Schedae Informaticae Vol. 25 (2016): 9–23
doi: 10.4467/20838476SI.16.001.6182

tfml 2017
theoretical foundations
of machine learning, Kraków

Extreme Classification under Limited Space and Time Budget

Kalina Jasinska[1], Krzysztof Dembczyński[1],
Nikos Karampatziakis[2]
[1]Institute of Computing Science, Poznan University of Technology, Poznań, Poland
[2]Microsoft Research, Redmond, USA
e-mail: *{kjasinska,kdembczynski}@cs.put.poznan.pl*, *nikosk@microsoft.com*

Abstract. We discuss a new framework for solving extreme classification (i.e., learning problems with an extremely large label space), in which we reduce the original problem to a structured prediction problem. Thanks to this we can obtain learning algorithms that work under a strict time and space budget. We mainly focus on a recently introduced algorithm, referred to as LTLS, which is to our best knowledge the first truly logarithmic time and space (in the number of labels) method for extreme classification. We compare this algorithm with two other approaches that also rely on transformation to structured prediction problems. The first algorithm encodes original labels as binary sequences. The second algorithm follows the label tree approach. The comparison shows the trade-off between computational complexity (in time and space) and predictive performance.

Keywords: supervised learning, space and time complexity of learning algorithms, extreme classification, multi-class classification, learning reductions

1. Introduction

Extreme classification refers to multi-class and multi-label problems where the size m of the output space is extremely large. This type of problems appears in many application areas of machine learning, such as recommendation, ranking, and language

Received: 11 December 2016 / Accepted: 30 December 2016

modeling. The extreme setting brings a lot of challenges, such as, time and space complexity of training and prediction, long tail of labels, missing labels and very few training examples per label.

A naive solution for the problem is to train an independent model for each label individually. This approach, often referred to as one-vs-all (OVA), has linear time and space complexity in the number of labels. Unfortunately, in many real world problems this complexity is too costly. One of the main challenges of extreme classification is to reduce the complexity, bearing in mind the need to control the trade-off between lowering the complexity and retaining good predictive performance. To achieve this goal, we consider a new approach for solving extreme classification, which casts the original problem to a structured prediction problem.

An example of a simple structured prediction problem is sequence labeling in which a categorical label is assigned to each member of a sequence of observed values. The aim is to find the most probable labeling for a given sequence. Finding the best output for a given example usually relies on performing a complex inference task. For simple problems, different variants of Viterbi algorithm [1, 2] can be applied. For more complex problems, advanced search techniques are used to explore the large output space efficiently [3]. A structured prediction problem can be treated as a complex multi-class classification problem over all possible label assignments to the sequence. In this paper, we consider the opposite approach that casts classification to structured output prediction. The simplest transformation of this type is to encode labels by sequences of bits. The length of the sequence does not have to be the same for each label. Then, by choosing a proper dependence structure between bits and using appropriate training and inference methods one can get a very compact representation of an extreme classification problem. In this new reduction framework, we can formulate the problem as optimizing the predictive performance under limited time and space budged.

Recently, Jasinska and Karampatziakis [4] have introduced a truly log-time and log-space training and prediction algorithm that can produce its top k predictions in time $O(k \log(k) \log(m))$ for an output space of size m. To do so, the algorithm, referred to as LTLS, encodes labels as paths in a trellis of width 2. The inference consists in finding the longest weighted path in the trellis from the source node to the sink node. Each weight associated with an edge in the trellis is obtained from a binary classifier trained by stochastic gradient descent. To perform efficient inference a variant of Viterbi algorithm is used. In the following we compare this approach to two other methods. The first method is also logarithmic in time and space, but treats all the code bits to be independent. We refer to this method as sequences of independent bits (SIB). The second method are probabilistic classifier trees (PCTs) [6, 5]. In this method the labels are coded by paths in a tree. Such a tree can be treated as a generalization of the trellis. Unfortunately, for this method we cannot give strict logarithmic bounds on time and space, but as shown in [5] this method is statistically consistent. In this paper we focus on multi-class problems, however, PCTs and LTLS can also be easily modified to multi-label problems [4, 6].

The paper is organized as follows. The next section shortly describes state-of-the-art methods for extreme classification. Section 3 states the problem formally. In Section 4 and 5 we present the SIB and PCT algorithm, respectively. Section 6 describes a variant of LTLS that is made to be similar to the previous methods in

order to perform a fair comparison between these different approaches. Section 7 presents experimental results. Last section concludes the paper.

2. Related work

There are several groups of extreme classification algorithms that follow different paradigms such as sparsity, low-rank approximation, tree-based search, or label filtering.

The sparsity-based methods can reduce model size and sometimes training and prediction times due to fewer operations. An example of such an approach is PD-Sparse [7], 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. Examples of the low-rank methods, also called embedding methods, are SLEEC [8], LEML [9], WSABIE [10] or Rembrandt [11]. 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 of the negative labels.

The tree-based approaches can be divided into decision tree- and label tree-based methods. Those methods reduce prediction time, but not necessary lead to models with space complexity that is logarithmic in the number of labels. For example, FastXML [12] builds a tree of depth logarithmic in the number of training examples. The multi-class logarithmic time prediction is also addressed by LOMtree [13]. Label tree-based methods such as PCT, discussed later in this paper, have usually $O(\log(m))$ training time, since an update with one training instance is applied to $O(\log(m))$ models. Even though these algorithms reduce prediction time significantly, by not querying all the models, their complexity in general is greaten than $O(\log(m))$.

The last group of algorithms assumes that learning can be performed off-line (so the complexity of training is allowed be higher) and focuses on the use of appropriate data structures to accelerate classification of test examples in the prediction phase. Therefore, this approach is sometimes called label filtering [14] as it avoids a linear scan over all labels. The label partitioning for sublinear ranking method [15] uses clustering to group training examples, and then assigns a set of possible labels to each group in a way that optimizes the overall performance. The clustering step is used only for filtering the labels and is performed independently from training a final model which can be any multi-class or multi-label classifier, even very expensive. During classification a test example is first assigned to one of the groups, and then the final model is called only for labels assigned to this group. Other approaches use Bloom filters [16], filtering lines [14] or tree structures [17]. In case of linear models (e.g., logistic regression, perceptron, the last layer in a deep network), the problem of speeding up classification of test examples is often referred to as maximum inner product search (MIPS) [18, 19]. An exact solution can be optimally obtained by the so-called threshold algorithm [20] which can be used in a variety of machine learning tasks [21]. However, this algorithm does not scale well to extreme classification.

Therefore approximate algorithms need to be considered. The MIPS problem is similar, but not equivalent, to the nearest neighbor search. It is therefore possible to adapt approximate nearest neighbor algorithms to this problem. For example, a modified variant of the locality-sensitive hashing [22] has been introduced in [19]. Let us also emphasize that the MIPS problem can be applied during training as a specific instance of negative sampling [18].

3. Problem Setting

In the following we consider multi-class classification problems. We denote with $(\boldsymbol{x}, y)$ a multi-class instance, where $\boldsymbol{x}$ is a feature vector, $\boldsymbol{x} \in \mathbb{R}^d$, and y a label, $y \in \{1, \ldots, m\}$. We focus on classification methods that are optimized for precision@k. In case of $k = 1$, this performance measure corresponds to accuracy, or stated differently, to:

$$\mathit{precision}@1 = 1 - \ell_{0/1}(y, f(\boldsymbol{x})) = 1 - [\![y \neq f(\boldsymbol{x})]\!] ,$$

where $\ell_{0/1}(y, f(\boldsymbol{x}))$ is the 0/1 loss and $f(\boldsymbol{x})$ a multi-class classifier.

The methods we consider rely on representing labels as binary sequences. The difference between methods lays in the choice of encoding and the assumed dependence structure between elements of the sequence. In general, this approach reduces the original problem to a bunch of binary subproblems. The task is then to appropriately transform original training examples to binary examples for each subproblem. During prediction, the binary outcomes are decoded to original labels. This inference task can vary depending on the chosen encoding and dependence structure.

We analyze each method under a strict time and space budget. For example, we would like to have budget that is logarithmic in the number of labels. We follow here the learning reduction framework [23] which studies a decomposition of complex learning tasks into simpler problems for which numerous and powerful algorithms are available. This decomposition should guarantee that a solution to the simple subproblems gives a solution to the original problem [24]. Ideally, a no-regret (i.e., optimal) solution to each base problem should translate into an optimal solution to the original problem. In such case, a reduction (i.e., decomposition) is called *consistent*. It is, however, an open question whether we can obtain consistent reductions under a strict time and space budget.

We assume that the time complexity of training of a binary classifier with respect to a single training example is $O(1)$. Similarly, calling a binary classifier for a single test example is also $O(1)$. The space complexity is harder to formalize in this way. We assume, however, that storing a single binary classifier is also $O(1)$. In this way we can easily express the computational complexity in terms of the number of labels. In the analysis, we do not take into account the space complexity needed for storing the mapping between original labels and bit sequences, which in general is $O(m)$, and the time complexity needed to encode labels to bit sequences.

4. Simple coding by sequences of independent bits

We start our discussion with a very simple algorithm. It relies on encoding labels as sequences of independent bits in such a way that each label is assigned to one and only one binary code. We refer to this approach as sequences of independent bits (SIB). In general, we encode an original label y by a binary code $c(y)$ of length l. Let $c_i(y)$ indicate the i-th bit in the code. If we assume that bits of the code are independent, then the probability of label y can be obtained by:

$$P(y \,|\, \boldsymbol{x}) = P(c(y) \,|\, \boldsymbol{x}) = \prod_{i=1}^{l} P(c_i(y)|\boldsymbol{x}) \,.$$

To get the model, we need to train base classifiers that estimate $P(c_i(y)|\boldsymbol{x})$. We can use any method that estimates probabilities, for example, logistic regression.

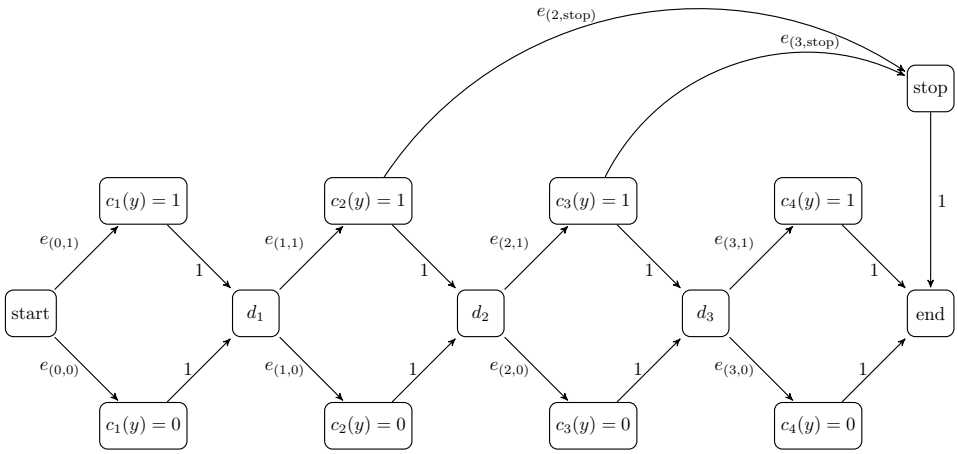

Figure 1. A trellis used in SIB for $m = 22$. Node *start* is connected to both states of the first bit; both states of the last bit are connected to node *end*. To handle an arbitrary number of classes m we connect to the *stop* node the *up* states at bits corresponding to 1's in binary representation of m.

When the number of labels is a power of 2, then we can use binary coding of fixed length. In such a case, learning and prediction with SIB is straight-forward. Otherwise, we need to use a specific coding to make the method logarithmic in time and space with the number of labels. We use here encoding similar to the one used in LTLS [4]. The codes of labels can be visualized as paths in a trellis with additional by-pass edges (see Figure 1). The binary states of bits are represented by *up* and *down* nodes (for the i-th bit, these are nodes $c_i(y) = 1$ and $c_i(y) = 0$, respectively). Moreover, we use auxiliary nodes, *begin*, *end*, and *stop*. The first two indicate the start and the end of the code. The *stop* node determines an early stop of the code. The intermediate

nodes d_i are used to show independence of bits. In this representation, we have exactly m paths, one for each label. More formally, let $2^a \leq m \leq 2^{a+1}$. Then, the first 2^a labels can be coded using a vanilla binary code on a bits. The rest of labels $b = m - 2^a$ are coded using shorter codes. We assume that the last bit in such code is always set to 1. In that way we can encode arbitrary number b of labels, $b \in \{1, 2^a - 1\}$. If the code ends after the i-th bit, then we get additional 2^{i-1} codes. The use of this code, however, requires to train a base classifier for additional *stop* class, for each bit i we use to extend the number of codes.

All incoming edges to *up*, *down*, and *stop* nodes are associated with a probability estimation function $Q_e(\boldsymbol{x})$. To indicate an edge we use a pair $(i, c_i(y))$, where i is the bit and $c_i(y)$ its value. For all other edges, we use weight equal 1. Thanks to this convention, prediction of the most probable label corresponds to finding the most probable path. To this end, we can use dynamic programming, which in context of probabilistic models is known as the Viterbi algorithm [1]. The top-k scoring paths can be found by a modification of the Viterbi algorithm called List Viterbi [2]. Let us remark that the upper bound of the number E of edges with probability estimates in the trellis is $3\lfloor \log_2 m \rfloor$. So, the space complexity of the model is logarithmic in the number of labels. Since the Viterbi algorithm is also linear with the number of edges, the time complexity is also logarithmic.

To compute estimates $P(c_i(y)|\boldsymbol{x})$, we train edge classifiers $Q_{(i,c_i(y))}(\boldsymbol{x})$ in such a way that:

$$Q_{(i,0)} + Q_{(i,1)} = 1\,.$$

Moreover, we need to train additional classifiers that check the early stop of the code, i.e., $Q_{(i,\text{stop})}$. In such a case, we can assume:

$$Q_{(i,0)} + Q_{(i,1)} + Q_{(i,\text{stop})} = 1\,.$$

Let us underline that the code described above is not entirely binary, as it includes additional symbol *stop*.

The independence assumption made in SIB is rather unrealistic. Therefore, this method will often lead to a very crude approximation. However, this simple approach can be treated as a good baseline for other methods with a strict time and space budget, since its time and space complexity is logarithmic with the number of labels. Let us also remark that SIB resembles the ECOC (Error-Correcting Output Codes) approach for classification [25]. ECOC uses, however, codes with additional redundancy, what is not a case of SIB.

5. Probabilistic classifier trees

In contrast to SIB, probabilistic classifier trees (PCTs) [5] take all dependences between bits into account. Moreover, they use prefix codes, so there is no need to use any additional symbol, like *stop* in case of SIB. For label y coded by $c(y)$ of length l its

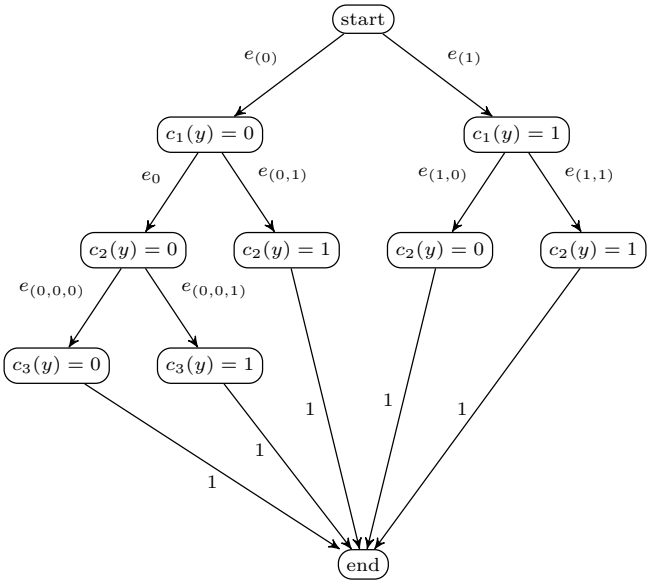

Figure 2. A tree used in PCT for $m = 5$. All leaf nodes are additionally connected with node *end* to resemble the trellis used in SIB.

conditional probability in PCT is given by:

$$P(y \,|\, \boldsymbol{x}) = P(c(y) \,|\, \boldsymbol{x}) = \prod_{i=1}^{l} P(c_i(y)|c_{i-1}(y), \ldots, c_1(y), \boldsymbol{x}) \,.$$

In other words, each bit of the code depends on all antecedent bits. This formulation is known as the chain rule of probability and holds for any joint distribution, i.e., any distribution can be factorized in this way. Therefore PCTs are statistically consistent [5].

Let us recall that any prefix code can be represented by a tree with 0/1 splits. If the number of labels is a power of 2, then binary coding of fixed length can be used (in result a fully balanced tree is obtained). Otherwise, one can use complete trees or Huffman coding [26]. Each path from the root to a leaf node corresponds to a code word. Figure 2 shows an example of a coding tree for multi-class classification with 5 labels. To make the representation of the tree to be similar to the trellis used in SIB, we denote the root node as the *start* node and added the *end* node. All nodes except those incoming to the *end* node are associated with probability estimators $Q_e(\boldsymbol{x})$. To indicate an edge, we use bits on a path from the *start* node to the node to which the edge directs, i.e., $(c_1(y), \ldots, c_i(y))$. To compute estimate $P(c_i(y)|c_{i-1}(y), \ldots, c_1(y), \boldsymbol{x})$, we train edge classifiers $Q_{(c_1(y),\ldots,c_i(y))}(\boldsymbol{x})$ in such a way that:

$$Q_{(c_1(y),\ldots,c_{i-1}(y),0)}(\boldsymbol{x}) + Q_{(c_1(y),\ldots,c_{i-1}(y),1)}(\boldsymbol{x}) = 1 \,.$$

It is easy to see that there are $m - 1$ classifiers in the tree (i.e., one classifier per internal node of the tree). Therefore, the space complexity of this method is linear

in the number of labels. However, by using the hashing trick [27] jointly over all models, the space complexity can be made constant. One can also use model sharing to reduce the space complexity, for example, by using one model for each tree level. The learning time for Huffman and balanced trees is logarithmic, since a given training example is used only in classifiers on a path corresponding to the code of a given label. Prediction can be logarithmic if it is made in a greedy way. PCTs can use, however, more involved search procedures such as uniform-cost search or A* [28, 29]. Thanks to them, the regret of PCTs can be bounded. Moreover, it can be shown that search time is inversely proportional to the probability of the top label. If this probability is lower bounded, then for the balanced trees the top label can be found in the logarithmic time. In the worse case scenario, however, the time complexity of the prediction procedure is linear in the number of labels [5, 28].

Let us notice that similar algorithms to PCTs appear under different names in the literature. In multi-class classification, this method is also known under the name of conditional probability trees [30] and nested dichotomies [31]. The same concept is also known in neural networks and natural language processing under the name of hierarchical softmax [32].

6. LTLS

LTLS, proposed by Jasinska and Karampatziakis [4], is a model that can be situated in-between sequences of independent bits and probabilistic classifier trees. In the following, we consider a specific instance of this approach that resembles maximum entropy markov models (MEMMs). In general, MEMMs take dependencies up to the k-th degree into account:

$$P(y \mid \boldsymbol{x}) = P(c(y) \mid \boldsymbol{x}) = \prod_{i=1}^{l} P(c_i(y) | c_{i-1}(y), \ldots, c_{i-k}(y), \boldsymbol{x}) \,.$$

We use $k = 1$, i.e., the current bit depends only on one previous bit. There are different possibilities of estimating $P(c_i(y)|c_{i-1}(y), \boldsymbol{x})$ and coding of original labels. LTLS undertakes the following approach.

As already mentioned above, SIB use the same encoding as LTLS. The trellis used in LTLS, presented in Figure 3, resembles the one used in SIB. As before, we have *up* and *down* nodes, and three auxiliary nodes, *start*, *end*, and *stop*. The number of bits is also $\lfloor \log(m) \rfloor$. The main difference lays in additional edges that model dependencies between consecutive bits. Therefore, the upper bound of the number E of edges with probability estimates in the trellis is $5\lfloor \log_2 m \rfloor$. To indicate an edge we use here a triple $(i, c_i(y), c_{i-1}(y))$, i.e., $e = (2, 1, 0)$ means that the edge is between the *up* state of the second bit and the *down* step of the first bit. As before, prediction is made by using the Viterbi algorithm.

Training of LTLS is logarithmic in the number of labels. To compute estimates of $P(c_i(y)|c_{i-1}(y), \boldsymbol{x})$, LTLS trains edge classifiers $Q_{(i,c_i(y),c_{i-1}(y))}$ in such a way that

$$Q_{(i,0,1)} + Q_{(i,1,1)} = 1 \quad \text{and} \quad Q_{(i,1,0)} + Q_{(i,0,0)} = 1\,.$$

As in case of SIB, we also need to train additional classifiers checking the early stop of the code, i.e., $Q_{(i,\text{stop},1)}(\boldsymbol{x})$. In such a case, we have:

$$Q_{(i,0,1)} + Q_{(i,1,1)} + Q_{(i,\text{stop},1)} = 1\,.$$

Probability estimators Q can be trained in various ways. In this paper, for the sake of consistency, we use logistic regression. We refer to this method using the name LTLS-LR.

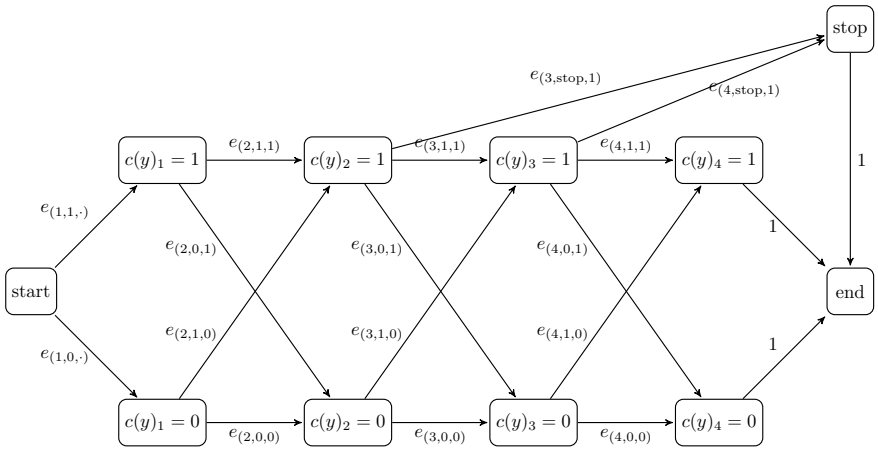

Figure 3. A trellis used in LTLS-LR for $m = 22$. Node *start* is connected to both states of the first bit; both states of the last bit are connected to node *end*. To handle an arbitrary number of classes m we connect to the *stop* node the *up* states at bits corresponding to 1's in binary representation of m. The code corresponding to path $[e_{(1,1,\cdot)}, e_{(2,0,1)}, e_{(3,1,0)}, e_{(4,stop,1)}]$ is $(1, 0, 1)$.

One can also consider a learning procedure, which will make LTLS to be very similar to conditional random fields [33]. We do not discuss this possibility in this paper. Let us, however, remark that in the original paper LTLS has been used with a learning procedure that minimizes a variant of structured hinge loss. From this point of view, LTLS can be seen as an instance of structured support vector machines [34]. In this paper, however, we have decided to investigate a variant of LTLS that is as much as possible similar to SIB and PCT to make a fair comparison between these methods.

7. Experiments

This section presents experimental evaluation of analyzed approaches. We report the results of SIB, LTLS-LR and PCT. For comparison, we also include results of two well-known tree-based algorithms, LOMtree [13], and FastXML [12]. We have run SIB, LTLS-LR and PCT on datasets used in [7], where one can find a comparison of a set of multi-class and multi-label classification algorithms in terms of precision@1, prediction time, and model size. The basic information about the datasets is given in Table 1. Table 2 contains performance results of all methods. The results of LOMtree and FastXML are taken from [7]. We do not report here other methods whose results are reported in the cited paper (including PD-sparse introduced therein), since we focus on graph- and tree-based methods, and other methods apply different approach.

We report precision@1 and the model size. We report precision@1 only since this value was given in [7]. To compare the model size in a comprehensive way, we need to remind several important issues. FastXML uses a sparse representation of the weight vectors. LOMtree (implemented in vw [35]) and PCT use feature hashing, so their model size can be treated as constant (i.e., can be set to all available memory; if it is too small then many conflicts between feature weights occur, which deteriorate their performance). The weight vectors of SIB and LTLS-LR are stored in dense representation. Therefore, in case of SIB and LTLS-LR the model size is proportional to $d \times E$, where E is the number of edges (with probability estimators).

Comparing the results of SIB, LTLS-LR and PCT one can see the trade-off between model size and predictive performance. With a growing size of the model, precision@1 also grows. The results of LTLS-LR are comparable to LOMtree and FastXML. On Dmoz dataset LTLS-LR gets results close to LOMtree, using a much smaller model. Results of PCT are competitive to FastXML.

The training and prediction times of all methods are not comparable due to significant differences in implementation, therefore they are not reported. Moreover, even training and prediction times of methods implemented in a similar manner cannot be clearly compared on benchmark datasets, since they depend not only on the number of labels, but also on the number of features and non-zero features in the dataset. Therefore, to show the dependence on the number of labels of SIB, PCT and LTLS-LR, we report results of an experiment on artificial data. Figure 4 shows the prediction times as a function of the number of labels, from 2^4 to 2^{13}, scaled logarithmically. One can notice, that in case of both LTLS-LR and SIB the dependence of the prediction time on the logarithm of m is definitely linear, but with a different constant. The prediction time of PCT does not depend linearly on the logarithm of m, but the average prediction time is definitely sublinear in m.

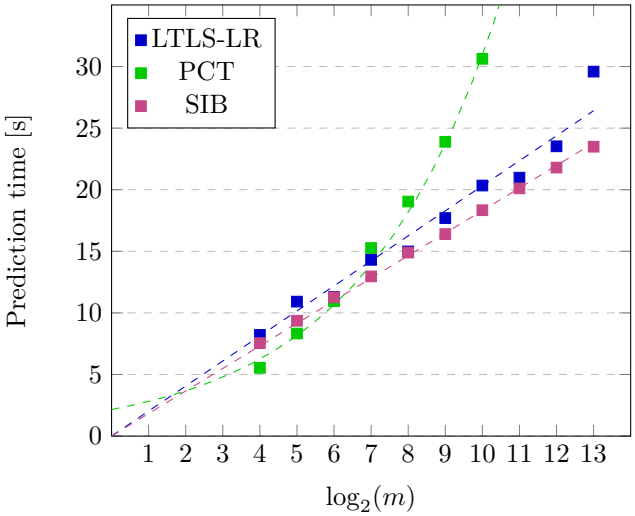

Figure 4. The prediction time, in seconds, of LTLS-LR (blue), PCT (green) and SIB (magenta) on artificial data. The artificial datasets were of the same number of instances and features. The numbers of labels m in the datasets were subsequent powers of 2.

Table 1. Basic statistics of benchmark datasets.

	sector	**aloi.bin**	**Dmoz**	**LSHTC1**
#examples	8658	100000	345068	83805
#features (d)	55197	636911	833484	347255
#labels (m)	105	1000	11947	12294

8. Conclusions

We have introduced a general learning reduction technique that relies on a transformation of a multi-class classification problem to a structured prediction problem. In this way we can control the time and space complexity. By using different codes and dependence structures between elements of the code, we show the trade-off between the predictive performance and the complexity. The preliminary experiments show that algorithms based on the proposed transformation technique can achieve results comparable with the state-of-the-art algorithms that are more costly in terms of time and space. The future work will aim at finding a well-sounded theoretical framework for the reduction of extreme classification to structured prediction problem.

Table 2. Precision@1 and model size [MB] of compared algorithms.

		SIB	LTLS-LR	PCT	LOMtree	FastXML
sector	precision@1	0.8543	0.8616	0.8730	0.8210	0.8490
	model size	6.43	12.06	16.00	17.00	7.00
aloi.bin	precision@1	0.7697	0.8128	0.9088	0.8947	0.9550
	model size	114	209	128	106	992
Dmoz	precision@1	0.1819	0.2082	0.3263	0.2127	0.3840
	model size	215	397	2048	1800	1500
LSHTC1	precision@1	0.0914	0.0950	0.1524	0.1056	0.2166
	model size	272	525	1024	744	308

Acknowledgments

This work has been supported by the Polish National Science Centre under grant no. 2013/09/D/ST6/03917. Large-scale computations have been performed in Poznan Supercomputing and Networking Center.

We would like to thank Wojciech Kotłowski for helpful discussions and valuable insights.

9. References

[1] Viterbi A.J., *Error bounds for convolutional codes and an asymptotically optimum decoding algorithm.* IEEE Transactions on Information Theory, 1967.

[2] Seshadri N., Sundberg C.E.W., *List viterbi decoding algorithms with applications.* IEEE Transactions on Communications, 1994.

[3] Doppa J., Fern A., Tadepalli P., *Hc-search: Learning heuristics and cost functions for structured prediction.* In: *Journal of Artificial Intelligence Research (JAIR)*, 2013.

[4] Jasinska K., Karampatziakis N., *Log-time and log-space extreme classication.* In: *Workshop on Extreme Classification at Neural Information Processing Systems (NIPS)*, 2016.

[5] Dembczyński K., Kotłowski W., Waegeman W., Busa-Fekete R., Hüllermeier E.,*Consistency of probabilistic classifier trees.* In: *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD).* Springer-Verlag 2016.

[6] Jasinska K., Dembczynski K., Busa-Fekete R., Pfannschmidt K., Klerx T., Hüllermeier E., *Extreme F-measure maximization using sparse probability estimates.* In: *International Confernece on Machine Learning (ICML)*, 2016.

[7] Yen I.E.H., Huang X., Ravikumar P., Zhong K., Dhillon I., *Pd-sparse: A primal and dual sparse approach to extreme multiclass and multilabel classification.* In: *International Conference on Machine Learning (ICML)*, 2016.

[8] Bhatia K., Jain H., Kar P., Varma M., Jain P., *Sparse local embeddings for extreme multi-label classification.* In: *Neural Information Processing Systems (NIPS)*, 2015.

[9] Yu H., Jain P., Kar P., Dhillon I.S., *Large-scale multi-label learning with missing labels.* In: *International Conference on Machine Learning (ICML)*, 2014.

[10] Weston J., Bengio S., Usunier N., *Wsabie: Scaling up to large vocabulary image annotation.* In: *International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

[11] Mineiro P., Karampatziakis N., *Fast label embeddings via randomized linear algebra.* In: *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD)*, 2015.

[12] Prabhu Y., Varma M., *FastXML: A fast, accurate and stable tree-classifier for extreme multi-label learning.* In: *Knowledge Discovery and Data Mining (KDD)*, 2014.

[13] Choromanska A., Langford J., *Logarithmic time online multiclass prediction.* In: *Neural Information Processing Systems (NIPS)*, 2015.

[14] Niculescu-Mizil A., Abbasnejad E., *Label filters for large scale multilabel classification.* In: *Workshop on Extreme Classification at the International Confernece on Machine Learning (ICML)*, 2015.

[15] Weston J., Makadia A., Yee H., *Label partitioning for sublinear ranking.* In: *International Conference on Machine Learning (ICML)*, 2013.

[16] Cissé M., Usunier N., Artières T., Gallinari P., *Robust bloom filters for large multilabel classification tasks.* In: *Neural Information Processing Systems (NIPS)*, 2013.

[17] Jasinska,K., Dembczynski K., *Consistent label tree classifiers for extreme multilabel classification.* In: *Workshop on Extreme Classification at the International Confernece on Machine Learning (ICML)*, 2015.

[18] Vijayanarasimhan S., Shlens J., Monga R., Yagnik J., *Deep networks with large output spaces.* In: *Workshop contribution at International Conference on Learning Representation (ICLR)*, 2014.

[19] Shrivastava A., Li P., *Improved asymmetric locality sensitive hashing (ALSH) for maximum inner product search (mips).* In: *Uncertainty in Artificial Intelligence (UAI)*, 2015.

[20] Fagin R., Lotem A., Naor M., *Optimal aggregation algorithms for middleware.* In: *Principles of Database Systems (PODS).* ACM 2001.

[21] Stock M., Pahikkala T., Airola A., De Baets B., Waegeman, W., *Efficient pairwise learning using kernel ridge regression: an exact two-step method.* Computing Research Repository (CoRR), 2016.

[22] Indyk P., Motwani R., *Approximate nearest neighbors: Towards removing the curse of dimensionality.* In: *ACM Symposium on Theory of Computing*, 1998.

[23] Beygelzimer A., Langford J., Zadrozny B., *Machine learning techniques-reductions between prediction quality metrics.* In: *Performance Modeling and Engineering.* Springer-Verlag 2008.

[24] Beygelzimer A., Daumé H., Langford J., Mineiro P., Learning reductions that really work. In: *Proceedings of the IEEE*, 2016.

[25] Dietterich T., Bakiri G., *Solving multiclass learning problems via error-correcting output codes.* Journal of Machine Learning Research (JMLR), 1996.

[26] Huffman D., *A method for the construction of minimum-redundancy codes.* Proceedings of the Institute of Radio Engineers (IRE), 1952.

[27] Weinberger K., Dasgupta A., Langford J., Smola A., Attenberg J., *Feature hashing for large scale multitask learning.* In: *International Conference on Machine Learning (ICML)*, ACM, 2009.

[28] Dembczyński K., Waegeman W., Cheng W., Hüllermeier E., *An analysis of chaining in multi-label classification.* In: *European Conference on Artificial Intelligence (ECAI)*, 2012.

[29] Mena D., Montanes E., Quevedo J.R., del Coz J.J., *Using a* for inference in probabilistic classifier chains.* In: *International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

[30] Beygelzimer A., Langford J., Lifshits Y., Sorkin G.B., Strehl A.L., *Conditional probability tree estimation analysis and algorithms.* In: *Uncertainty in Artificial Intelligence (UAI)*, 2009.

[31] Fox J., *Applied regression analysis, linear models, and related methods.* Sage, 1997.

[32] Morin F., Bengio Y., *Hierarchical probabilistic neural network language model.* In: *Artificial Intelligence and Statistics Conference (AISTATS)*, 2005, pp. 246–252.

[33] Lafferty J.D., McCallum A., Pereira F.C.N., *Conditional random fields: Probabilistic models for segmenting and labeling sequence data.* In: *International Conference on Machine Learning (ICML)*, 2001.

[34] Tsochantaridis Y., Joachims T., Hofmann T., Altun Y., *Large margin methods for structured and interdependent output variables.* Journal of Machine Learning Research (JMLR), 2005.

[35] Langford J., Strehl A., Li L., *Vowpal wabbit*, 2007 `http://mloss.org/software/view/53/`.