Log-time and Log-space Extreme Classification

Kalina Jasinska
Poznan Univeristy of Technology
kjasinska@cs.put.poznan.pl

Nikos Karampatziakis
Microsoft Research
nikosk@microsoft.com

Abstract

We present LTLS, a technique for multiclass and multilabel prediction that can perform training and inference in logarithmic time and space. LTLS embeds large classification problems into simple structured prediction problems and relies on efficient dynamic programming algorithms for inference. We train LTLS with stochastic gradient descent on a number of multicclass and multilabel datasets and show that despite its small memory footprint it is often competitive with existing approaches.

1 Introduction

Extreme multi-class and multi-label classification refers to problems where the size $C$ of the output space is extremely large. This problem appears in many application areas of machine learning, such as recommendation, ranking, and language modeling. The extreme setting brings a lot of challenges, such as, inter alia, time and space complexity of training and prediction, long tail of labels, missing labels and very few training examples per label. In this paper we undertake the first mentioned problem – the complexity, and propose the first, to our best knowledge, truly log-time and log-space training and prediction algorithm that can produce its top $k$ predictions in time $O(k \log(k) \log(C))$ for an output space of size $C$. To do so, we adapt ideas from structured prediction to impose an efficient search structure on top of any multiclass and multilabel problem. This allows us to (a) characterize when we expect our technique to work as well as an One-Vs-All (OVA) classifier and (b) work with any model that can be trained with (online) gradient descent including deep neural networks.

There exist several techniques for reducing training time, prediction time, and/or model size, in comparison to a OVA classifier. One technique is sparsity which can reduce model size and sometimes training and prediction times due to fewer operations. An example of such an approach is PD-Sparse [4], where the authors show that it is possible to get accurate sparse models in high dimensional datasets. However sparsity is not guaranteed to reduce the model size without severely hurting model accuracy. Another group of methods are embedding based models. Examples of such methods are SLEEC [1], LEML [9], WSABIE [8] or Rembrandt [6]. These techniques can be thought of as (supervised) dimensionality reduction followed by an OVA classifier. All these approaches still remain linear in the size of the output space during training and prediction unless additional approximations are employed, such as subsampling the negative classes.

Another group consists of tree based models. Those can be further divided into decision tree based and label tree based methods. Those methods reduce prediction time, but not necessary lead to models with space complexity that is logarithmic in number of labels. For example, a multi-label decision tree based method, FastXML [7] builds a tree of depth logarithmic in the number of training examples. Label tree based method, PLT [5], has a $O(\log(C))$ training time, since an update with one training instance is applied to $O(\log(C))$ models. Even though this algorithm reduces prediction

*Work done while the author was visiting Microsoft

Submitted to Extreme Classification Workshop at NIPS 2016
We denote with $(least\ significant)$ bit in binary representation of which in this context is known as the Viterbi algorithm. It requires there are many topologies we could have selected but in this paper we are exploring a minimal one. The graph, see an example on Figure 1, is a trellis of vector, $2$ Problem Setting

Our idea is to embed our multiclass/multilabel problem in a structured prediction problem where the structured label $s$ is a combinatorial object. The model is usually specified via a compatibility function $F(x, s; w)$ which measures how well feature vector $x$ and $s$ go together for a particular setting of the parameters $w$. The inference problem $\hat{s} = \arg\max_{s \in S} F(x, s; w)$ requires maximization over a combinatorial set which can sometimes be solved exactly in time $O(poly \log(|S|))$ such as when $S$ is the set of spanning trees or the set of matchings of a graph. Our criteria for selecting a structured prediction problem are the simplicity of algorithms for finding the top 1 and top $k$ elements of $S$ according to $F(x, s; w)$.

In this paper our construction is a directed acyclic graph (DAG) $G$ that contains exactly $C$ paths from a source vertex to a sink vertex. Every edge $e$ in the graph is associated with a learnable function $h_e(\cdot; w) : \mathcal{X} \rightarrow \mathbb{R}$. Every class corresponds to a path and the model predicts the class with the highest scoring path. As usual, the score of a path is the sum of the scores of the edges in the path. There are many topologies we could have selected but in this paper we are exploring a minimal one because in this way we can force every learnable parameter to train roughly on 25% of the data (for problems whose class prior is close to uniform); thus avoiding data decimation issues.

The graph, see an example on Figure 1 is a trellis of $\lceil\log(C)\rceil$ steps with $2$ “states” on each step. The source is connected to each vertex in the first step. An auxiliary vertex collects all vertices of the last step to one point. If we connect the sink vertex to the auxiliary vertex then we can only express multiclass/multilabel problems where $C$ is a power of 2. In order to handle an arbitrary number of classes $C$ we connect the sink to one of the states at step $i$ for all $i$ such that the $i$-th (least significant) bit in binary representation of $C$ is 1. The upper bound of number of edges $E$ in the graph is $5\lceil\log_2 C\rceil + 1$. The maximum scoring path can be found with dynamic programming, which in this context is known as the Viterbi algorithm. It requires $O(E)$ steps. For each edge in order $e_1, e_2, \ldots$, the algorithm updates the highest scoring path from the source to its destination by checking whether this edge would increase it. After all edges are processed the algorithm backtracks to retrieve the longest path. The top-$k$ scoring paths can be found by a modification of the Viterbi algorithm called list Viterbi. We will refer to the proposed approach using name LTLS, which stands for Log-Time Log-Space.

3 Proposed Approach

A path $s$ is a vector of length $E$, with $s_e = 1$ if edge $e$ is a part of this path, and $s_e = 0$ otherwise, such that one can reach from the source vertex to the auxiliary vertex in the graph $G$ using all, and only, edges in $s$. We say that edge $e \in s$ if and only if $s_e = 1$. There are exactly $C$ paths in graph $G$. All paths in a graph $G$ stacked horizontally form a matrix $M_G$ of dimensionality $C \times E$. Each label $\ell$ in $\mathcal{L} = \{1, 2, \ldots, C\}$ is exclusively assigned to a path $s(\ell)$.

Given a feature vector $x$ of dimension $D$ and model weights $w$ every edge $e$ gets a value $h_e(w, x)$. Values for all the edges form a $E$-dimensional vector $h(w, x)$. The score of the model for a label $\ell$ is the score for the corresponding path $s(\ell)$

$$F(x, s; w) = \sum_{e \in s} h_e(w, x) = s \cdot h(w, x), \quad (1)$$

all the label scores are based on the same $h(w, x)$. 2
While we can show that if a well performing OVA linear model
This loss also works for multiclass problems so we will use it for all our experiments when the
underlying model is a deep network.
(dual) sparsity. We use the
Therefore the model is a low-rank model \( f = M_G h(w, x) \) embedding the label vector of dimension
underlying model is linear. Finding the loss requires finding scores of two labels only,
Fortunately, our approach can also be used as an output layer of a deep network, where the scores of the edges \( h(w, x) \) are given by a deep network while still being able to perform inference and backpropagate through this layer in \( O(\log(C)) \).

4.1 Underlying models
LTLS can use various learnable functions to estimate edge values \( h(w, x) \). The most basic one may
use a linear model to predict each edge weight. Then the weights become
For multilabel classification we restrict ourselves to linear predictors and use a loss that induces
4.5 Optimization
One of our goals is to have logarithmic training time. For multiclass classification this is easy even
for multinomial logistic regression because the trellis graph can compute the log partition function
\( \log \sum_{\ell = 1}^{C} \exp(F(x, s(\ell); w)) \) efficiently. Backpropagation (also known as the forward-backward algorithm in this context) can be used to compute derivatives for all parameters. This is what we use
when the underlying model is a deep network.
For multilabel classification we restrict ourselves to linear predictors and use a loss that induces
(dual) sparsity. We use the separation ranking loss \([3, 4]\), being zero if all the positive labels \( \mathcal{P}(y) \) for given instance \((x, y)\) have higher scores than all the negative labels \( \mathcal{N}(y) \) plus a margin, and being the difference between highest scoring negative label \( \ell_n \) score \( F(\cdot, s(\ell_n), w) \) plus a margin and lowest scoring positive label \( \ell_p \) score \( F(\cdot, s(\ell_p), w) \). Formally,
\[
L(w, y) = \max_{\ell_n \in \mathcal{N}(y)} \max_{\ell_p \in \mathcal{P}(y)} (1 + F(\cdot, s(\ell_n), w) - F(\cdot, s(\ell_p), w))_.
\]
This loss also works for multiclass problems so we will use it for all our experiments when the
underlying model is linear. Finding the loss requires finding scores of two labels only, \( \ell_n \) and \( \ell_n \). Those can be found efficiently. Getting a score \( F(\cdot, s(\ell), w) \) for a given label \( \ell \) is \( O(E) \).
In the multilabel case \( F(\cdot, s(\ell_p), w) \) is the score of the one positive class. In multilabel case
\( F(\cdot, s(\ell_p), w) = \max(\{F(\cdot, s(\ell), w) : \ell \in \mathcal{P}(y)\}) \). Since in extreme classification \( |\mathcal{P}(y)| \ll C \), this step is fast. To find \( \ell_n \) in multiclass case it is sufficient to find the 2 longest paths, in multilabel
the \( |\mathcal{P}(y)| + 1 \) longest paths, and determine which of them is negative.
Since the label score $F(\cdot, s(\ell), w)$ is defined as a sum of edge scores, we need to update only the learnable functions $h_e(\cdot; w)$ for edges $e$ in the symmetric difference of $s(l_p)$ and $s(l_n)$.

We use stochastic gradient descent with averaging to minimize the loss. For the linear model the gradient is 0 if the loss is 0 and otherwise it is $x$ for the models on the edges used only by the positive path and $-x$ for the models on the edges used only by the negative path.

Figure 2: A graph with example lowest scoring positive path (green) and highest scoring negative path (red). Learnable functions on edges $e_7$ and $e_{12}$ get a negative update, on $e_8$ and $e_{14}$ a positive, $e_0$, $e_2$, $e_{17}$ and $e_{18}$ are not updated.

5.1 Label-path assignment policy

Since the decompression matrix $M_G$ structure is fixed to enable fast inference, bipartite matching between labels $\ell \in \mathcal{L}$ and paths $s \in \mathcal{S}$ becomes an important issue. To keep the training online and fast we could not propose a very complex method for finding a good path for each class. A simple approach is once an instance $(x, y)$ with an unseen label $\ell$ is encountered, we find the top $m$ paths for $x$ and assign $\ell$ to the highest ranked free path. If there is no free path we assign a random path. We restrict size of the ranking, so that $m$ is $O(\log(C))$, to keep the training fast. While this increases our memory requirements to $O(C)$ (for knowing which paths are free) this memory is not for model parameters and therefore stays constant as the input size (or the model size in case of deep networks) increases. Training time also increases to $O(\log^2(C) \log \log(C))$ but in our experiments this makes no difference.

6 Experiments

This section presents an experimental evaluation of LTLS\textsuperscript{2}. First we report the results of LTLS with a simple linear model on each edge and separation ranking loss. We have run LTLS on the datasets used in [4], where one can find a comparison of a set of multi-class and multi-label algorithms in terms of precision@1, training and prediction time, and model size. In the reported training times bear in mind that LTLS implementation is at the moment in Python, while other algorithms are implemented in compiled languages.

In Tables[1] and [2] we compare LTLS with LOMtree, FastXML, and LEML, for which we report the results from [4]. In case of multi-class problems on all except one dataset LTLS gets results comparable to the LOMtree, while creating a smaller model and providing the predictions faster (even though LTLS is currently in Python). The results using described assignment policy are significantly better than using random assignment.

A low result on the ImageNet is a good starting point for analysis when does LTLS work, and when does not. The ImageNet dataset is quite dense in comparison to other multi-class datasets used in the experiments, average number of features active for an example is 308 out of 1000. A model build

\textsuperscript{2}Code is available at [https://github.com/kjasinska/ltls]
by LTLS, weight matrix $w$, is dense – there are nearly no zero elements. This is because learnable function on each edge must learn many classes. Class distribution is fairly balanced in case of this dataset, so a classifier on each edge is being updated with many examples. Therefore we see that the underlying linear model on each edge it too simple to learn to distinguish between classes when the feature space is small and dense.

To verify the hypothesis that the poor result stems from the fact that the underlying model is too simple, we have used LTLS with a deep network. We have used a network with $E$ outputs to predict edge weights, and LTLS as an output layer, decoding $E$ outputs to $C$ classes. With a network with 2 layers, 500 hidden units in each, and ReLU nonlinearities, after 10 iterations of training we have reached 0.0507 test precision.

On two of the multi-class datasets, LSHTC1 and Dmoz, we have observed that LTLS overfitted. We have trained by adding an $L_1$ regularization term in the objective with strength $\lambda$. This simply means predicting with soft-thresholded weights $w$:

$$st(w_{ij}, \lambda) = \begin{cases} 
    w_{ij} - \lambda & w_{ij} > \lambda \\
    w_{ij} + \lambda & w_{ij} < -\lambda \\
    0 & |w_{ij}| \leq \lambda.
\end{cases}$$

For multi-label datasets the results are mixed. LTLS has performed well on rcv1-regions and LSHTCwiki, especially taking into account also prediction time and model size in case of the biggest dataset. On Eur-Lex we have encountered underfitting.

So far we have compared LTLS, a very limited model in terms of model size, with methods without such constraint. In table 3 we present comparison of LTLS results with a naive baseline algorithm having the same model size and $O(\log(C))$ prediction time. We have trained a 1-vs-All classifier for $E$ most frequent labels in each dataset. As a binary classifier we have used L2-regularized Logistic Regression with tuned regularization constant. We report an upper bound for the result, result of the naive baseline and LTLS result.

7 Conclusions

We have presented LTLS the first log-time and log-space technique for extreme classification. By embedding extreme problems into structured prediction we are able to address both time and space complexities while providing clear connections with low rank models and ways to incorporate deep learning into extreme classification. Many of our design choices have been motivated purely from convenience and leave many interesting questions on the impact of these choices as open questions for future work.

Acknowledgments

Kalina Jasinska is also supported by the Polish National Science Centre under grant no. 2013/09/D/ST6/03917. Some experiments were run in Poznan Supercomputing and Networking Center under computational grant no 243.

References

[1] K. Bhatia, H. Jain, P. Kar, M. Varma, and Jain P. Sparse local embeddings for extreme multi-label classification. In NIPS, 2015.
[2] A. Choromanska and J. Langford. Logarithmic time online multiclass prediction. In NIPS, 2015.
[3] Koby Crammer and Yoram Singer. A family of additive online algorithms for category ranking. J. Mach. Learn. Res., 3:1025–1058, March 2003.
[4] I. En-Hsu Yen, X. Huang, P. Ravikumar, K. Zhong, and I. Dhillon. Pd-sparse : A primal and dual sparse approach to extreme multiclass and multilabel classification. In Proceedings of The 33rd International Conference on Machine Learning, 2016.
Table 1: Results on multi-class datasets. We mark with an asterisk * results where LTLS cannot fit the data, and with a dagger † results of a L1-regularized LTLS model.

| Sector | LTLS | LOMtree | FastXML |
|--------|------|---------|---------|
| aloi.bin | 8658 | 0.8845 | 0.8210 | 0.8490 |
| features | 55197 | 0.14 | 0.16 | 0.25 |
| classes | 105 | 5.91 | 17.00 | 7.00 |
| LSHTC1 | 100000 | 0.8224 | 0.8947 | 0.9550 |
| features | 636911 | 1.00 | 1.59 | 10.99 |
| classes | 1000 | 102 | 106 | 992 |
| ImageNet | 83805 | 0.0950 | 0.1056 | 0.2166 |
| features | 347255 | 0.65 | 6.84 | 6.33 |
| classes | 12294 | 260 | 744 | 308 |
| Dmoz | 100000 | 0.8224 | 0.8947 | 0.9550 |
| features | 636911 | 1.00 | 1.59 | 10.99 |
| classes | 1000 | 102 | 106 | 992 |

Table 2: Multi-label results. We mark with an asterisk * results where LTLS cannot fit the data.

| Sector | LTLS | LEML | FastXML |
|--------|------|------|---------|
| Bibtex | 5991 | 0.2719 | 0.6401 | 0.6414 |
| features | 1837 | 0.07 | 0.22 | 0.09 |
| classes | 159 | 264.0 | 8.6 | 27.0 |
| rcv1regions | 20835 | 0.8964 | 0.9628 | 0.9328 |
| features | 47237 | 0.52 | 2.52 | 0.82 |
| classes | 225 | 6.15 | 205.00 | 14.60 |
| Eur-Lex | 15643 | *0.0559 | 0.6782 | 0.6730 |
| features | 5000 | 0.24 | 42.24 | 1.00 |
| classes | 3956 | 1.79 | 78.00 | 324.50 |
| LSHTCwiki | 2355436 | 0.2240 | 0.2846 | 0.7828 |
| features | 2085167 | 5.43 | 2896.00 | 164.80 |
| classes | 320338 | 769 | 10400 | 8900 |

[5] K. Jasinska, K. Dembczynski, R. Busa-Fekete, K. Pfannschmidt, T. Klerx, and E. Hullermeier. Extreme f-measure maximization using sparse probability estimates. In *Proceedings of The 33rd International Conference on Machine Learning*, 2016.

[6] P. Mineiro and N. Karampatziakis. Fast label embeddings via randomized linear algebra. In *ECML*, 2015.

[7] Y. Prabhu and M. Varma. FastXML: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *KDD*, 2014.

[8] J. Weston, S. Bengio, and N. Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *IJCAI*, 2011.

[9] H. Yu, P. Jain, P. Kar, and I. Dhillon. Large-scale multi-label learning with missing labels. In *Proceedings of The 31st International Conference on Machine Learning*, 2015.
Table 3: Naive baseline precision@1 results compared to LTLS.

| dataset       | #edges | LTLS top-#edges oracle | LR       | LTLS       |
|---------------|--------|-------------------------|----------|------------|
| sector        | 28     | 0.2362                  | 0.2248   | 0.8945     |
| aloi.bin      | 42     | 0.0275                  | 0.0274   | 0.8224     |
| LSHTC1        | 56     | 0.1463                  | 0.0966   | 0.0950     |
| imageNet      | 42     | 0.0697                  | 0.0340   | 0.0075     |
| Dmoz          | 61     | 0.3507                  | 0.2376   | 0.2304     |
| bibtex        | 34     | 0.7126                  | 0.2220   | 0.2719     |
| rcv1-regions  | 34     | 0.8644                  | 0.6576   | 0.8964     |
| Eur-Lex       | 52     | 0.6672                  | 0.1262   | 0.0579     |
| LSHTCwiki     | 81     | 0.2520                  | 0.0314   | 0.2240     |