Label Filters for Large Scale Multilabel Classification

Alexandru Niculescu-Mizil
NEC Laboratories America
Princeton, NJ, USA
alex@nec-labs.com

Ehsan Abbasnejad
Australian National University
eabbasnejad@gmail.com

Abstract

When assigning labels to a test instance, most multilabel and multiclass classifiers systematically evaluate every single label to decide whether it is relevant or not. This linear scan over labels becomes prohibitive when the number of labels is very large. To alleviate this problem we propose a two step approach where computationally efficient *label filters* pre-select a small set of candidate labels before the base multiclass or multilabel classifier is applied. The label filters select candidate labels by projecting a test instance on a filtering line, and retaining only the labels that have training instances in the vicinity of this projection. The filter parameters are learned directly from data by solving a constraint optimization problem, and are independent of the base multilabel classifier. The proposed label filters can be used in conjunction with any multiclass or multilabel classifier that requires a linear scan over the labels, and speed up prediction by orders of magnitude without significant impact on performance.

1 Introduction

With the advent of big data, classifiers can make more and more fine grained distinctions between various classes, leading to the rise of classification problems with very large numbers of labels. Data sets with tens and hundreds of thousands labels are already becoming standard benchmarks in domains like object recognition and text classification (Deng et al., 2009; Partalas et al., 2015), and multilabel problems with millions of labels have recently been tackled in the literature (Weston et al., 2013; Prabhu and Varma, 2014). As more and more data becomes available, the number of problems with huge label sets, as well as the number of labels per problem is going to increase.

Proceedings of the 20^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2017, Fort Lauderdale, Florida, USA. JMLR: W&CP volume 54. Copyright 2017 by the author(s).

One consequence of the explosion in the number of labels is a significant increase in the test-time (production time) computational burden. Most approaches to multiclass and multilabel classification, such as the very popular one-vs-all scheme or the Crammer-Singer multiclass SVM (Crammer and Singer, 2002), have to systematically evaluate the match between each label and the test instance in order to make a prediction, leading to a test-time complexity linear in the number of labels. As the number of labels grows this systematic evaluation of all labels becomes prohibitive, especially for applications that require real-time response and/or have limited computational resources.

Imagine being asked to assign tags to this paper based on its abstract. You might consider the appropriateness of tags like “extreme classification”, “multilabel”, “optimization”, “large-scale learning” or “big data”. There are, however, a vast number of possible tags, like “gaussian processes”, “social networks”, or “cook book” that would probably not even cross your mind. If you were to go through a long list of tags to find the relevant ones, akin to how most multilabel classifiers work, the task would be rather tedious. But, by quickly focusing on a small number of potentially relevant tags, the problem became much more tractable.

In line with this intuition, we explore a two step approach to multilabel classification: for each test instance, first reduce the label set by pre-selecting a small, instance dependent, set of candidate labels, then assign labels from only this restricted candidate set using a classic multilabel classifier. This two step approach allows the multilabel classifier to focus on a small number of candidate labels rather than the entire label set, speeding up classification and reducing the computational footprint.

In this paper we introduce *label filters*, a novel technique for pre-selecting a set of candidate labels. Given a test instance, the goal of a label filter is to return a set of candidate labels that 1) contains most of the true labels of the given instance (to avoid degrading performance by missing relevant labels); and 2) it is as small as possible (to reduce the computational cost of the subsequent multilabel classifier). We propose a principled optimization framework for training label filters that expresses these two requirements as a mixed integer problem. The learned label filters are compact, efficient, and highly effective, increasing prediction speed up to 500 times without a significant impact on performance. Moreover, the label filters can be used in conjunctions with any multilabel classifier that requires a linear scan over labels, and can be trained independently of the base multilabel classifier, eliminating the need for joint hyper-parameter selection or retraining when the base multilabel classifier changes.

The flexibility to use any multilabel classifier and to train the label filter independently of the base classifier means that the use of label filters does not alter the training of the base multilabel classifier and, in particular, does not reduce the computational cost in the training phase[1]. While reducing the computational burden of training large scale multilabel classifiers is an important and interesting challenge, there are numerous applications where the constraints on computational resources and response time are much more stringent in production than in the offline training phase. For example, in applications such as interactive tag recommendation or real-time bidding, training can be done offline over multiple days or weeks, but a real-time response is required in production; or in high volume streaming problems such as ad placement the volume of data processed in production is much larger than at training time; or in applications where classifiers must be deployed on restricted hardware such as laptops, smart phones or satellites, there often is no such restriction at training time and the models can often be trained on powerful clusters. In all these types of applications, reducing the computational burden at test time while maintaining top performance is more important than speeding up the training phase.

2 Related Work

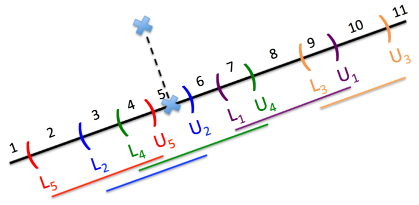
Several approaches have been proposed to reduce the test-time computational burden of multiclass or multilabel classifiers: coarse to fine classification using label hierarchies that are either predefined (Liu et al., 2005), or inferred from data (Bengio et al., 2010; Deng et al., 2011; Gao and Koller, 2011); Error Correcting Output Codes and label embeddings (Ji et al., 2008; Hsu et al., 2009; Chen and Lin, 2012; Cissé et al., 2012; Bi and Kwok, 2013; Cissé et al., 2013); Error Correcting Tournaments (Beygelzimer et al., 2009); input partitioning (Weston et al., 2013); ensembles of decision trees (Agrawal et al., 2013; Prabhu and Varma, 2014); and fast nearest neighbor (Indyk and Motwani, 1998; Bentley, 1975). Below we briefly review the approaches that are most related to our work.

Most similar to the technique proposed in this paper, Label Partitioning for Sublinear Ranking (LPSR) (Weston et al., 2013) is also a two-step approach based on pre-selecting a small set of candidate labels prior to applying a multilabel base classifier. LPSR clusters the training instances, then assigns a fixed number of possible labels to each cluster. The assignment of labels to clusters is optimized to increase the overall performance, based on the labels of the training instances in that cluster and on the predictions of the base multiclass or multilabel technique. At test time an instance is first assigned to one of the clusters, then the multiclass or multilabel classifier is applied using only the labels attached to this cluster.

A number of approaches are based on learning ensembles of decision trees using various optimization criteria to generate the splits at the inner nodes. The Multi-Label Random Forests (Agrawal et al., 2013) use a multilabel version of the Gini Index as the splitting criterion. FastXML (Prabhu and Varma, 2014) learns a hyperplane at each inner node by optimizing an nDCG based ranking loss and uses this hyperplane as a splitting function. The decision tree based techniques have the advantage that they can be trained more efficiently, but their performance is sometimes inferior to more expensive techniques such as OvA SVMs.

A related problem is approximate Maximum Inner Product Search (MIPS). When the base multilabel classifier is linear, finding the top-k predicted labels reduces to finding the k weight vectors that have the largest inner product with the test instance. In the context of the two step approach discussed in this paper, approximate MIPS can be used to quickly find a set of candidate labels that would be further refined by calculating the exact inner-products. Two main types of techniques have been proposed to solving approximate MIPS problems: tree-based (Ram and Gray, 2012; Bachrach et al., 2014), and hashing based (Shrivastava and Li, 2014, 2015; Vijayanarasimhan et al., 2014). The label filter approach we propose could be thought of as learning a supervised hashing functions that are optimized to maximize collisions with relevant labels and minimize collisions with irrelevant labels.

[1]We discuss in section 5 potential avenues to speed up the training of multilabel classifiers, but we do not explore them in this paper



Segment	Active Labels
1	
2	5
3	5,2
4	5,2,4
5	2,4
6	4
7	4,1
8	1
9	1,3
10	3
11	

Figure 1: A label filter. A test example is projected on the filter line. Labels 2 and 4 are selected as candidate labels. Labels 1, 3 and 5 are filtered out because the projection falls outside their corresponding intervals. Pre-computing the active labels for each segment of the filtering line allows retrieving candidate labels in time logarithmic in K.

3 The Label Filter

The purpose of a label filter is to speed up the labeling process at test (production) time by restricting the set of labels a multiclass or multilabel classifier must evaluate. Given a test example $x \in \mathbb{R}^d$ and a set of labels $\mathcal{Y}$, a label filter outputs a subset $\mathbf{Y}_x \subset \mathcal{Y}$ as candidate or active labels. Only labels in $\mathbf{Y}_x$ are further evaluated by the base classifier, with all labels not in $\mathbf{Y}_x$ being deemed irrelevant for the given example and excluded from further consideration. For the rest of the paper we assume $\mathcal{Y} = \{1, ..., K\}$.

The basic label filter consists of a *filtering line* parametrized by $w \in \mathbb{R}^d$, and one *active region* or *active interval* $[L_k, U_k)$ on this line for each label $k \in \{1, ..., K\}$. Figure 1 shows an example of a label filter with five classes.

Given a test example x, the label filter calculates the projection $x_p = \langle w, x \rangle$ on the filtering line and selects all labels whose active region contains x_p. That is, all labels k with $L_k \leq x_p < U_k$ are retained as candidate labels, and all labels k' with $x_p < L_{k'}$ or $x_p \geq U_{k'}$ are filtered out and considered irrelevant for x.[2]

A naive implementation would incur a cost of $O(d+K)$ for the projection followed by a linear scan over the labels to check whether the projection x_p falls within each label's active interval or not. Note that, while the complexity is still linear in the number of labels, it is significantly better than the multiplicative complexity $O(d \cdot K)$ of typical linear multilabel classifiers[3].

[2]The projection of a point on a line is given by $\langle w, x \rangle / \|w\|$. In this paper we absorb $\|w\|$ in L_k and U_k.

[3]If the data or model is sparse, d is replaced by the number of non-zeros.

The dependence on K can be further improved by pre-computing the set of active labels (labels to be retained by the filter) at every possible projection point on the filtering line. The end points $\{L_k\}_{1:K}$ and $\{U_k\}_{1:K}$ of all intervals split the filtering line into $2K+1$ segments, and the set of active labels remains constant within each segment (i.e., all test examples whose projection falls within a segment will have the same set of candidate labels selected by the filter. See Figure 1.). The set of active labels in each segment differs by exactly one label from that of neighboring segments, and can be pre-computed in one pass and stored. Now, applying the filter to a test instance costs only $O\left(d + \log(2K)\right)$ for the projection followed by a binary search to determine the segment where the projection falls.

3.1 Learning the Label Filter Parameters

For a label filter to be effective, filter parameters w, L_k, and U_k must be set such that 1) most instances are projected within the intervals corresponding to their true labels in order to retain as many true labels as possible in the candidate set; and 2) the overlap among intervals for different labels is minimized in order to keep the set of candidate labels small.

Given a training set $(x_i, y_i)_{i=1..n}$ with $y_i \subset \{1, .., K\}$ representing the set of labels assigned to example x_i (the cardinally of y_i is 1 for multiclass problems), the filter parameters w, L_k, and U_k can be estimated by the following optimization problem that directly encodes the desired properties:

$$\min_{\substack{w,L,U,\\ s,\xi}} \|w\|^2 + C_1 \sum_{i=1}^{n} \sum_{k \in y_i} \left(\xi_{ik}^L + \xi_{ik}^U\right) + \qquad (1)$$

$$+ \frac{C_2}{K} \sum_{i=1}^{n} \sum_{k' \notin y_i} \left(\xi_{ik'}^L + \xi_{ik'}^U\right)$$

Such that:

For all $i, k \in y_i$:

$$\begin{aligned} &L_k - \langle w, x_i \rangle + 1 < \xi_{ik}^L \\ &\langle w, x_i \rangle - U_k + 1 < \xi_{ik}^U \end{aligned} \qquad (2)$$

For all $i, k' \notin y_i$:

$$\begin{aligned} &s_{ik'} \left(\langle w, x_i \rangle - L_{k'} + 1\right) < \xi_{ik'}^L \\ &\left(1 - s_{ik'}\right)\left(U_{k'} - \langle w, x_i \rangle + 1\right) < \xi_{ik'}^U \end{aligned} \qquad (3)$$

For all i, k:

$$s_{ik} \in \{0, 1\};\ \xi_{ik}^L \geq 0;\ \xi_{ik}^U \geq 0$$

Intuitively, the constraints (2) require that the projection of each training example falls within the active interval(s) of its label(s) while the constraints (3) require that the projections of a training example falls outside the active intervals of classes not in its label set. That is, the projection of example i should fall either on the left (when $s_{ik'} = 1$) or on the right (when $s_{ik'} = 0$) of the interval corresponding to class $k' \notin y_i$. Slack variables ξ_{ik} allow some constraints to be violated, at a cost. C_1 weights the cost of an example falling outside the interval of a class in its label set, and C_2 weights the cost of an example falling within the interval of a class not in its label set. Since there are usually $O(n \cdot K)$ constraints of type (3) and only $O(n)$ constraints of type (2) C_2 is divided by K to balance the two costs. $\|w\|$ controls the margin between the projection of a training instance and the endpoints L_k and U_k of the active intervals. Lower values of $\|w\|$ lead to wider margins, improving generalization.

While the optimization problem above encodes the intuition that examples should be projected within the intervals corresponding to their labels and outside the intervals corresponding to other labels, it has one shortcoming: two examples i and j having the same label k could have $s_{ik'} \neq s_{jk'}$. This may lead to local minima where the interval for label k' is nested within the interval for label k increasing the overlap among intervals and reducing the performance of the filter. To alleviate this problem, we instead solve the following optimization problem:

$$\min_{\substack{w,L,U,\\ \Pi,\xi}} \|w\|^2 + C_1 \sum_{i=1}^{n} \sum_{k \in y_i} \left(\xi_{ik}^L + \xi_{ik}^U\right) + \qquad (4)$$

$$+ \frac{C_2}{K} \sum_{i=1}^{n} \sum_{k \in y_i} \sum_{k' \notin y_i} \xi_{ikk'}$$

Such that:

For all $i, k \in y_i$:

$$\begin{aligned} L_k - \langle w, x_i \rangle + 1 &< \xi_{ik}^L \\ \langle w, x_i \rangle - U_k + 1 &< \xi_{ik}^U \end{aligned} \qquad (5)$$

For all $i, k \in y_i, k' \notin y_i$:

$$\begin{aligned} \langle w, x_i \rangle - L_{k'} + 1 &< \xi_{ikk'} \text{ if } \Pi(k) < \Pi(k') \\ U_{k'} - \langle w, x_i \rangle + 1 &< \xi_{ikk'} \text{ if } \Pi(k) > \Pi(k') \end{aligned} \qquad (6)$$

For all $i, k \in y_i, k' \notin y_i$:

$$\xi_{ik}^L \geq 0;\ \xi_{ik}^U \geq 0;\ \xi_{ikk'} \geq 0$$

where Π is an ordering over the K labels. Intuitively, the constraints (6) require the projections of *all* examples with some label k to fall to the right of intervals corresponding to labels lower than k in the ordering Π, and to the left of intervals corresponding to labels higher than k in the ordering Π.

While the optimization problem (4) seems difficult, a local minima can be found via alternating minimization: optimize w, L, U with Π fixed, then optimize Π with w, L, U fixed. With Π fixed, the minimization over w, L, U is a convex problem with linear constraints and can be solved via any convex optimization technique. Given that the types of problems that require label filtering will tend to have a large number of training instances, we use averaged stochastic projected subgradient descent in our implementation. Also, rather than fully optimizing w, L, and U, we only perform a fixed number of gradient descent iterations before updating the ordering Π. Each gradient descent iteration takes $O(d + K)$ time. If K is very large, the gradient computation can be sped up by sampling uniformly at random a subset of $K' \ll K$ of the constraints (6) to enforce at each gradient iteration and ignoring the rest.

If both w and Π are fixed, the optimal L and U can found in $O(n \cdot d + n \cdot \log(n) + n \cdot K)$ time. Essentially, with w an Π fixed, problem (4) decomposes into independent optimization problems for each L_k and U_k with convex piece-wise linear objectives and can be minimized via a linear scan through the data. This optimization is too computationally expensive to apply after every update of w, so we only use it at the beginning of the optimization to initialize L and U, and at the end of the optimization to get the final L and U values.

To optimize Π with w, L and U fixed we use the following heuristic that takes $O(n \cdot d + K \cdot \log(K))$ time: project all the training instances that have label k on the line determined by the current w and calculate the mean of these projections. Then order the labels by these mean projected values. While we can not guarantee that this heuristic will always decrease the objective function or that it will lead to convergence to a local minima of (4), it seems to perform well in practice. If guaranteed convergence to a local minima is desired, then Π can be treated as a set of $K^2/2$ pairwise constraints between labels rather than a total ordering and the optimal Π given w, L and U can be calculated in $O(n \cdot d + K^2)$ time. For the experiments in this paper we use the total ordering based approach.

3.2 Multiple Filters

A single label filter is not expressive enough reliably return a very small candidate set without missing true labels. As a result, the speedup obtained using a single filter is rather modest (a factor of 2 to 5 speedup in our experiments). Significantly larger speedups can be obtained by taking the intersection of the candidate sets returned by multiple, diverse, label filters.

The more diverse the label filters, the smaller the intersection of their returned candidate sets will be and the

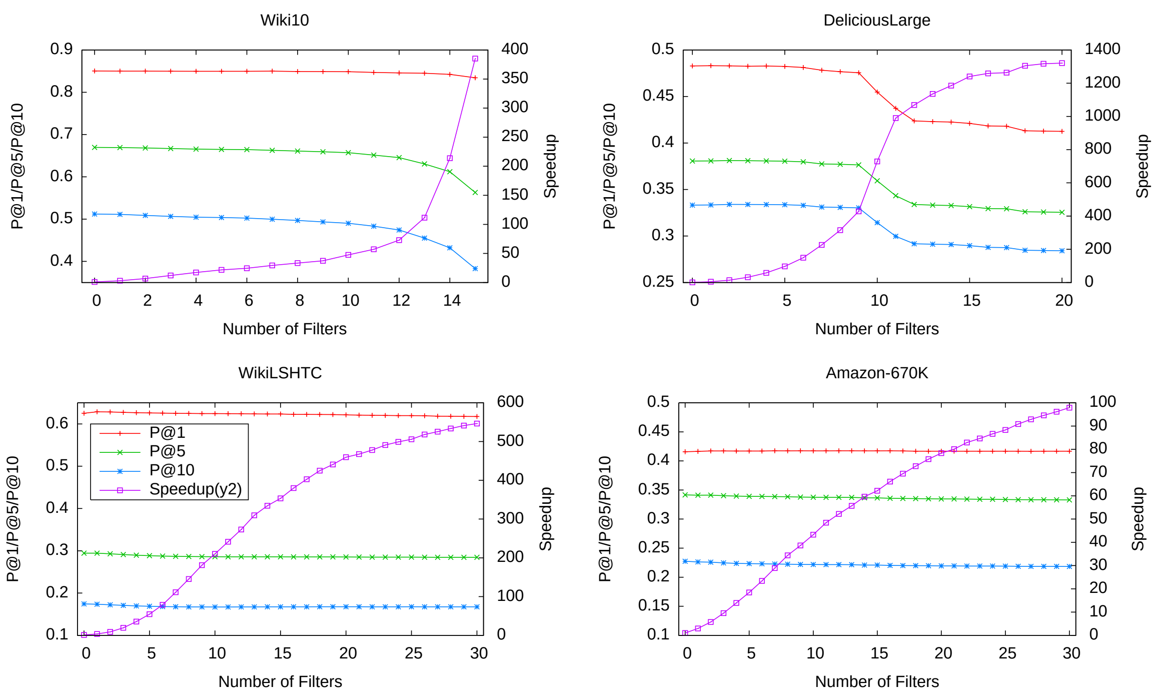


Figure 2: Performance (y1 axis) and speedup (y2 axis) as a function of the number of label filters used.

larger the speedup. To promote diversity in the label filters, we train them in a sequential manner, and encourage each filter to focus only on the labels that have not been eliminated by previous filters. Specifically, if a previous filter eliminated label k' for a training example i (i.e., the projection of example x_i on w falls outside the interval $[L_{k'}, U_{k'})$ for some previous filter), then the constraints (6) that involve the example x_i and the label k' are eliminated when training the current filter. The hyper-parameter for the current filter C_2 is increased to compensate for the removed constraints.

4 Experimental Results

We evaluate the performance of label filters on four large scale multilabel classification datasets from the Extreme Classification Repository[4]: Wiki10 (Zubiaga, 2009) with 30K labels, DeliciousLarge (Wetzker et al., 2008) with 200K labels, WikiLSHTC (Partalas et al., 2015) with 325K labels, and Amazon-670K (Leskovec and Krevl, 2014) with 670K labels. We use the same train-test splits as in (Bhatia et al., 2015) and standard data pre-processing (rare features appearing in less than 10 documents are eliminated[5] and instances are normalized to unit norm).

[4]`https://manikvarma.github.io/downloads/XC/XMLRepository.html`

[5]Using all the features does not have a significant impact on performance

Since the focus is on evaluating the potential of label filters to reduce the test-time computational burden, rather than on obtaining the best possible performance, we opt for using simple base multilabel classifiers: Naive Bayes (NB) and one vs. all (OvA) linear SVMs. Using 10% of the training data as a validation set we select the Laplacian smoothing constant for NB, and the C hyper-parameter as well as whether or not to use an intercept for SVMs. Following Babbar and Schölkopf (2017) SVM weights smaller than 0.01 are set to zero, significantly reducing the memory requirements. We note, however, that the label filters can be used in conjunction with any multilabel classifier, so better performance can be obtained by replacing the OvA SVMs or Naive Bayes with more advanced multilabel classifiers, and/or by more careful hyper-parameter selection and data pre-processing.

Following previous work on large scale multilabel classification (Weston et al., 2013; Prabhu and Varma, 2014; Bhatia et al., 2015) we use precision at k (P@k) as the evaluation metric. Precision at k is defined as the fraction of true labels among the top k predictions made by the classifier. In order to provide results that are independent of hardware and implementation details, we use the number of vector-vector multiplications as a measure for test-time computation.

We first look at how effective label filters are at reducing the computational burden without significantly degrading the performance of the base classifier. To

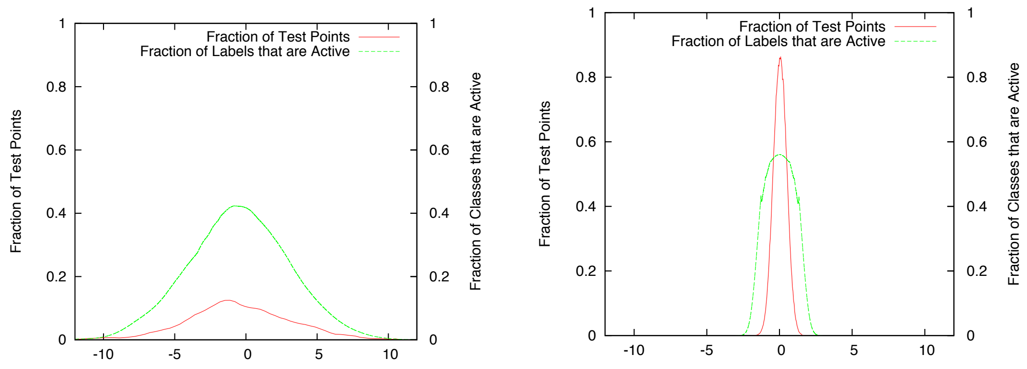


Figure 3: Fraction of test examples projected at each point on the filtering line (y1axis), and fraction of labels that are active at each point on the filtering line(y2axis). X-axis is the filtering line. Left: both filtering line and active intervals are optimized. Right: only active intervals are optimized, filtering line is random.

this end, we select "conservative" label filter hyperparameters (C_1, C_2, and the number of filters) that provide the highest speedup while lowering P@1, P@5 and P@10 by at most one percent on the validation set. The results depicted in Table 1 show that label filters (LF) are able to speed up classification by a factor of 33 on Wiki10, 428 on DeliciousLarge, 547 on WikiLSHTC and 98 on Amazon-670K, all with minimal impact on performance. This translates to eliminating, on average, 97%, 99.77% , 99.82%, and 98.97% of labels for each test example. Remarkably, very few label filters are needed to achieve this speedup: 8 for Wiki10, 9 for DeliciousLarge and 30 for WikiLSHTC and Amazon-670K. It is worth noting that the label filters remain the same irrespective of whether an SVM or Naive Bayes is used as a base classifier.

Figure 2 shows performance and prediction speedup as a function of the number of label filters. As expected, a single label filter only provides a modest reduction in prediction time, speeding up prediction by a factor of 2 to 5. With each additional label filter the prediction speed increases, at first without a significant loss in performance. On two problems, however, when too many filters are used the candidate label sets become excessively small and many true labels are wrongly eliminated, degrading performance significantly.

To gauge the importance of adapting the filtering lines w to the data, we experimented with using random filtering lines and only optimizing L and U to set the active regions for each class. Figure 3 shows the number of test examples projected, and the fraction of labels active, at each point on an optimized filtering line (left sub-figure) and a random filtering line of the same norm (right sub-figure). In both sub-figures the filtering line is the x-axis. When the filtering line is learned from data, the test examples are spread in a wider range, and fewer labels are active at each point on the filtering line. This label filter will eliminate more than half the labels for every test example, and significantly more in many cases. In contrast, if the filtering line is random, the projections of the test examples are concentrated around 0, in a region where the random label filter is only able to eliminate about 42% of the labels.

Figure 4 gives insight into what the first label filter has learned for the Wiki10 dataset. It shows the most frequent labels in different regions of the filtering line. To generate the figure, test examples are projected on the filtering line, and split into four quantiles. The left-most tag-cloud in the figure shows the 100 most frequent labels of the left-most 25% test examples. The next tag-cloud corresponds to the next 25% of the test examples, and so on. The label filter roughly aligns the instances from IT (linux, hardware, mac) to science (chemistry, physics, space) to art and culture (artist, film, poetry). Many labels are localized to small regions on the filtering line (e.g. algorithm, linux, chemistry, space, animals, nature, christianity, war), allowing the the label filer to eliminate them from consideration for all instances projected outside these regions.

We also compare the label filtering approach to state of the art multilabel classification techniques designed to handle large numbers of labels: FastXML (Prabhu and Varma, 2014), LPSR (Weston et al., 2013), and SLECC (Bhatia et al., 2015). FastXML (Prabhu and Varma, 2014) is the leading tree-based technique and has been shown to have fast prediction time and competitive performance. To compare the label filtering approach with FastXML on an equal footing, we select filter parameters that yield a speedup that is similar to or better than FastXML (denoted as LF aggressive in table 1). Label filters with OvA SVM base classifiers have 1% better P@1, 5% better P@5 and 2.5% better

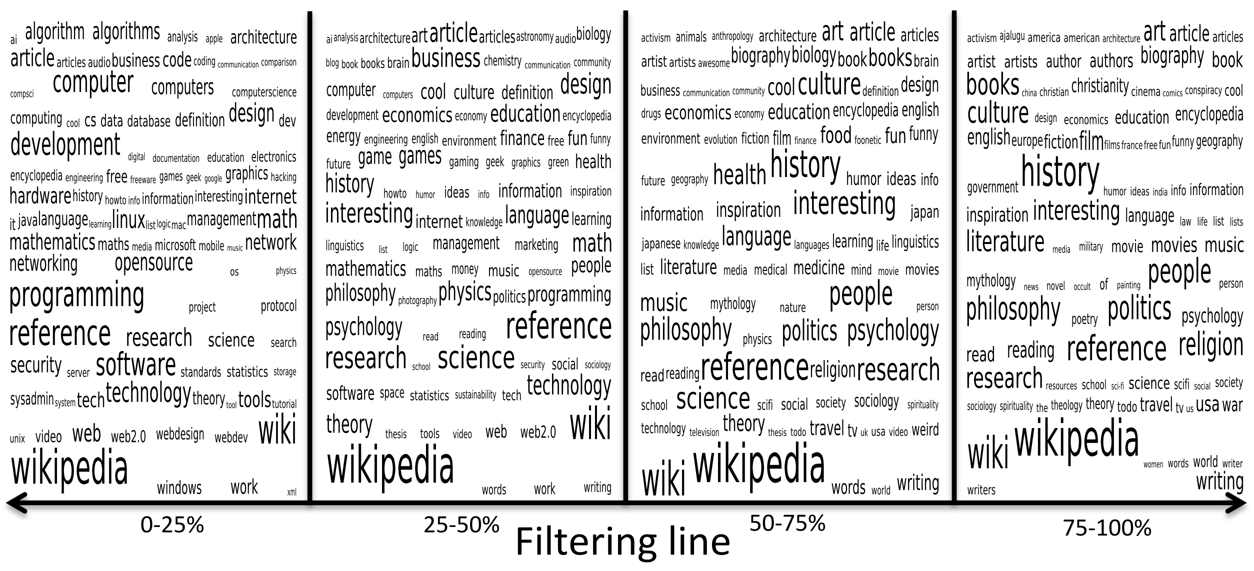


Figure 4: Most frequent labels in different regions of the filtering line for Wiki10 dataset. Test examples are sorted by their projection on the filtering line and split into 4 quantiles. The 100 most frequent labels in each quantile are displayed as a word cloud. Size is proportional to label frequency.

P@10 than FastXML, on Wiki10 and 5% better P@1, 2% better P@5 and similar P@10 on Amazon-670K, with similar speedups on both datasets. On WikiLSHTC the label filtering approach is about 50% faster and performs better than FastXML (12% higher P@1, 5% higher P@5 and 3% higher P@10). On the Delicious-Large dataset LF is 75% faster than FastXML, but the performance remains limited by the performance of the base OvA SVM classifier which trails behind FastXML by up to 2%. The same is true when using NB as a base classifier, which performs worse than FastXML on most datasets and metrics. These results show that the flexibility to use any base multilabel classifier in conjunction with label filters enables the user to obtain both high performance and fast prediction times.

We also report the performance of SLEEC and LPSR from (Bhatia et al., 2015)[6]. Like the label filter approach, LPSR (Weston et al., 2013) is wrapper technique designed to reduce the test time computational burden for any base multilabel classifier. However, unlike label filters, LPSR does significantly degrade performance making it non-competitive. Compared to the label filtering approach using the same Naive Bayes base classifier, the performance of LPSR can be more than 20% lower. One reason for the poor performance of LPSR might be that, in the high-dimensional regime, it fails to find clusters that are pure enough to safely eliminate a large fraction of labels. In contrast, because the label filters are specifically trained to retain the true labels while eliminating as many false labels as possible, they do not hinder performance.

SLEEC (Bhatia et al., 2015) is a state of the art label embedding based method that can achieve better performance than FastXML, but has significantly higher prediction times (e.g., SLEEC is 16 times slower than FastXML on WikiLSHTC (Bhatia et al., 2015)). The label filter approach is both more accurate and faster than SLEEC on Wiki10, WikiLSHTC and Amazon-670K datasets and slightly less accurate but still faster on DeliciousLarge.

5 Conclusions

We have proposed the use of label filters to speed up large scale multiclass and multilabel classifiers at production time by reducing the number of labels they have to consider. Label filters are designed to pre-select a small set of candidate labels, striving to eliminate as many labels as possible while retaining the true labels. We have shown how label filters can be learned from data by solving a constraint optimization problem, and that they significantly reduce the test-time computational burden without degrading the accuracy of the base multiclass or multilabel classifiers.

An open question is how to better encourage diversity among label filters so that they are complementary to

[6]The results are comparable since we are using the same train-test split as (Bhatia et al., 2015).

Table 1: Results for Label Filters and competing methods. Conservative LF attempts to preserve performance and aggressive LF attempts to match the FastXML speedup (see text). Speed is measured in terms of number of vector-vector multiplications. Results for LPSR and SLEEC are taken from (Bhatia et al., 2015) which does not report P@10 or speedup.

Dataset	Method	P@1	P@5	P@10	Speedup
Wiki10	NB	80.00	57.55	44.60	1x
	LF NB (conservative)	80.62	57.90	44.43	34x
	LF NB (aggressive)	79.70	56.95	43.18	73x
	LPSR NB(Bhatia et al., 2015)	72.2	49.0	–	–
	OvA SVM	85.03	66.96	51.20	1x
	LF SVM (conservative)	84.90	66.10	49.68	34x
	LF SVM (aggressive)	84.58	64.54	47.42	73x
	FastXML	83.6	59.4	44.8	60x
	SLEEC(Bhatia et al., 2015)	85.5	63.1	–	–
DeliciousLarge	NB	45.33	38.08	34.31	1x
	LF NB	45.25	37.98	34.17	428x
	LPSR NB(Bhatia et al., 2015)	18.59	14.07	–	–
	OvA SVM	48.28	38.06	33.33	1x
	LF SVM	47.55	37.64	33.02	428x
	FastXML	48.59	39.76	35.30	295x
	SLEEC(Bhatia et al., 2015)	47.03	38.88	–	–
WikiLSHTC	NB	45.03	21.09	13.21	1x
	LF NB	44.15	20.92	13.28	547x
	LPSR NB(Bhatia et al., 2015)	27.43	12.01	–	–
	OvA SVM	62.52	29.44	17.44	1x
	LF SVM	61.75	28.45	16.75	547x
	FastXML	49.21	23.11	13.81	381x
	SLEEC(Bhatia et al., 2015)	55.57	24.07	–	–
Amazon-670K	NB	39.48	33.03	22.90	1x
	LF NB (conservative)	39.10	31.82	21.71	98x
	LF NB (aggressive)	38.29	30.00	19.98	806x
	LPSR-NB(Bhatia et al., 2015)	28.65	22.37	–	–
	OvA SVM	41.58	34.16	22.74	1x
	LF SVM (conservative)	41.67	33.30	21.85	98x
	LF SVM (aggressive)	40.59	31.04	19.89	806x
	FastXML	35.72	28.94	20.06	824x
	SLEEC(Bhatia et al., 2015)	35.05	28.56	–	–

each other and eliminate different sets of labels. Using more diverse filters would increase their joint filtering power and further reduce the prediction time.

Another problem left for further study is how to use label filters to improve training time. Since label filters are trained independently of the base multilabel classifier, they can potentially be used to speed up the training phase. For example, when using the one vs. all approach, the binary classifier corresponding to a particular label could be trained using only examples for which the label of interest has not been eliminated by the label filter. This would speed up training and might even lead to improved accuracy by reducing the label imbalance and by preventing the classifier from focusing on regions of the space that will be filtered out anyway by the label filters.

Acknowledgments

We thank Hans Peter Graf, Erik Kruus, Cristian Danescu Niculescu-Mizil and Krzysztof Dembczyński for helpful discussions and suggestions.

References

Agrawal, R., Gupta, A., Prabhu, Y., and Varma, M. (2013). Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages. In *Proceedings of the 22nd international conference* *on World Wide Web*, pages 13–24. International World Wide Web Conferences Steering Committee.

Babbar, R. and Schölkopf, B. (2017). DiSMEC – Distributed Sparse Machines for Extreme Multi-label Classification. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM 2017)*, pages 721–729.

Bachrach, Y., Finkelstein, Y., Gilad-Bachrach, R., Katzir, L., Koenigstein, N., Nice, N., and Paquet, U. (2014). Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 257–264. ACM.

Bengio, S., Weston, J., and Grangier, D. (2010). Label embedding trees for large multi-class tasks. In *Advances in Neural Information Processing Systems*, pages 163–171.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517.

Beygelzimer, A., Langford, J., and Ravikumar, P. (2009). Error-correcting tournaments. In *Algorithmic Learning Theory*, pages 247–262. Springer.

Bhatia, K., Jain, H., Kar, P., Varma, M., and Jain, P. (2015). Sparse local embeddings for extreme multi-label classification. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 730–738. Curran Associates, Inc.

Bi, W. and Kwok, J. (2013). Efficient multi-label classification with many labels. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 405–413.

Chen, Y.-N. and Lin, H.-T. (2012). Feature-aware label space dimension reduction for multi-label classification. In *Advances in Neural Information Processing Systems*, pages 1529–1537.

Cissé, M., Artières, T., and Gallinari, P. (2012). Learning compact class codes for fast inference in large multi class classification. In *Machine Learning and Knowledge Discovery in Databases*, pages 506–520. Springer.

Cissé, M., Usunier, N., Artieres, T., and Gallinari, P. (2013). Robust bloom filters for large multilabel classification tasks. In *Advances in Neural Information Processing Systems*, pages 1851–1859.

Crammer, K. and Singer, Y. (2002). On the algorithmic implementation of multiclass kernel-based vector machines. *The Journal of Machine Learning Research*, 2:265–292.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

Deng, J., Satheesh, S., Berg, A. C., and Li, F. (2011). Fast and balanced: Efficient label tree learning for large scale object recognition. In *Advances in Neural Information Processing Systems*, pages 567–575.

Gao, T. and Koller, D. (2011). Discriminative learning of relaxed hierarchy for large-scale visual recognition. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2072–2079. IEEE.

Hsu, D., Kakade, S., Langford, J., and Zhang, T. (2009). Multi-label prediction via compressed sensing. In *NIPS*, volume 22, pages 772–780.

Indyk, P. and Motwani, R. (1998). Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM.

Ji, S., Tang, L., Yu, S., and Ye, J. (2008). Extracting shared subspace for multi-label classification. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 381–389. ACM.

Leskovec, J. and Krevl, A. (2014). SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`.

Liu, T.-Y., Yang, Y., Wan, H., Zeng, H.-J., Chen, Z., and Ma, W.-Y. (2005). Support vector machines classification with a very large-scale taxonomy. *ACM SIGKDD Explorations Newsletter*, 7(1):36–43.

Partalas, I., Kosmopoulos, A., Baskiotis, N., Artieres, T., Paliouras, G., Gaussier, E., Androutsopoulos, I., Amini, M.-R., and Galinari, P. (2015). LSHTC: A benchmark for large-scale text classification. *CoRR*, abs/1503.08581.

Prabhu, Y. and Varma, M. (2014). FastXML: a fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 263–272. ACM.

Ram, P. and Gray, A. G. (2012). Maximum inner-product search using cone trees. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 931–939. ACM.

Shrivastava, A. and Li, P. (2014). Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *Advances in Neural Information Processing Systems*, pages 2321–2329.

Shrivastava, A. and Li, P. (2015). Improved asymmetric locality sensitive hashing (ALSH) for maximum inner product search (MIPS). In *Proceedings of the*

Thirty-First Conference on Uncertainty in Artificial Intelligence, UAI 2015, July 12-16, 2015, Amsterdam, The Netherlands, pages 812–821.

Vijayanarasimhan, S., Shlens, J., Monga, R., and Yagnik, J. (2014). Deep networks with large output spaces. *CoRR*, abs/1412.7479.

Weston, J., Makadia, A., and Yee, H. (2013). Label partitioning for sublinear ranking. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 181–189.

Wetzker, R., Zimmermann, C., and Bauckhage, C. (2008). Analyzing social bookmarking systems: A del. icio. us cookbook. In *Proceedings of the ECAI 2008 Mining Social Data Workshop*, pages 26–30.

Zubiaga, A. (2009). Enhancing navigation on wikipedia with social tags. In *Wikimania 2009*. Wikimedia Foundation.