (4):1497–1537.

[21] Pl Peter L Bartlett and Shahar Mendelson. Rademacher and Gaussian Complexities: Risk Bounds and Structural Results. *Journal of Machine Learning Research*, (3):463–482.

[22] Jonathan Baxter. A model of inductive bias learning. *J. Artif. Intell. Res.(JAIR)*, 12:149–198, 2000.

[23] Lluís a. Belanche and Alessandra Tosi. Averaging of kernel functions. *Neurocomputing*, 112(April):19–25, 2013.

[24] Aurélien Bellet, Amaury Habrard, and Marc Sebban. A survey on metric learning for feature vectors and structured data. *arXiv preprint arXiv:1306.6709*, 2013.

[25] R. Bellman. Adaptive control processes: A guided tour. *Princeton University Press, New Jersey*, 1961.

[26] Shai Ben-David and Reba Schuller. Exploiting task relatedness for multiple task learning. In *Learning Theory and Kernel Machines*, pages 567–580. Springer, 2003.

[27] Yoshua Bengio, Yoshua Bengio, Olivier Delalleau, Olivier Delalleau, Nicolas Le Roux, Nicolas Le Roux, Downtown Branch, and Downtown Branch. The Curse of Dimensionality for Local Kernel Machines. 2(2):1–17, 2005.

[28] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *Pattern Analysis and . . .* , (1993):1–30.

[29] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy Layer-Wise Training of Deep Networks. *Advances in neural information processing systems*, 19(1):153, 2007.

[30] Yoshua Bengio, Jean-Francois Paiement, and Pascal Vincent. Out-of-sample extensions for lle, isomap, mds, eigenmaps, and spectral clustering. In *In Advances in Neural Information Processing Systems*, pages 177–184. MIT Press, 2003.

[31] Adriana Birlutiu, Perry Groot, and Tom Heskes. Efficiently learning the preferences of people. *Machine Learning*, 90(1):1–28, 2013.

[32] V Bolón-Canedo, N Sánchez-Maroño, A Alonso-Betanzos, JM Benítez, and F Herrera. A review of microarray datasets and applied feature selection methods. *Information Sciences*, 282:111–135, 2014.

[33] Verónica Bolón-Canedo, Noelia Sánchez-Maroño, and Amparo Alonso-Betanzos. A review of feature selection methods on synthetic data. *Knowledge and information systems*, 34(3):483–519, 2013.

[34] Verónica Bolón-Canedo, Noelia Sánchez-Maroño, and Amparo Alonso-Betanzos. Data classification using an ensemble of filters. *Neurocomputing*, 135:13–20, 2014.

[35] Olivier Bousquet and Léon Bottou. The tradeoffs of large scale learning. In *Advances in neural information processing systems*, pages 161–168, 2008.

[36] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[37] Thorsten Brants and Alex Franz. {Web 1T 5-gram Version 1}. 2006.

[38] Gavin Brown, Adam Pocock, Ming-Jie Zhao, and Mikel Luján. Conditional likelihood maximisation: a unifying framework for information theoretic feature selection. *The Journal of Machine Learning Research*, 13(1):27–66, 2012.

[39] Serhat S Bucak, Rong Jin, and Anil K Jain. Multiple kernel learning for visual object recognition: A review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(7):1354–1369, 2014.

[40] S.S. Bucak, R. Jin, and Ak. Jain. Multiple Kernel Learning for Visual Object Recognition: A Review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (7):1354–1369.

[41] Evan Calabrese, Alexandra Badea, Charles Watson, and G Allan Johnson. A quantitative magnetic resonance histology atlas of postnatal rat brain development with regional estimates of growth and variability. *NeuroImage*, 71C:196–206, January 2013.

[42] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[43] Eduardo Castro, Vanessa Gómez-Verdejo, Manel Martínez-Ramón, Kent a. Kiehl, and Vince D. Calhoun. A multiple kernel learning approach to perform classification of groups from complex-valued fMRI data analysis: Application to schizophrenia. *NeuroImage*, pages 1–17.

[44] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006.

[45] William W. Cohen. Stacked sequential learning. In *International Joint Conference on Artificial Intelligence*, pages 671–676, 2005.

[46] Michael Collins and Nigel" Duffy. Convolution Kernels for Natural Language. *Advances in Neural Information Processing Systems*, 14:625–632, 2001.

[47] Corinna Cortes. Learning Kernels Using Local Rademacher Complexity. *Nips*, pages 1–9, 2013.

[48] Corinna Cortes, Marius Kloft, and Mehryar Mohri. Learning kernels using local rademacher complexity. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors,*Advances in Neural Information Processing Systems 26*, pages 2760–2768. Curran Associates, Inc., 2013.

[49] Corinna Cortes, M Mohri, and a Rostamizadeh. Learning non-linear combinations of kernels. *Advances in Neural Information . . .*, pages 1–9.

[50] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Learning sequence kernels. In *Machine Learning for Signal Processing, 2008. MLSP 2008. IEEE Workshop on*, pages 2–8. IEEE, 2008.

[51] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Generalization bounds for learning kernels. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 247–254, 2010.

[52] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Algorithms for learning kernels based on centered alignment. *The Journal of Machine Learning Research* , 13(1):795–828, 2012.

[53] Andrew Cotter, Joseph Keshet, and Nathan Srebro. Explicit Approximations of the Gaussian Kernel. *arXiv preprint arXiv:1109.4603*, page 11.

[54] Andrew Cotter, Joseph Keshet, and Nathan Srebro. Explicit approximations of the gaussian kernel. *arXiv preprint arXiv:1109.4603*, 2011.

[55] Nello Cristianini, John Shawe-Taylor, André Elisseeff, and Jaz S. Kandola. On kernel-target alignment. In Dietterich et al. [62], pages 367–373.

[56] Giovanni Da San Martino, Nicolo Navarin, and Alessandro Sperduti. A memory efficient graph kernel. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*. Ieee, June 2012.

[57] Giovanni Da San Martino, Nicolò Navarin, and Alessandro Sperduti. A Tree-Based Kernel for Graphs. In *Proceedings of the Twelfth SIAM International Conference on Data Mining*, pages 975–986, 2012.

[58] Giovanni Da San Martino, Nicolò Navarin, and Alessandro Sperduti. Exploiting the ODD framework to define a novel effective graph kernel. In *23th European Symposium on Artificial Neural Networks, ESANN 2015, Bruges, Belgium, April 22-24, 2015*, 2015.

[59] Anirban Dasgupta, Petros Drineas, Boulos Harb, Vanja Josifovski, and Michael W Mahoney. Feature selection methods for text classification. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 230–239. ACM, 2007.

[60] Manoranjan Dash and Huan Liu. Consistency-based search in feature selection. *Artificial intelligence*, 151(1):155–176, 2003.

[61] Costanza D'Avanzo, Anahita Goljahani, Gianluigi Pillonetto, Giuseppe De Nicolao, and Giovanni Sparacino. A multi-task learning approach for the extraction of single-trial evoked potentials. *Computer methods and programs in biomedicine*, 110(2):125–136, 2013.

[62] Thomas G Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors. *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*. MIT Press, 2001.

[63] Paul D Dobson and Andrew J Doig. Distinguishing Enzyme Structures from Non-enzymes Without Alignments. *Journal of Molecular Biology*, 330(4):771–783, 2003.

[64] Carlotta Domeniconi and Dimitrios Gunopulos. Adaptive nearest neighbor classification using support vector machines. In Dietterich et al. [62], pages 665–672.

[65] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons,, 1999.

[66] Jonathan Eckstein and Dimitri P Bertsekas. On the douglas—rachford splitting method and the proximal point algorithm for maximal monotone operators*Mathematical Programming*, 55(1-3):293–318, 1992.

[67] Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. *International Conference on Artificial Intelligence and Statistics*, pages 153–160.

[68] Theodoros Evgeniou and Massimiliano Pontil. Regularized multi–task learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 109–117. ACM, 2004.

[69] B. J. Fogg. A behavior model for persuasive design. In *Proceedings of the 4th International Conference on Persuasive Technology*, Persuasive '09, pages 1–7, New York, NY, USA, 2009. ACM.

[70] George Forman. An extensive empirical study of feature selection metrics for text classification. *The Journal of ;achine Learning Research*, 3:1289–1305, 2003.

[71] George Forman. Feature selection for text classification. *Computational Methods of Feature Selection*, pages 257–276, 2008.

[72] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

[73] R. Gelineau-Morel, V. Tomassini, and M. Jenkinson. The effect of hypointense white matter lesions on automated gray matter segmentation in multiple sclerosis. *Hum Brain Mapp*, 2011. cited By (since 1996) 4.

[74] Ali Gholipour, Alireza Akhondi-Asl, Judy a Estroff, and Simon K Warfield. Multi-atlas multi-shape segmentation of fetal brain MRI for volumetric and morphometric analysis of ventriculomegaly. *NeuroImage*, 60(3):1819–31, April 2012.

[75] Amir Globerson and Sam T. Roweis. Metric learning by collapsing classes. In *NIPS*, 2005.

[76] Jacob Goldberger, Sam T. Roweis, Geoffrey E. Hinton, and Ruslan Salakhutdinov. Neighbourhood components analysis. In *NIPS*, 2004.

[77] Tom Goldstein, Brendan O'Donoghue, Simon Setzer, and Richard Baraniuk. Fast alternating direction optimization methods. *SIAM Journal on Imaging Sciences*, 7(3):1588–1623, 2014.

[78] Mehmet Gönen and Ethem Alpaydin. Localized multiple kernel learning. *Proceedings of the 25th international conference on Machine learning - ICML '08*, (x):352–359.

[79] Mehmet Gönen and Ethem Alpaydin. Multiple kernel learning algorithms. *Journal of Machine Learning Research*, 12:2211–2268, 2011.

[80] Ioannis S Gousias, Daniel Rueckert, Rolf a Heckemann, Leigh E Dyet, James P Boardman, a David Edwards, and Alexander Hammers. Automatic segmentation of brain MRIs of 2-year-olds into 83 regions of interest. *NeuroImage*, 40(2):672–84, April 2008.

[81] Isabelle Guyon. *Feature extraction: foundations and applications*, volume 207. Springer, 2006.

[82] Isabelle Guyon, Asa Ben Hur, Steve Gunn, and Gideon Dror. Result analysis of the nips 2003 feature selection challenge. In *Advances in Neural Information Processing Systems 17*, pages 545–552. MIT Press, 2004.

[83] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1-3):389–422, 2002.

[84] Piotr a Habas, Kio Kim, James M Corbett-Detig, Francois Rousseau, Orit a Glenn, a James Barkovich, and Colin Studholme. A spatiotemporal atlas of MR intensity, tissue probability and shape of the fetal brain with application to segmentation. *NeuroImage*, 53(2):460–70, November 2010.

[85] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[86] Mark A Hall. *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato, 1999.

[87] Robert M. Haralick. Statistical and structural approaches to texture. *Proc IEEE*, 67(5):786–804, 1979. cited By (since 1996) 1640.

[88] Anne-Claire Haury, Pierre Gestraud, and Jean-Philippe Vert. The influence of feature selection methods on accuracy, stability and interpretability of molecular signatures. *PloS one*, 6(12):e28210, 2011.

[89] David Haussler. Convolution Kernels on Discrete Structures. Technical report, Department of Computer Science, University of California at Santa Cruz, 1999.

[90] Bingsheng He and Xiaoming Yuan. On the o(1/n) convergence rate of the douglas-rachford alternating direction method. *SIAM Journal on Numerical Analysis*, 50(2):700–709, 2012.

[91] Christoph Helma, Tobias Cramer, Stefan Kramer, and Luc De Raedt. Data mining and machine learning techniques for the identification of mutagenicity inducing substructures and structure activity relationships of noncongeneric compounds. *Journal of Chemical Information and Computer Sciences*, 44(4):1402–1411, 2004.

[92] Geoffrey E Hinton. A Fast Learning Algorithm for Deep Belief Nets. 1554:1527–1554, 2006.

[93] S Hochreiter, Y Bengio, P Frasconi, and J Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. *A Field Guide to Dynamical Recurrent Networks*, pages 237–243.

[94] Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1–32, 1997.

[95] Mingyi Hong and Zhi-Quan Luo. On the linear convergence of the alternating direction method of multipliers. *arXiv preprint arXiv:1208.3922*, 2012.

[96] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S Sathiya Keerthi, and Sellamanickam Sundararajan. A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th international conference on Machine learning*, pages 408–415. ACM, 2008.

[97] Hui-Huang Hsu, Cheng-Wei Hsieh, and Ming-Da Lu. Hybrid feature selection by combining filters and wrappers. *Expert Systems with Applications*, 38(7):8144–8150, 2011.

[98] Jianping Hua, Waibhav D Tembe, and Edward R Dougherty. Performance of feature-selection methods in the classification of high-dimension data. *Pattern Recognition*, 42(3):409–424, 2009.

[99] Cheng-Lung Huang and Cheng-Yi Tsai. A hybrid sofm-svr with a filter-based feature selection for stock market forecasting. *Expert Systems with Applications*, 36(2):1529–1539, 2009.

[100] Shu Huang, Wei Peng, Jingxuan Li, and Dongwon Lee. Sentiment and topic analysis on social media: a multi-task multi-label classification approach. In *Proceedings of the 5th annual ACM web science conference*, pages 172–181. ACM, 2013.

[101] Zakria Hussain and John Shawe-Taylor. Improved loss bounds for multiple kernel learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, pages 370–377, 2011.

[102] Tâm Huynh and Bernt Schiele. Analyzing Features for Activity Recognition. *Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies*, (october):159–163.

[103] G Ilczuk, R Mlynarski, W Kargul, and A Wakulicz-Deja. New feature selection methods for qualification of the patients for cardiac pacemaker implantation. In *Computers in Cardiology, 2007*, pages 423–426. IEEE, 2007.

[104] A. Jain, S. V. N. Vishwanathan, and M. Varma. Spg-gmkl: Generalized multiple kernel learning with a million kernels. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, August 2012.

[105] Pratik Jawanpuria. Generalized Hierarchical Kernel Learning. 16:617–652, 2015.

[106] Pratik Jawanpuria, Manik Varma, and Saketha Nath. On p-norm path following in multiple kernel learning for non-linear feature selection. In Tony Jebara and Eric P. Xing, editors, *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 118–126. JMLR Workshop and Conference Proceedings, 2014.

[107] Rodolphe Jenatton, Julien Mairal, Francis R Bach, and Guillaume R Obozinski. Proximal methods for sparse hierarchical dictionary learning. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 487–494, 2010.

[108] Thorsten Joachims. *Learning to classify text using support vector machines: Methods, theory and algorithms*. Kluwer Academic Publishers, 2002.

[109] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[110] Sham M Kakade, Shai Shalev-Shwartz, and Ambuj Tewari. Regularization techniques for learning with matrices. *The Journal of Machine Learning Research*, 13(1):1865–1890, 2012.

[111] Hamidreza Rashidy Kanan and Karim Faez. An improved feature selection method based on ant colony optimization (aco) evaluated on face recognition system. *Applied Mathematics and Computation*, 205(2):716–725, 2008.

[112] Purushottam Kar and Harish Karnick. Random feature maps for dot product kernels. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2012, La Palma, Canary Islands, April 21-23, 2012*, pages 583–591, 2012.

[113] Dor Kedem, Stephen Tyree, Fei Sha, Gert R. Lanckriet, and Kilian Q. Weinberger. Non-linear metric learning. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2573–2581. Curran Associates, Inc., 2012.

[114] Sahand Khakabimamaghani, Farnaz Barzinpour, and Mohammad R Gholamian. Enhancing ensemble performance through feature selection and hybridization. *International Journal of Information Processing and Management* , 2(2), 2011.

[115] Hyunsoo Kim, Peg Howland, and Haesun Park. Dimension reduction in text classification with support vector machines. In *Journal of Machine Learning Research*, pages 37–53, 2005.

[116] Kyoung-jae Kim and Ingoo Han. Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert systems with Applications*, 19(2):125–132, 2000.

[117] Marius Kloft. Lp-Norm Multiple Kernel Learning. *Journal of Machine Learning Research*, (3):953–997.

[118] Marius Kloft and Gilles Blanchard. The local rademacher complexity of lp-norm multiple kernel learning. In *Advances in Neural Information Processing Systems*, pages 2438–2446, 2011.

[119] V. Koltchinskii. Rademacher penalties and structural risk minimization. *IEEE Transactions on Information Theory*, 47(5):1902–1914, 2001.

[120] Vladimir Koltchinskii. Local Rademacher complexities and oracle inequalities in risk minimization. *The Annals of Statistics*, (6):2593–2656.

[121] Igor Kononenko. Estimating attributes: analysis and extensions of relief. In *Machine Learning: ECML-94*, pages 171–182. Springer, 1994.

[122] Gert R G Lanckriet, Nello Cristianini, Peter Bartlett, Laurent El Ghaoui, and Michael I Jordan. Learning the Kernel Matrix with Semidefinite Programming. *Journal of Machine Learning Research*, 5:27–72, 2004.

[123] Gert R. G. Lanckriet, Nello Cristianini, Peter L. Bartlett, Laurent El Ghaoui, and Michael I. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72, 2004.

[124] Hugo Larochelle, Hugo Larochelle, Yoshua Bengio, Yoshua Bengio, Jerome Lourador, Jerome Lourador, Pascal Lamblin, and Pascal Lamblin. Exploring Strategies for Training Deep Neural Networks. *Journal of Machine Learning Research*, 10:1–40, 2009.

[125] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, (7553):436–444.

[126] Chien-Pang Lee and Yungho Leu. A novel hybrid feature selection method for microarray data analysis. *Applied Soft Computing*, 11(1):208–213, 2011.

[127] Michael C Lee, Lilla Boroczky, Kivilcim Sungur-Stasik, Aaron D Cann, Alain C Borczuk, Steven M Kawut, and Charles A Powell. Computer-aided diagnosis of pulmonary nodules using a two-step approach for feature selection and classifier ensemble construction. *Artificial intelligence in medicine*, 50(1):43–53, 2010.

[128] Ming-Chi Lee. Using support vector machine with a hybrid feature selection method to the stock trend prediction. *Expert Systems with Applications*, 36(8):10896–10904, 2009.

[129] Seung Ho Lee, Jae Young Choi, Konstantinos N Plataniotis, and Yong Man Ro. Color component feature selection in feature-level fusion based color face recognition. In *Fuzzy Systems (FUZZ), 2010 IEEE International Conference on*, pages 1–6. IEEE, 2010.

[130] Guy Lever, Tom Diethe, and John Shawe-Taylor. Data-dependent kernels in nearly-linear time. *arXiv preprint arXiv:1110.4416*, 2011.

[131] H. Liu and R. Setiono. Chi2: Feature selection and discretization of numeric attributes. In J.F. Vassilopoulos, editor, *Proceedings of the Seventh IEEE International Conference on Tools with Artificial Intelligence, November 5-8, 1995*, pages 388–391, Herndon, Virginia, 1995. IEEE Computer Society.

[132] Wei Liu, Buyue Qian, Jingyu Cui, and Jianzhuang Liu. Spectral kernel learning for semi-supervised classification. In *IJCAI*, pages 1150–1155, 2009.

[133] Xiao Liu, Jun Shi, and Qi Zhang. Tumor classification by deep polynomial network and multiple kernel learning on small ultrasound image dataset. In *Machine Learning in Medical Imaging*, pages 313–320. Springer, 2015.

[134] Yijuan Lu, Ira Cohen, Xiang Sean Zhou, and Qi Tian. Feature selection using principal feature analysis. In *Proceedings of the 15th international conference on Multimedia*, pages 301–304. ACM, 2007.

[135] James Martens. Deep learning via Hessian-free optimization. *27th International Conference on Machine Learning*, pages 735–742.

[136] Andreas Maurer. Bounds for linear multi-task learning. *The Journal of Machine Learning Research*, 7:117–139, 2006.

[137] Debasis Mazumdar, Soma Mitra, and Sushmita Mitra. Evolutionary-rough feature selection for face recognition. In *Transactions on rough sets XII*, pages 117–142. Springer, 2010.

[138] Donald G McLaren, Kristopher J Kosmatka, Terrance R Oakes, Christopher D Kroenke, Steven G Kohama, John a Matochik, Don K Ingram, and Sterling C Johnson. A population-average MRI-based atlas collection of the rhesus macaque. *NeuroImage*, 45(1):52–9, March 2009.

[139] Tom Mitchell, William Cohen, Estevam Hruschka, Partha Talukdar, Justin Betteridge, Andrew Carlson, Bhavana Dalvi Mishra, Matthew Gardner, Bryan Kisiel, Jayant Krishnamurthy, Ni Lao, Kathryn Mazaitis, Thahir Mohamed, Ndapa Nakashole, Emmanouil Platanios, Alan Ritter, Mehdi Samadi, Burr Settles, Richard Wang, Derry Wijaya, Abhinav Gupta, Xinlei Chen, Abulhair Saparov, Malcolm Greaves, and Joel Welling. Never-Ending Learning, 2015.

[140] Radhika Mittal, Aman Kansal, and Ranveer Chandra. Empowering developers to estimate app energy consumption. In*Proceedings of the 18th annual International Conference on Mobile Computing and Networking (MobiCom '12)*, pages 317–328, 2012.

[141] David Mizell. Using gravity to estimate accelerometer orientation. In *Proceedings of the 7th IEEE International Symposium on Wearable Computers (ISWC'03)*, page 252, 2003.

[142] Gregoire Montavon, Mikio Braun, and Klaus-Robert Müller. Kernel Analysis of Deep Networks. pages 2563–2581.

[143] João FC Mota, João MF Xavier, Pedro MQ Aguiar, and Markus Püschel. A proof of convergence for the alternating direction method of multipliers applied to polyhedral-constrained functions. *arXiv preprint arXiv:1112.2295*, 2011.

[144] K. Nakamura and E. Fisher. Segmentation of brain magnetic resonance images for measurement of gray matter atrophy in multiple sclerosis patients. *NeuroImage*, 44(3):769–776, 2009.

[145] Andrew Y Ng. On Feature selection: Learning with Exponentially many Irrelevant Features Training Examples. *Proc. 15th Interntional Conference on Machine Learning*, pages 404–412, 1998.

[146] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *International Conference on Machine Learning*, (2):1310–1318.

[147] Abhinav Pathak, Y. Charlie Hu, and Ming Zhang. Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with eprof. In *Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys '12)*, pages 29–42, 2012.

[148] Paul Pavlidis, Jason Weston, Jinsong Cai, and William Noble Grundy. Gene functional classification from heterogeneous data. In *Proceedings of the fifth annual international conference on Computational biology*, pages 249–255. ACM, 2001.

[149] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[150] Hanchuan Peng, Fuhui Long, and Chris Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(8):1226–1238, 2005.

[151] Susanna Pirttikangas, Kaori Fujinami, and Tatsuo Nakajima. Feature selection and activity recognition from wearable sensors. In HeeYong Youn, Minkoo Kim, and Hiroyuki Morikawa, editors, *Ubiquitous Computing Systems* , volume 4239 of *Lecture Notes in Computer Science*, pages 516–527. Springer Berlin Heidelberg, 2006.

[152] E. Poletti, E. Veronese, M. Calabrese, A. Bertoldo, and E. Grisan. Supervised classification of brain tissues through local multi-scale texture analysis by coupling dir and flair mr sequences. *Proceedings of SPIE*, 8314:83142T, 2012.

[153] Marcel Prastawa, Elizabeth Bullitt, and Guido Gerig. Synthetic ground truth for validation of brain tumor mri segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2005*, pages 26–33. Springer, 2005.

[154] Shibin Qiu and Terran Lane. A framework for multiple kernel support vector regression and its applications to sirna efficacy prediction. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, 6(2):190–199, 2009.

[155] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[156] Alexander Rakhlin, Ohad Shamir, and Karthik Sridharan. Making gradient descent optimal for strongly convex stochastic optimization. *arXiv preprint arXiv:1109.5647*, 2011.

[157] A. Rakotomamonjy, F. Bach, S. Canu, and Y. Grandvalet. SimpleMKL. *Journal of Machine Learning Research*, 9:2491–2521, 2008.

[158] Rabab M Ramadan and Rehab F Abdel-Kader. Face recognition using particle swarm optimization-based selected features. *International Journal of Signal Processing, Image Processing and Pattern Recognition*, 2(2):51–65, 2009.

[159] Ronald C Read and Derek G Corneil. The graph isomorphism disease. *Journal of Graph Theory*, 1(4):339–363, 1977.

[160] B. Remeseiro, V. Bolon-Canedo, D. Peteiro-Barral, A. Alonso-Betanzos, B. Guijarro-Berdinas, A. Mosquera, M.G. Penedo, and N. Sanchez-Marono. A methodology for improving tear film lipid layer classification. *Biomedical and Health Informatics, IEEE Journal of*, 18(4):1485–1493, 2014.

[161] Y. Saeys, I. Inza, and P. Larrañaga. A review of feature selection techniques in bioinformatics. *Bioinformatics*, 23(19):2507–2517, 2007.

[162] Juergen Schmidhuber. Deep Learning in Neural Networks: An Overview. pages 1–88.

[163] Isaac Jacob Schoenberg. Positive definite functions on spheres. *Duke Mathematical Journal*, 9(1):96–108, 1942.

[164] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *Artificial Neural Networks ICANN97*, pages 583–588. Springer, 1997.

[165] Toby Segaran and Jeff Hammerbacher. *Beautiful data: the stories behind elegant data solutions*. " O'Reilly Media, Inc.", 2009.

[166] Ahmed Serag, Paul Aljabar, Gareth Ball, Serena J Counsell, James P Boardman, Mary a Rutherford, a David Edwards, Joseph V Hajnal, and Daniel Rueckert. Construction of a consistent high-definition spatio-temporal atlas of the developing brain using adaptive kernel regression. *NeuroImage*, 59(3):2255–65, February 2012.

[167] Thomas Serre, Lior Wolf, and Tomaso Poggio. Object recognition with features inspired by visual cortex. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2:994–1000, 2005.

[168] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[169] Shai Shalev-Shwartz, Yoram Singer, and Andrew Y. Ng. Online and batch learning of pseudo-metrics. In *ICML*, 2004.

[170] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

[171] J Shawe-Taylor, P L Bartlett, Robert C Williamson, and M Anthony. Structural Risk Minimization over Data-Dependent Hierarchies. *IEEE Transactions on Information Theory*, 44(5):1926–1940, 1996.

[172] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[173] Feng Shi, Pew-Thian Yap, Yong Fan, John H Gilmore, Weili Lin, and Dinggang Shen. Construction of multi-region-multi-reference atlases for neonatal brain MRI segmentation. *NeuroImage*, 51(2):684–93, June 2010.

[174] Si Si, Cho-Jui Hsieh, and Inderjit Dhillon. Memory Efficient Kernel Approximation. *Proceedings of The 31st International Conference on Machine Learning*, pages 701–709.

[175] Pekka Siirtola and Juha Röning. Recognizing human activities user-independently on smartphones based on accelerometer data. *International Journal of Interactive Multimedia and Artificial Intelligence*, 1(5):38–45, 06/2012 2012.

[176] Daniel L Silver. The parallel transfer of task knowledge using dynamic learning rates based on a measure of relatedness. *Connection Science*, 8(2):277–294, 1996.

[177] Amit Singhal. Introducing the Knowledge Graph: things, not strings, 2012.

[178] S.M. Smith, M. Jenkinson, M.W. Woolrich, C.F. Beckmann, T.E.J. Behrens, H. Johansen-Berg, P.R. Bannister, M. De Luca, I. Drobnjak, D.E. Flitney, R.K. Niazy, J. Saunders, J. Vickers, Y. Zhang, N. De Stefano, J.M. Brady, and P.M. Matthews. Advances in functional and structural mr image analysis and implementation as fsl. *NeuroImage*, 23(SUPPL. 1):S208–S219, 2004. cited By (since 1996) 1935.

[179] Sören Sonnenburg, Gunnar Rätsch, Christin Schäfer, and Bernhard Schölkopf. Large Scale Multiple Kernel Learning. *Journal of Machine Learning Research*, pages 1531—-1565.

[180] Nathan Srebro. *Learning with Matrix Factorizations*. PhD thesis, Massachusetts Institute of Technology, 2004.

[181] Bo Sun, Di Zhang, Jun He, Lejun Yu, and Xuewen Wu. Multi-feature based robust face detection and coarse alignment method via multiple kernel learning. In *SPIE Security+ Defence*, pages 96520H–96520H. International Society for Optics and Photonics, 2015.

[182] Zhaonan Sun, Nawanol Ampornpunt, Manik Varma, and Svn Vishwanathan. Multiple kernel learning and the smo algorithm. In *Advances in neural information processing systems*, pages 2361–2369, 2010.

[183] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. *Jmlr W&Cp*, 28(2010):1139–1147, 2013.

[184] Yuchun Tang, Cornelius Hojatkashani, Ivo D Dinov, Bo Sun, Lingzhong Fan, Xiangtao Lin, Hengtao Qi, Xue Hua, Shuwei Liu, and Arthur W Toga. The construction of a Chinese MRI brain atlas: a morphometric comparison study between Chinese and Caucasian cohorts. *NeuroImage*, 51(1):33–41, May 2010.

[185] Sebastian Thrun and Lorien Pratt. *Learning to learn.* Springer Science & Business Media, 2012.

[186] Chih-Fong Tsai and Yu-Chieh Hsiao. Combining multiple feature selection methods for stock prediction: Union, intersection, and multi-intersection approaches. *Decision Support Systems*, 50(1):258–269, 2010.

[187] Harun Uğuz. A two-stage feature selection method for text categorization by using information gain, principal component analysis and genetic algorithm. *Knowledge-Based Systems*, 24(7):1024–1032, 2011.

[188] Vladimir Naumovich Vapnik and Vlamimir Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

[189] Manik Varma and Bodla Rakesh Babu. More generality in efficient multiple kernel learning. *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, pages 1–8.

[190] Alexander Vergara, Shankar Vembu, Tuba Ayhan, Margaret A. Ryan, Margie L. Homer, and Raman Huerta. Chemical gas sensor drift compensation using classifier ensembles, 2012.

[191] E. Veronese, E. Poletti, M. Calabrese, A. Bertoldo, and E. Grisan. Unsupervised segmentation of brain tissues using multiphase level sets on multiple mri sequences. In *Intelligent Systems and Control/742: Computational Bioscience*. ACTA Press, 2011.

[192] S V N Vishwanathan and Alexander J Smola. Fast Kernels for String and Tree Matching. In *NIPS*, pages 569–576, 2002.

[193] A. Vovk, R.W. Cox, J. Stare, D. Suput, and Z.S. Saad. Segmentation priors from local image properties: Without using bias field correction, location-based templates, or registration. *NeuroImage*, 2010. cited By (since 1996) 1.

[194] Nikil Wale, Ian Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3):347–375, 2008.

[195] Fei Wang and Jimeng Sun. Survey on distance metric learning and dimensionality reduction in data mining. *Data Mining and Knowledge Discovery*, pages 1–31, 2014.

[196] Tinghua Wang, Haihui Xie, Liyun Zhong, and Shengzhou Hu. A multiple kernel learning approach to text categorization. *Journal of Computational and Theoretical Nanoscience*, 12(9):2121–2126, 2015.

[197] Kilian Q. Weinberger and Lawrence K. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10:207–244, 2009.

[198] Owen S Weislow, Rebecca Kiser, Donald L Fine, John Bader, Robert H Shoemaker, and Michael R Boyd. New soluble-formazan assay for HIV-1 cytopathic effects: application to high-flux screening of synthetic and natural products for AIDS-antiviral activity. *Journal of the National Cancer Institute*, 81(8):577–586, 1989.

[199] Christopher Williams and Matthias Seeger. The effect of the input density distribution on kernel-based classifiers. In *Proceedings of the 17th International Conference on Machine Learning*, number EPFL-CONF-161323, pages 1159–1166, 2000.

[200] Juanying Xie and Chunxia Wang. Using support vector machines with a novel hybrid feature selection method for diagnosis of erythemato-squamous diseases. *Expert Systems with Applications*, 38(5):5809–5815, 2011.

[201] Xinxing Xu, Ivor W. Tsang, and Dong Xu. Soft margin multiple kernel learning. *IEEE Trans. Neural Netw. Learning Syst.*, 24(5):749–761, 2013.

[202] Zenglin Xu, Rong Jin, Haiqin Yang, Irwin King, and Michael R. Lyu. Simple and efficient multiple kernel learning by group lasso. In *ICML*, pages 1175–1182, 2010.

[203] Feng Yang and KZ Mao. Robust feature selection for microarray data based on multicriterion fusion. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 8(4):1080–1092, 2011.

[204] Jing Yang, Dengju Yao, Xiaojuan Zhan, and Xiaorong Zhan. Predicting disease risks using feature selection based on random forest and support vector machine. In *Bioinformatics Research and Applications*, pages 1–11. Springer, 2014.

[205] Jingjing Yang, Yonghong Tian, Ling-Yu Duan, Tiejun Huang, and Wen Gao. Group-sensitive multiple kernel learning for object recognition. *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, (Iccv):2838–52.

[206] L. Yu and H. Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *International Conference on Machine Learning*, volume 20, page 856, 2003.

[207] Shi Yu, Tillmann Falck, Anneleen Daemen, Leon-Charles Tranchevent, Johan Ak Suykens, Bart De Moor, and Yves Moreau. L2-norm multiple kernel learning and its application to biomedical data fusion. *BMC bioinformatics*, 11:309, 2010.

[208] Xiao-tong Yuan, Zhenzhen Wang, Jiankang Deng, Qingshan Liu, and Senior Member. Efficient χ 2 Kernel Linearization via Random Feature Maps. pages 1–6, 2015.

[209] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent Neural Network Regularization. *Icrl*, (2013):1–8.

[210] Zheng Zhao and Huan Liu. Searching for interacting features. In *IJCAI*, volume 7, pages 1156–1161, 2007.

[211] Zheng Alan Zhao and Huan Liu. *Spectral feature selection for data mining*. Chapman & Hall/CRC, 2011.

[212] Xiaojin Zhu and Andrew B. Goldberg. *Introduction to Semi-Supervised Learning* . Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2009.

[213] Alexander Zien and Cheng Soon Ong. Multiclass multiple kernel learning. In *Proceedings of the 24th international conference on Machine learning*, pages 1191–1198. ACM, 2007.