Learning Preferences for Large Scale Multi-label Problems

Ivano Lauriola[1,2] Mirko Polato[1] Alberto Lavelli[2] Fabio Rinaldi[2,3] Fabio Aiolli[1]

[1] University of Padova - Department of Mathematics
Via Trieste, 63, 35121 Padova - Italy

[2] Fondazione Bruno Kessler
Via Sommarive, 18, 38123 Trento - Italy

[3] University of Zurich - Institute of Computational Linguistics
Andreasstrasse 15, CH-8050 Zurich - Switzerland

ivano.lauriola@phd.unipd.it

Abstract. Despite that the majority of machine learning approaches aim to solve binary classification problems, several real-world applications require specialized algorithms able to handle many different classes, as in the case of single-label multi-class and multi-label classification problems. The Label Ranking framework is a generalization of the above mentioned settings, which aims to map instances from the input space to a total order over the set of possible labels. However, generally these algorithms are more complex than binary ones, and their application on large-scale datasets could be untractable.
The main contribution of this work is the proposal of a novel general online preference-based label ranking framework. The proposed framework is able to solve binary, multi-class, multi-label and ranking problems. A comparison with other baselines has been performed, showing effectiveness and efficiency in a real-world large-scale multi-label task.

Keywords: Preference Learning Machine, Multi-class, Multi-label, Big data, Large-scale

1 Introduction

Nowadays, the majority of Machine Learning techniques are able to solve binary classification problems, where the algorithms try to determine if a pattern belongs to either a positive $(+1)$ or a negative (-1) class. Despite that the binary classification setting is the most known, studied and used, there are several problems and real-world applications in which this approach is not suitable, as is the case of multi-class and multi-label models.

In the literature several mechanisms exist to extend the binary classification setting. The simplest approach is based on decomposition methods, such as the one-against-one and one-against-all [8] approaches. Basically, these methods decompose the original multi-class problem in several binary tasks. Then, these

binary problems are solved using binary classifiers and predictions are combined with a voting procedure. More complex approaches try to model a single multi-class/multi-label problem, as in the case of the Label Ranking framework based on preferences [15], which aims to learn a total order on the set of possible labels. However, these methods usually suffer from scalability issues with respect to the number of classes, making the original problem untractable when this number is large. Besides, due to the constant growth of the available data, a challenging goal of these algorithms is to solve these problems efficiently in terms of computational cost, and required resources.

Inspired by these motivations, this paper presents an extension of the Preference Learning Machine (PLM) [2], a general label ranking framework to learn preferences in binary, multi-class and multi-label setting. The proposed extension mainly includes an efficient and scalable learning procedure, based on the Voted Perceptron algorithm [7], and online learning capability.

The proposed approach has been compared with Neural Networks on a real-world multi-label application. The multi-label task consists of a large-scale semantic indexing of PubMed documents, based on the Medical Subject Headings (MeSH) thesaurus.

2 Notation and Background

In the (single-label) multi-class classification problem, the unique label associated to each pattern $\boldsymbol{x}$ from the input space $\mathcal{X} \subseteq \mathbb{R}^d$, is selected from a predefined set of labels $\Omega = \{\omega_1, \dots, \omega_m\}$, where m is the number of possible labels $m = |\Omega|$. A common example of multi-class problem is the digit recognition, where the goal is to find the true digit corresponding to a handwritten input [9]. Let us now consider the problem of associating keywords from a given set Ω to a textual document [3]. Differently from the previous case, the number of associated labels (keywords) can be more than 1, and each document might have a different number of keywords. Hence, the task is to learn a mapping from a document to a set of labels. These kinds of problems are referred to multi-label classification problems.
It is easy to see that the single-label multi-class problem is a generalization of the binary setting, where $m = 2$ and, in turn, the multi-label is a generalization of the single-label multi-class problem.

In all of these settings, the label set $\boldsymbol{y} \in \mathcal{Y} \subseteq \{+1, -1\}^m$ associated to each pattern $\boldsymbol{x} \in \mathcal{X}$ can be coded as a binary m-dimensional vector, where each element y_i is active $(+1)$ if and only if the label ω_i is assigned to the pattern $\boldsymbol{x}$. Based on this code, training examples can be kept into two matrices. Let $\boldsymbol{X} \in \mathbb{R}^{l \times d}$ be the training matrix, where d-dimensional vectors are arranged in l rows, and let $\boldsymbol{Y} \in \{+1, -1\}^{l \times m}$ be the corresponding label matrix, where rows contain the code of the training patterns. The notation $\boldsymbol{x}_i$ is also used to identify the i-th pattern.

Besides the concept of multi-label classification, the more general *multi-label ranking* has been introduced [4]. The multi-label ranking approach aims to predict the ranking of all labels instead of predicting only the set of relevant ones.

2.1 Related work

Motivated by the increasing number of new applications, such as automatic annotations of video, images and textual documents, the problem of learning from multi-label data is affecting a large part of the modern research. Recently, several different approaches have been developed aiming to solve multi-label problems [12, 14, 6]. It is possible to divide these methods into two categories [13]: adaption methods and problem transformation methods.
Adaption methods extend specific machine learning algorithms to handle multi-label data, as in the case of Neural Networks which use an extended back-propagation algorithm with dedicated error functions (see [11] for a detailed explanation).
Problem transformation methods, instead, are those algorithms which map the multi-label classification problem into one or more binary tasks. The most known problem transformation approach is the one-against-all decomposition method [8]. This method generates an ensemble of $m = |\Omega|$ binary classifiers. The i-th classifier is trained with all the examples of the i-th class as positive labels, and all the other examples as negative labels. When models are trained, there are m decision functions. In a ranking multi-label setting, these decision functions define the score for each label. Furthermore, in a single-label multi-class problem the predicted label is the one which achieves the highest score.

See [16, 15, 1] for detailed surveys of multi-label problems.

3 Working with Preferences

Several algorithms able to solve Label Ranking problems exist in the literature. Some of them are based on the concept of *preferences*, which define an ordering relation on labels and examples. Methods based on preferences try to find a ranking hypothesis $f_\Theta : \mathcal{X} \times \Omega \to \mathbb{R}$ with parameters Θ, which assigns for each label $\omega_i \in \Omega$ a score to a fixed pattern $\boldsymbol{x} \in \mathcal{X}$, $f_\Theta(\boldsymbol{x}, \omega_i)$.

These algorithms can be restricted to two particular cases: *learning instance preference* and *learning label preference* [5].
In the instance preference scenario, a preference relations is defined as a bipartite graph $g = (N, A)$, where $N \subseteq \mathcal{X} \times \Omega$ is the set of nodes and $A \subseteq N \times N$ is the set of arcs.
A node $n = (\boldsymbol{x}_i, \omega_j) \in N$ is a pair composed by an example and a label, and it is a positive node iff the label ω_j is positive for the example $\boldsymbol{x}_i$, otherwise n is a negative node. An arc $a = (n_s, n_e) \in A$ connects a starting (positive) node $n_s = (\boldsymbol{x}_i, \omega_j)$ to its ending (negative) node $n_e = (\boldsymbol{x}_k, \omega_q)$. The direction of the arc indicates that the starting node must be *preferred* over the ending node.

The margin of an arc $a = (n_s, n_e)$ is the difference between the application of the ranking function f_Θ on the starting and ending nodes,

$$\rho_A(a, \Theta) = f_\Theta(n_s) - f_\Theta(n_e) = f_\Theta(\boldsymbol{x}_i, \omega_j) - f_\Theta(\boldsymbol{x}_k, \omega_q).$$

An arc $a = (n_s, n_e)$ is consistent with the hypothesis f_Θ iff the assigned score to the node n_s is greater than the score assigned to the node n_e, $f_\Theta(n_s) > f_\Theta(n_e)$, thus the margin $\rho_A(a, \Theta) > 0$. The margin of a graph $g = (N, A)$ is the minimum margin of its arcs $\rho_G(g, \Theta) = \min_{a \in A} \rho_A(a, \Theta)$. Then, a graph is consistent with the hypothesis f_Θ iff its arcs are consistent, $\rho_G(g, \Theta) > 0$.

In the instance preference task instead, preferences are defined by considering a single example at a time. In this scenario, an arc $a \in A$ considers nodes with the same example, $a = (n_s, n_e)$, with $n_s = (\boldsymbol{x}_i, \omega_j)$ and $n_e = (\boldsymbol{x}_i, \omega_q)$

It is easy to see that the label preference scenario tries to separate simultaneously the whole set of examples with their positive nodes and the set of negative nodes. Thus, it is suitable for solving classification tasks. In the instance preference approach instead the algorithms try to optimize the inner ordering for each example.

Some examples of instance preference graphs for a 2-label classification problem are shown in Fig. 1, where for each example: a) there is only one fully connected graph which connects all positive labels to all negative ones; b) for each example there are two graphs which connect each positive label to all of the negatives; c) there is a graph for each pair of labels, the first positive and the second negative. The architecture of these graphs is a hyperparameter selected a priori. Note that for each graph structure, the number of total arcs is the same.

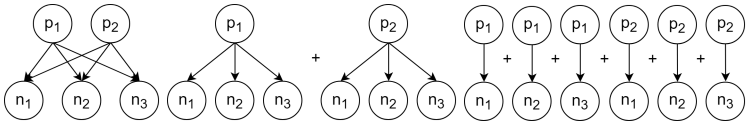

Fig. 1. Examples of preferences for 2-label classification. p_i are the positive labels and n_j the negative ones.

The last ingredient of a preference algorithm is a loss function $\mathcal{L}$ which penalizes the non-consistent preferences. A label ranking algorithm based on preferences tries to find the hypothesis $\hat{f}$ from the hypothesis space $\mathcal{F}$ which minimizes $\mathcal{L}$. Loss functions considered in this work are based on the margin of graphs:

$$\hat{f} = \arg \min_{f_\Theta \in \mathcal{F}} \sum_{g \in \mathcal{V}} \mathcal{L}(\rho_G(g, \Theta))$$

where $\mathcal{V}$ is the set of preference graphs.

3.1 Preference Learning Machine

The Preference Learning Machine (PLM) [2] belongs to the label preference setting. It is a general kernelized framework for solving multi-class and label ranking problems, by learning a function to map each example to a total order on the set of possible labels.

The PLM framework consists of a multivariate embedding $\boldsymbol{h} : \mathcal{X} \to \mathbb{R}^s$ parametrized by a set of s vectors $\boldsymbol{W}_k \in \mathbb{R}^d$, $k \in \{1, \ldots, s\}$ arranged in the matrix $\boldsymbol{W} \in \mathbb{R}^{s \times d}$. Thus, $\boldsymbol{h}(\boldsymbol{x}) = [h_1(\boldsymbol{x}), \ldots, h_s(\boldsymbol{x})] = [\langle \boldsymbol{W}_1, \boldsymbol{x} \rangle, \ldots, \langle \boldsymbol{W}_s, \boldsymbol{x} \rangle]$. Furthermore, let $\boldsymbol{M} \in \mathbb{R}^{m \times s}$ be the matrix containing the s-dimensional code for each label $\omega_i \in \Omega$.

The scoring function for a given example $\boldsymbol{x}$ and a given label ω_r can be computed as the dot product between the embedding and the code vector of ω_r, that is

$$f(\boldsymbol{x}, \omega_r) = \langle \boldsymbol{h}(\boldsymbol{x}), M_r \rangle = \sum_{k=1}^{s} M_{rk} \langle \boldsymbol{W}_k, \boldsymbol{x} \rangle.$$

The original PLM [2] considers a fixed m-dimensional orthogonal coding $\boldsymbol{M}$, defined as the $m \times m$ identity matrix. Authors also formulated the problem of learning the embedding $\boldsymbol{W}$ as a kernelized optimization problem.

4 The proposed extension

In the proposed setting, preferences consist of graphs with two nodes connected by a single arc. The first node is represented by an example with one of its positive labels, whereas the latter node is an (potentially different) example with one of its negative labels.

The main extension concerns the possibility of learning the Coding matrix $\boldsymbol{M}$, making the algorithm more expressive with respect to the original one. Two version of the algorithm are proposed in this work, which are the *EC-PLM* (Embedding-Coding PLM) and the *EP-PLM* (Embedding-PCA PLM).

The EC-PLM uses a pair of Voted Perceptron [7] algorithm to efficiently learn both, the Embedding $\boldsymbol{W}$ and the coding $\boldsymbol{M}$. Broadly speaking, the EC-PLM performs an alternate optimization procedure to learn its parameters. During each epoch, the algorithm fixes the Coding and optimizes the Embedding by means of a Voted Perceptron. Then, it fixes the Embedding while optimizes the Coding by using the same procedure. After each optimization, the Embedding $\boldsymbol{W}$ and the Coding $\boldsymbol{M}$ are rescaled with their Frobenius norm, $\boldsymbol{W} \leftarrow \frac{\boldsymbol{W}}{\sum_{ij} W_{ij}}$, $\boldsymbol{M} \leftarrow \frac{\boldsymbol{M}}{\sum_{ij} M_{ij}}$.

The training set used to learn the Embedding is composed by preferences. Let a be the arc of a preference graph which connect the starting node $(\boldsymbol{x}_i, \omega_j)$ with the ending node $(\boldsymbol{x}_k, \omega_q)$. The preference uses the same representation of the PLM, which consists of a $s \times d$ dimensional vector $\boldsymbol{z} = \big(\boldsymbol{M}_{\omega_j} \otimes \boldsymbol{x}_i\big) - \big(\boldsymbol{M}_{\omega_q} \otimes \boldsymbol{x}_k\big)$, where $\otimes$ denotes the *kron* product between vectors and $\boldsymbol{M}_{\omega_j}$, $\boldsymbol{M}_{\omega_q}$ are the codes of ω_j and ω_q. The dimensionality s of codes is a hyperparameter.

When the latter perceptron learns the coding matrix, preferences are defined as $\boldsymbol{z} = (y_s \otimes \langle \boldsymbol{W}, \boldsymbol{x}_i \rangle) - (y_e \otimes \langle \boldsymbol{W}, \boldsymbol{x}_j \rangle)$, where y_j is a 0 m-dimensional vector with an 1 at the j-th element. However, the algorithm requires an initialized code matrix at the first epoch, to learn the first embedding. The initial coding $\boldsymbol{M}$ contains random values.

Furthermore, a faster version of the PLM has been considered, dubbed *EP-PLM*, in which the coding $\boldsymbol{M}$ is computed by means of a Principal Component Analysis (PCA) procedure. Thus, the algorithm requires a single Voted Perceptron procedure.
Let $\boldsymbol{K_T}$ be the linear kernel between labels $\boldsymbol{K_T} = \boldsymbol{Y}\boldsymbol{Y}^\top$, which counts the number of common examples for each pair of labels. The kernel matrix is then decomposed as $\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$, where $\boldsymbol{U}$ is the matrix contains the eigenvectors, and $\boldsymbol{\Lambda}$ the diagonal matrix containing the eigenvalues. The Coding $\boldsymbol{M}$ is defined as $\boldsymbol{U_s}\boldsymbol{\Lambda_s}$, where $\boldsymbol{U_s}$ is the matrix contains the s eigenvectors associated to the top s eigenvalues. Note that the complexity of this approach mainly depends on the number of labels, and it can be applied on very large scale datasets.
The pseudo-code of the EC-PLM algorithm is shown in the Algorithm 1.

Algorithm 1: The Embedding-Coding Preference Learning Machine

Input:
s: the dimensionality of codes
t: the number of epochs
$\boldsymbol{X}$: the training matrix
$\boldsymbol{Y}$: the label matrix
Output:
$\boldsymbol{W}$: the embedding function
$\boldsymbol{M}$: the coding function

$\boldsymbol{W}^{(0)} \leftarrow \{0\}^{s \times d}$
$\boldsymbol{M}^{(0)} \leftarrow$ random $m \times s$ code matrix
for $i \in 1 \dots t$ **do**
  $\boldsymbol{W}^{(i)} \leftarrow Voted\_Perceptron(\boldsymbol{M}^{(i-1)})$
  $\boldsymbol{M}^{(i)} \leftarrow Voted\_Perceptron(\boldsymbol{W}^{(i)})$
end
return $\boldsymbol{W}^{(t)}, \boldsymbol{M}^{(t)}$

Due to the characteristics of the Voted Perceptron algorithm and its capability to work with one preference at a time, the EC-PLM can be easily used to work with on-line streams of examples and preferences.
On the other hand, the EP-PLM is able to learn the coding with millions of examples efficiently. Furthermore, on each epoch it uses a single Voted Perceptron to learn the Embedding. The complete procedure is very fast, especially if the input examples use a sparse representation.

5 Experimental assessment

In order to empirically evaluate the proposed method, it has been tested on a complex multi-label task, which consists of a large-scale online biomedical semantic indexing of PubMed documents based on the Medical Subject Headings (MeSH) [10]. The MeSH thesaurus is a controlled vocabulary produced by the National Library of Medicine (NLM), used for indexing and cataloging the biomedical literature in MEDLINE, that is the NLM bibliographic database containing 24 million journal articles.
The MeSH vocabulary consists of a hierarchy of tags. This work focused on the bottom layer of this hierarchy, which includes 28 333 descriptors or heading tags, that represent main topics or concepts in the biomedical literature.

In this setting, heading tags represent the set of all possible classes or labels, and the task is to find for each example a total order in this set.

5.1 Baselines

The proposed methods have been compared against a Multiple Layer Perceptron (MLP) which represents the same architecture used in the PLM. Let us consider a fully connected MLP with a d-dimensional input layer, which maps the input into a hidden s-dimensional layer by means of a dense $d \times s$ linear connection. Then, the hidden layer maps information on a $m = |\Omega|$ dimensional output layer by using a dense $s \times c$ linear connection. With this perspective, it is easy to show that the two mappings between layers correspond to the Embedding $\boldsymbol{W}$ and Coding $\boldsymbol{M}$ used in the PLM setting.
However, although the PLM can be mapped into a MLP and vice versa, the learning mechanisms used are quite different. The MLP uses a back-propagation procedure whereas the PLM tries to optimize each input preference.

Other baselines have been initially considered. These are the Support Vector Machine (SVM) with one-against-all multi-class strategy, and the original PLM. Anyhow, due to the dimensionality of the considered problem and the complexity of these methods, only the MLP has been used.

5.2 Empirical evaluation

A wide experimental setting has been used to compare the two versions of the algorithm, in terms of AUC score, computational cost and required resources.

At first, 20 000 abstracts have been randomly selected from the PubMed repository with their respective MeSH tags. Abstracts have been tokenized by considering spaces and punctuation, and stop-words have been removed. The stop-list is the one defined by the scikit-learn library. The global dictionary has been computed by considering only unigrams.
Then, the resulting dictionary has been reduced, by considering only the 100 000 most frequent terms. Finally, the Bag-Of-Words (BOW) feature vector has been computed on each input document. A test set has been preprocessed using the same pipeline, and it also includes 20 000 abstracts. To compute the coding

matrix $\boldsymbol{M}$ used in the EP-PLM version, a PCA over 10 million of PubMed documents has been used. The dimension s of codes has been fixed to 50. On each epoch, the Voted Perceptron procedure optimizes 2000 preferences randomly selected. Finally, the training subsampling covers 17071 different MeSH tags.

In order to understand properly the behavior and the empirical convergence of the two algorithms, a preliminary analysis has been performed, showing the micro and macro AUC measures while increasing the number of training epochs. Results are shown in the Fig. 2.
It is self-evident from the picture that the EP-PLM outperforms empirically the EC-PLM, even if it uses a fixed code matrix instead of learning dynamically it from data. Probably, this improvement is due to the fact that the EP-PLM uses 10 million of examples to learn the coding instead of 20 000 as is the case of EC-PLM. In terms of computational cost, the EC-PLM requires on average 132 seconds to complete a single epoch, whereas the EP-PLM required 95 minutes to compute the PCA, and 19 seconds per epoch. The experiments were carried out on an Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz.

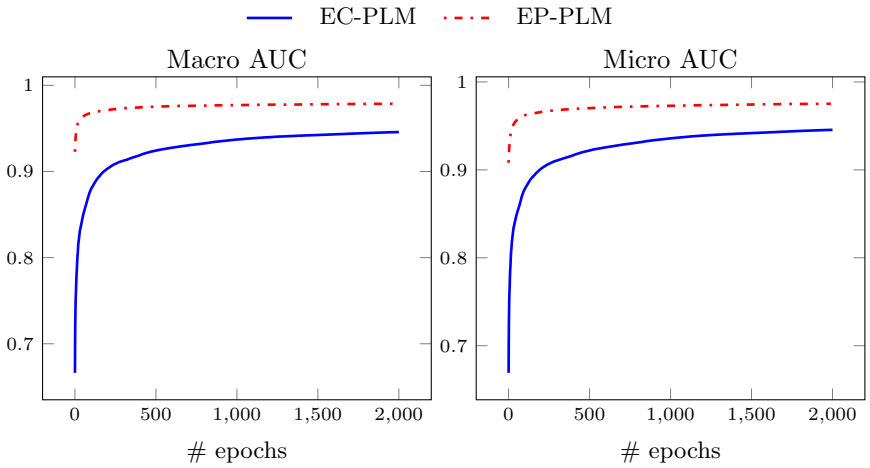

Fig. 2. Empirical convergence of the proposed algorithm.

Subsequently, the combination of several EC/EP-PLM models have been analyzed exploiting a bagging procedure, aiming to facilitate the application of these algorithms on large-scale problems. 10 different datasets have been extracted from the PubMed repository following the procedure mentioned at the beginning of this section, each with 20 000 training examples. Fig. 3 shows the empirical effectiveness of the algorithm while increasing the number of models in the case of EC-PLM and EP-PLM. Not surprisingly, the bagging procedure has a strong impact on the EC-PLM setting, in which each model uses only 20 000 examples for both the Embedding and the Coding. The EP-PLM also increases the AUC scores while the number of models increases.

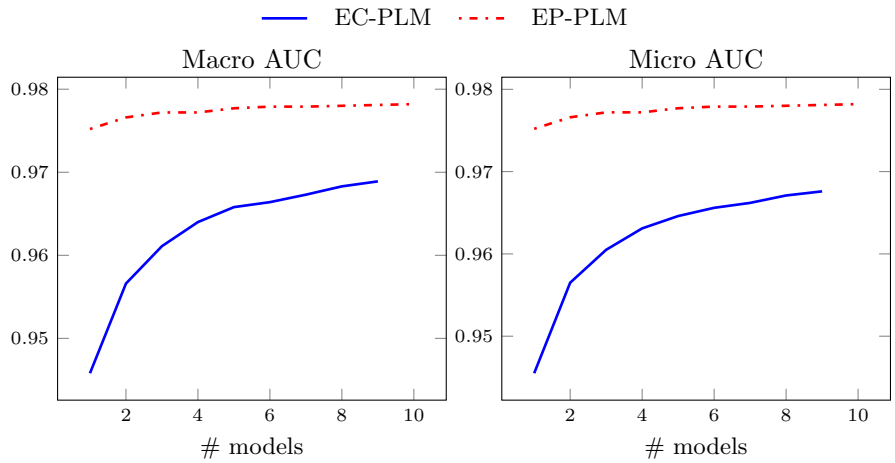

Fig. 3. Micro and Macro AUC scores while increasing the number of combined models.

Finally, a comparison against the Multiple Layer Perceptron has been performed. Fig. 4 shows Micro and Macro AUC scores of the EC-PLM and EP-PLM against the MLP with linear and sigmoid activation functions, while increasing the dimension $s \in \{25, 50, 100, 200\}$ of the codes and the hidden layer. This experiment shows that the proposed methodologies outperform a MLP with the same inner structure of the PLM. Moreover, the value of s affects significantly the EC/EP-PLM and the linear MLP in particular.

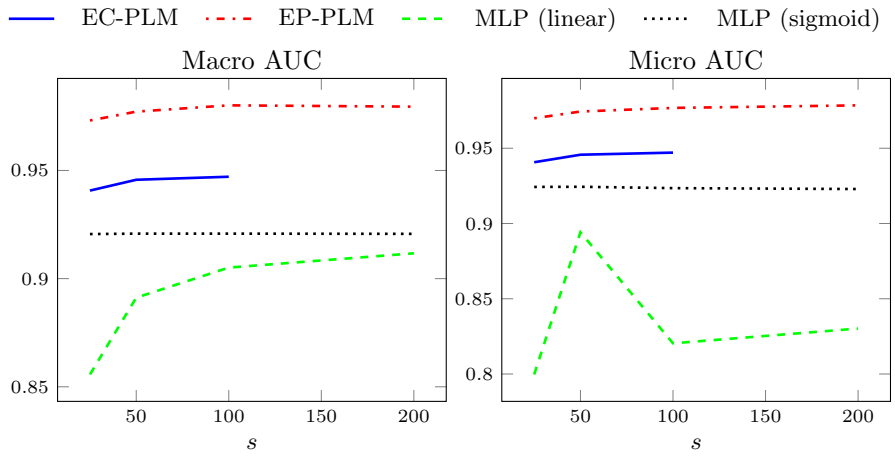

Fig. 4. Micro and Macro AUC scores of EC/EP-PLM and the MLP while increasing the dimension s of the middle space.

6 Conclusion

We have proposed a general framework for on-line preference-based label ranking that can be applied to binary, multi-class, multi-label and ranking problems. Two different versions of the algorithm have been discussed and analyzed. The first focuses on the efficiency whereas the latter is an effective on-line learner. A comparison with some baselines has shown its effectiveness and efficiency in a real-world large-scale multi-label task.

References

1. Aiolli, F.: Large margin multiclass learning: models and algorithms. Ph.D. thesis, PhD thesis, Dept. of Computer Science, University of Pisa, 2004 (2004)
2. Aiolli, F., Sperduti, A.: Learning preferences for multiclass problems. In: Advances in neural information processing systems. pp. 17–24 (2005)
3. Allan, J.: Topic detection and tracking: event-based information organization, vol. 12. Springer Science & Business Media (2012)
4. Brinker, K., Fürnkranz, J., Hüllermeier, E.: A unified model for multilabel classification and ranking. In: Proceedings of the 2006 conference on ECAI 2006: 17th European Conference on Artificial Intelligence August 29–September 1, 2006, Riva del Garda, Italy. pp. 489–493. IOS Press (2006)
5. Chu, W., Ghahramani, Z.: Preference learning with gaussian processes. In: Proceedings of the 22nd international conference on Machine learning. pp. 137–144. ACM (2005)
6. Dembczynski, K., Cheng, W., Hüllermeier, E.: Bayes optimal multilabel classification via probabilistic classifier chains. In: ICML. vol. 10, pp. 279–286 (2010)
7. Freund, Y., Schapire, R.E.: Large margin classification using the perceptron algorithm. Machine learning 37(3), 277–296 (1999)
8. Hsu, C.W., Lin, C.J.: A comparison of methods for multiclass support vector machines. IEEE Transactions on Neural Networks 13(2), 415–425 (2002)
9. Meier, U., Ciresan, D.C., Gambardella, L.M., Schmidhuber, J.: Better digit recognition with a committee of simple neural nets. In: Document Analysis and Recognition (ICDAR), 2011 International Conference on. pp. 1250–1254. IEEE (2011)
10. Nentidis, A., Bougiatiotis, K., Krithara, A., Paliouras, G., Kakadiaris, I.: Results of the fifth edition of the BioASQ challenge. In: BioNLP 2017. pp. 48–57. Association for Computational Linguistics, Vancouver, Canada, (August 2017)
11. Ou, G., Murphey, Y.L.: Multi-class pattern classification using neural networks. Pattern Recognition 40(1), 4–18 (2007)
12. Read, J., Pfahringer, B., Holmes, G., Frank, E.: Classifier chains for multi-label classification. Machine learning 85(3), 333 (2011)
13. Tsoumakas, G., Katakis, I.: Multi-label classification: An overview. International Journal of Data Warehousing and Mining 3(3) (2006)
14. Tsoumakas, G., Vlahavas, I.: Random k-labelsets: An ensemble method for multilabel classification. In: European conference on machine learning. pp. 406–417. Springer (2007)
15. Vembu, S., Gärtner, T.: Label ranking algorithms: A survey. In: Preference learning, pp. 45–64. Springer (2010)
16. Zhang, M.L., Zhou, Z.H.: A review on multi-label learning algorithms. IEEE transactions on knowledge and data engineering 26(8), 1819–1837 (2014)