Efficient Loss-Based Decoding on Graphs for Extreme Classification

Itay Evron
Efficient Loss-Based Decoding on Graphs for Extreme Classification

Research Thesis

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

Itay Evron

Submitted to the Senate of the Technion — Israel Institute of Technology
Cheshvan 5779 Haifa November 2018
The research thesis was done under the supervision of Prof. Koby Crammer in the Department of Computer Science.

Acknowledgments

I would like to thank my advisor Prof. Koby Crammer for his guidance. It was a privilege to work with such a bright and resourceful researcher. I also thank Koby for funding my travel to the 2017 Data Science Summer School at the École Polytechnique.

I thank Dr. Daniel Soudry for funding my travel to NeurIPS 2018.

I thank my coauthor, PhD candidate Edward Moroshko, for his valuable insights and advice.

Finally, I wish to give special thanks to my mother Efrat, for her unconditional love and support.

The generous financial help of the Technion is gratefully acknowledged.
Publications

Parts of this work were published in
Efficient Loss-Based Decoding on Graphs For Extreme Classification.
# Contents

**Abstract**

1

**Abbreviations and Notations**

2

1 Introduction

3

1.1 Problem setting

3

1.2 Multiclass classification

3

1.2.1 One vs Rest (OVR)

4

1.2.2 Error Correcting Output Coding (ECOC)

4

1.2.3 Loss-based decoding

5

1.3 Extreme classification

6

1.4 LTLS

8

1.4.1 Path assignment

8

1.4.2 Limitations

9

2 Our work

11

2.1 Efficient loss-based decoding

11

2.2 Wide-LTLS (W-LTLS)

14

2.2.1 Time and space complexity analysis

15

2.2.2 Wider graphs induce sparse models

17

2.2.3 Connection to other coding schemes

18

2.3 Experiments

20

2.3.1 Loss-based decoding

20

2.3.2 Multiclass test accuracy

23

2.3.3 Multiclass training error

26

2.3.4 Exploiting the sparsity of the datasets

28

2.3.5 Comparison to BCH codes

30
3 Discussion and Future Work

3.1 The average binary loss $\varepsilon$ can also be controlled . . . . . . . . . . . 32
3.2 Error correcting codes can also induce sparse models . . . . . . . . . . . 33
3.3 Improving the minimum Hamming distance $\rho$ . . . . . . . . . . . . . . . 33
3.4 Exploiting the graph’s structure to ease the binary subproblems . . 34
3.5 Edges weighting scheme . . . . . . . . . . . . . . . . . . . . . . 35
3.6 Other types of graphs and graph algorithms . . . . . . . . . . . . . . . 35

A Extension to arbitrary K

A.1 Graph construction . . . . . . . . . . . . . . . . . . . . . . . . 36
A.2 Loss-based decoding generalization . . . . . . . . . . . . . . . 38

B Experiments appendix

B.1 Experimental results of the multiclass test accuracy experiments . 42
B.2 Experimental results of the sparsity experiments . . . . . . . . . . 45
List of Figures

1.1 Path codeword representation ........................................ 9
1.2 Illustration of two closest paths .................................... 9

2.1 Trellis graphs for different slice widths ($b$) .................. 15
2.2 Multiclass test accuracy vs model size using different loss functions for decoding ........................................ 21
2.3 The average absolute margin vs the slice width $b$. .......... 22
2.4 Multiclass test accuracy vs model size and prediction time .......................... 23
2.5 Average binary loss and multiclass training error vs model size .................... 26
2.6 Multiclass test accuracy vs model size for sparse models .................. 28
2.7 Percentage of non-zero weights before and after pruning ........ 29
2.8 Comparison of W-LTLS to BCH codes .......................... 31

A.1 Illustration of the graph construction algorithm ............. 37
A.2 Illustration of a constructed graph .............................. 39
List of Tables

2.1 Datasets used in the experiments. ........................................ 20

B.1 Tabular results of the multiclass test accuracy experiments ........ 44
B.2 Tabular results of the multiclass test accuracy experiments for sparse models ....................................................... 46
Abstract

Classification is a popular task in the field of machine learning, in which learning algorithms are required to map instances to labels from a finite set of labels. In binary classification problems, the label set consists of only two labels. For instance, given an email we would like to determine whether it is spam. Binary tasks are well studied and often have simple and intuitive solutions.

In multiclass classification problems, the label set contains three or more labels. For example, given an image of an animal, we would like to determine which kind of animal appears in it. Due to the simplicity of binary classification learning algorithms, many solutions for multiclass problems reduce single multiclass problems to many binary subproblems.

Extreme classification problems, are multiclass problems, where the label set becomes extremely large. In this regime, the widely practiced reductions to binary subproblems become infeasible due to their excessive space and time complexity requirements, which are often linear in the number of classes.

We build on a recent extreme classification framework with logarithmic time and space [19], and on a well-known general approach for error correcting output coding (ECOC) with loss-based decoding [1], and introduce a flexible and efficient approach accompanied by theoretical bounds.

Our framework employs output codes induced by graphs, for which we show how to perform efficient loss-based decoding. The suggested framework is a time and space efficient scheme for both training and inference. Moreover, it allows using any loss function for decoding, potentially improving accuracy.

In addition, our framework offers a tradeoff between accuracy, model size and prediction time. We show how to find the sweet spot of this tradeoff using only the training data.

Our experimental study demonstrates the validity of our assumptions and claims, and shows that our method is competitive with state-of-the-art algorithms.
### Abbreviations and Notations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVR</td>
<td>One vs rest (see Section 1.2.1).</td>
</tr>
<tr>
<td>ECOC</td>
<td>Error correcting output coding (see Section 1.2.2).</td>
</tr>
<tr>
<td>LTLS</td>
<td>Log time log space (see Section 1.4).</td>
</tr>
<tr>
<td>W-LTLS</td>
<td>Wide-LTLS (see Section 2.2).</td>
</tr>
<tr>
<td>$K$</td>
<td>Number of classes in a multiclass classification problem.</td>
</tr>
<tr>
<td>$d$</td>
<td>The dimension of the input data.</td>
</tr>
<tr>
<td>$b$</td>
<td>A trellis graph’s slice width.</td>
</tr>
<tr>
<td>$\delta(v)$</td>
<td>The shortest distance from the source vertex of a trellis graph to the vertex $v$.</td>
</tr>
<tr>
<td>$S(e)$</td>
<td>The set of edges outgoing from the same vertical slice as the edge $e=(u,v)$. Formally, $S(e) = {(u',u''): \delta(u') = \delta(u)}$.</td>
</tr>
<tr>
<td>$w(P)$</td>
<td>Additive path weight. Formally, $w(P) = \sum_{e \in P} w(e)$.</td>
</tr>
<tr>
<td>$\rho(a,b)$</td>
<td>Hamming distance between two codewords in a coding matrix $M = {-1,1}^K \times \ell$. Formally, $\rho(a,b) = \sum_{j=1}^\ell 1 - M_{a,j}M_{b,j}$.</td>
</tr>
<tr>
<td>$\rho$</td>
<td>Minimum Hamming distance of a code matrix $M$. Formally, $\rho = \min_{a \neq b} \rho(a,b)$.</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>Average binary loss on the training set w.r.t learned binary functions ${f_j}<em>{j=1}^\ell$, a coding matrix $M$, and a loss function $L$. Formally, $\varepsilon = \frac{1}{m\ell} \sum</em>{i=1}^m \sum_{j=1}^\ell L(M_{y_i,j}f_j(x_i))$.</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Problem setting

We consider multiclass classification with $K$ classes, where $K$ is extremely large. Given a training set of $m$ examples $(x_i, y_i)$ for $x_i \in \mathcal{X} \subseteq \mathbb{R}^d$ and $y_i \in \mathcal{Y} = \{1, ..., K\}$ our goal is to learn a mapping from $\mathcal{X}$ to $\mathcal{Y}$. We focus on the 0/1 loss and evaluate the performance of the learned mapping by measuring its accuracy on a test set – i.e., the fraction of instances with a correct prediction. Formally, the accuracy of a mapping $h : \mathcal{X} \rightarrow \mathcal{Y}$ on a set of $n$ pairs, $\{(x_i, y_i)\}_{i=1}^n$, is defined as $\frac{1}{n} \sum_{i=1}^n 1_{h(x_i)=y_i}$, where $1_z$ equals 1 if the predicate $z$ is true, and 0 otherwise.

1.2 Multiclass classification

Multiclass classification is the task of assigning instances with a category or class from a finite set. Its numerous applications range from finding a topic of a news item, via classifying objects in images, via spoken words detection, to predicting the next word in a sentence. Our ability to solve multiclass problems with larger and larger sets improves with computation power.

Research throughout the years focused mainly on directly solving the multiclass tasks (e.g. [7][11][25][34]), and on reducing these tasks to binary ones (also called binarization; e.g., one vs rest [see Section 1.2.1], one vs one [16], ECOC [see Section 1.2.2], etc.).

In this work we focus on multiclass tasks, as opposed to multilabel tasks, where each sample belongs to one or more classes. I.e., the learned mapping is from $\mathcal{X}$...
to \( \mathcal{Y} = \{ y : y \subseteq \{1, \ldots, K\}, y \neq \emptyset \} \).

### 1.2.1 One vs Rest (OVR)

One of the most widely used reductions from multiclass classification problems to binary ones, is One vs Rest (OVR in short; sometimes called one vs all [OVA], or one against all [OAA]). In this approach, a problem of classifying \( K \) classes is reduced to \( K \) binary classification subproblems, where the \( k \)-th classifier simply learns how to distinguish the \( k \)-th class from the rest of the classes. During inference, each of the \( K \) trained classifiers yields a score (roughly speaking – a classification confidence), and the predicted class is the class belonging to the classifier with the maximal score.

The OVR scheme was empirically shown to give very accurate results [28]. We note in passing that OVR somewhat resembles the common practice in neural networks, where the last layer tries to linearly separate all \( K \) classes. During inference, each output neuron out of \( K \) such neurons, computes a score (arising from the binary-separation margin), and the class corresponding to the output neuron with the maximal score is predicted.

### 1.2.2 Error Correcting Output Coding (ECOC)

Dietterich and Bakiri [13] employed ideas from coding theory [22] to create Error Correcting Output Coding (ECOC) – a reduction from a multiclass classification problem to multiple binary classification subproblems. In this scheme, each class is assigned with a (distinct) binary codeword of \( \ell \) bits (with values in \( \{-1, +1\} \)). The \( K \) codewords create a matrix \( M \in \{-1, +1\}^{K \times \ell} \) whose rows are the codewords and whose columns induce \( \ell \) partitions of the classes into two subsets. Each of these partitions induces a binary classification subproblem. We denote by \( M_k \) the \( k \)-th row of the matrix, and by \( M_{k,j} \) its \((k, j)\) entry. In the \( j \)-th partition, class \( k \) is assigned with the binary label \( M_{k,j} \).

ECOC introduces redundancy \(^1\) in order to acquire error-correcting capabilities such as a minimum Hamming distance between codewords. The Hamming distance between two codewords \( M_a, M_b \) is defined as \( \rho(a, b) \triangleq \sum_{j=1}^{\ell} \frac{1-M_{a,j}M_{b,j}}{2} \), and the minimum Hamming distance of \( M \) is \( \rho = \min_{a \neq b} \rho(a, b) \). A high minimum distance of the code matrix potentially allows overcoming binary classification errors during inference time.

\(^1\)Distributed output coding without redundancy was previously presented in [30]
At training time, this scheme generates $\ell$ binary classification training sets of the form $\{x_i, M_{y_j,i}\}_{i=1}^m$ for $j = 1, \ldots, \ell$, and executes some binary classification learning algorithm that returns $\ell$ classifiers, each trained on one of these sets. We assume these classifiers are margin-based, that is, each classifier is a real-valued function, $f_j : \mathcal{X} \to \mathbb{R}$, whose binary prediction for an input $x$ is $\text{sign}(f_j(x))$. The binary classification learning algorithm defines a margin-based loss $L : \mathbb{R} \to \mathbb{R}_+$, from the reals to the non-negatives, and minimizes the average loss over the induced set. Formally, $f_j = \arg \min_{f \in \mathcal{F}} \frac{1}{m} \sum_{i=1}^m L(M_{y_j,i}f(x_i))$, where $\mathcal{F}$ is a class of functions, such as the class of bounded linear functions. Few well known loss functions are the hinge loss $L(z) \triangleq \max(0, 1 - z)$, used by SVM, its square, the log loss $L(z) \triangleq \log(1 + e^{-z})$ used in logistic regression, and the exponential loss $L(z) \triangleq e^{-z}$ used in AdaBoost [29].

Once these classifiers are trained, a straightforward inference is performed. Given an input $x$, the algorithm first applies the $\ell$ functions on $x$ and computes a $\{\pm 1\}$-vector of size $\ell$, that is $(\text{sign}(f_1(x)) \ldots \text{sign}(f_\ell(x)))$. Then, the class $k$ which is assigned to the codeword closest in Hamming distance to this vector is returned. This inference scheme is often called Hamming decoding.

### 1.2.3 Loss-based decoding

The Hamming decoding used in ECOC, uses only the binary prediction of the binary learners, ignoring the confidence each learner has in its prediction per input. Allwein et al. [1] showed that this margin or confidence holds valuable information for predicting a class $y \in \mathcal{Y}$, and proposed the loss-based decoding framework for ECOC\(^2\). In loss-based decoding, the margin is incorporated via the loss function $L(z)$. Specifically, the class predicted is the one minimizing the total loss

$$k^* = \arg \min_k \sum_{j=1}^{\ell} L(M_{k,j}f_j(x)). \tag{1.1}$$

They [1] also developed error bounds and showed theoretically and empirically that loss-based decoding outperforms Hamming decoding.

One drawback of their method is that given a loss function $L$, loss-based de-
coding requires an exhaustive evaluation of the total loss for each codeword $M_k$ (each row of the coding matrix). This implies a decoding time at least linear in $K$, making it intractable when $K$ is very large. We address this problem below.

### 1.3 Extreme classification

Recent research focuses on **extreme classification** – multiclass classification where the number of possible classes $K$ is extremely large.

Extreme classification was studied extensively in the past decade. It faces unique challenges, amongst which is the model size of its designated learning algorithms. An extremely large model size often implies long training and test times, as well as excessive space requirements. When the number of classes $K$ is extremely large, the inference time complexity should be sublinear in $K$ for the classifier to be useful. In this work, we focus on learning schemes with sublinear time and space.

Naturally, hierarchical classification models are very popular for extreme classification tasks. Such models can be seen as decision trees allowing inference time complexity linear in the tree height, which can be $O \left( \log K \right)$ if the tree is (approximately) balanced. Many such models (see recent works and the references therein [3, 8, 12]) have been proposed throughout the years, showing competitive results even in comparison to OVR. Despite having a sublinear inference time complexity, these models have some noticeable shortcomings. They often require training $O \left( K \right)$ classifiers, implying a training time at least linear in $K$. Consequently, the space complexity is $O \left( K \right)$, even at inference time.

The aforementioned **Error Correcting Output Coding (ECOC)** approach (see Section 1.2.2) seems promising for extreme classification, as it potentially allows a very compact representation of the label space with $K$ codewords of length $\ell = O \left( \log K \right)$. Indeed, many works concentrate on utilizing ECOC for extreme classification. Some formulate dedicated optimization problems to find ECOC matrices suitable for extreme classification [9] and others focus on learning better binary learners [23]. Recently, Huang et al. [18] proposed to use hashing functions to encode classes as $B$-ary codewords instead of binary ones, inducing many multiclass subproblems (when $B$ is not too large, a simple learner, such as a multiclass logistic regression, can potentially achieve high accuracy on the induced multiclass subproblems).

However, very little attention has been given to the decoding time complexity. In the multiclass regime where only one class is the correct class, many of these
works are forced to use exact (i.e. not approximated) decoding algorithms which often require \( O(K\ell) \) time in the worst-case [20]. Norouzi et al. [26] proposed a fast exact search nearest neighbor algorithm in the Hamming space, which for coding matrices suitable for extreme classification can achieve \( o(K) \) time complexity, but not \( O(\log K) \). These algorithms are often limited to binary (dense) matrices and hard decoding (i.e. Hamming decoding). Other algorithms employing soft decoding (e.g. loss-based decoding [1] or maximum a-posteriori decoding [MAP] used in the aforementioned [18]) usually strictly require \( O(K\ell) \) time for inference even in the average-case. Some approaches [21] utilize graphical processing units in order to find the nearest neighbor in Euclidean space, which can be useful for soft decoding, but might be too demanding for weaker devices. In our work we keep the time complexity of any loss-based decoding logarithmic in \( K \).

Moreover, most existing ECOC methods employ coding matrices with higher minimum distance \( \rho \), but with balanced binary subproblems. In Section 2.2 we explain how our ability of inducing less balanced subproblems is beneficial both for the learnability of these subproblems, and for the post-pruning of learned weights to create sparse models.

It is also worth mentioning that many of the ECOC-based works (like randomized or learned codes [9,37]) require storing the entire coding matrix even during inference time. Hence, the additional space complexity needed only for decoding during inference is \( O(K\log K) \).

Both LTLS and W-LTLS do not directly use the coding matrix for decoding the binary predictions and only require a mapping from code to label (e.g. a binary tree). This reduces the additional space complexity to \( O(\log K) \).

Another line of research focused on label-embedding methods [8,31,33,36]. These methods try to exploit label correlations and project the labels onto a low-dimensional space, reducing training and prediction time. However, the low-rank assumption usually leads to an accuracy degradation.

Linear methods were also the focus of some recent works [2,14,35]. They learn a linear classifier per label and incorporate sparsity assumptions or perform distributed computations. However, the training and prediction complexities of these methods do not scale gracefully to datasets with a very large number of labels. Using a similar post-pruning approach and independent (i.e. not joint) learning of the subproblems, W-LTLS is also capable of exploiting sparsity and learn in parallel.

We next describe a recently proposed approach for extreme classification, upon
which we build our work.

1.4 LTLS

A recent approach for extreme classification, proposed by Jasinska and Karampatziakis [19], performs training and inference in time and space logarithmic in \( K \) by embedding the \( K \) classes into \( K \) paths of a directed-acyclic trellis graph \( T \), built compactly with \( \ell = \mathcal{O}(\log K) \) edges. We denote the sets of vertices and edges by \( V \) and \( E \) (respectively). A multiclass model is defined using \( \ell \) functions from the feature space to the reals, \( w_j(x) \), one function per edge in \( E = \{e_j\}_{j=1}^\ell \). Given an input, the algorithm assigns weights to the edges, and computes the heaviest path using the Viterbi [32] algorithm in \( \mathcal{O}(|E|) = \mathcal{O}(\log K) \) time. It then outputs the class (from \( Y \)) assigned to the heaviest path.

Jasinska and Karampatziakis [19] proposed to train the model in an online manner. The algorithm maintains \( \ell \) functions \( f_j(x) \) and works in rounds. In each training round a specific input-output pair \((x_i, y_i)\) is considered, the algorithm performs inference using the \( \ell \) functions to predict a class \( \hat{y}_i \), and the functions \( f_j(x) \) are modified to improve the overall prediction for \( x_i \) according to \( y_i, \hat{y}_i \). The inference performed during train and test times, includes using the obtained functions \( f_j(x) \) to compute the weights \( w_j(x) \) of each input, by simply setting \( w_j(x) = f_j(x) \).

Specifically, they used margin-based learning algorithms, where \( f_j(x) \) is the margin of a binary prediction.

Our first contribution is the observation that the LTLS approach can be thought of as an ECOC scheme, in which the codewords (rows) represent paths in the trellis graph, and the columns correspond to edges on the graph. Figure 1.1 illustrates how a codeword corresponds to a path on the graph.

It might seem like this approach can represent only numbers of classes \( K \) which are powers of 2. However, in Appendix A.1 we show how to create trellis graphs with exactly \( K \) paths, for any \( K \in \mathbb{N} \).

1.4.1 Path assignment

LTLS requires a bijective mapping between paths to classes and vice versa. It was proposed in [19] to employ a greedy assignment policy suitable for online learning, where during training, a sample whose class is yet unassigned with a path, is assigned with the heaviest unassigned path. Instead, we consider a naive
random assignment between paths and classes.

We extend the discussion on path assignment policies as a direction for future work in Section 3.4.

1.4.2 Limitations

The elegant LTLS construction suffers from two limitations:

1. **Difficult induced binary subproblems**: The induced binary subproblems are hard, especially when learned with linear classifiers. Each path contains one of the four edges between every two adjacent vertical slices. Therefore, each edge is used by $\frac{1}{4}$ of the classes, inducing a $\frac{1}{4}K$-vs-$\frac{3}{4}K$ subproblems. Also, the edges connected to the source or sink induce $\frac{1}{2}K$-vs-$\frac{1}{2}K$ subproblems. In both cases classes are split into two groups, almost arbitrarily, with no clear semantic interpretation for that partition. For comparison, in 1-vs-Rest (OVR) the induced subproblems are considered much simpler as they require classifying only one class vs the rest (meaning they are much less balanced).

2. **Low minimum distance**: In the LTLS trellis architecture, every path has another (closest) path within 2 edge deletions and 2 edge insertions (see Figure 1.2). Thus, the minimum Hamming distance in the underlying coding

---

3 A similar observation is given in Section 6 of Allwein et al. regarding OVR.
matrix is small: $\rho = 4$, which might imply a poor error correcting capability (see Figure 1.2). The OVR coding matrix also suffers from a small minimum distance ($\rho = 2$), but as we explained, the induced subproblems are very simple, allowing a higher classification accuracy in many cases.

We focus on improving the multiclass accuracy by tackling the first limitation, namely making the underlying binary subproblems easier. Addressing the second limitation is deferred to future work.
Chapter 2
Our work

2.1 Efficient loss-based decoding

We now introduce another contribution – a new algorithm performing efficient loss-based decoding (inference) for any loss function by exploiting the structure of trellis graphs. Similarly to [19], our decoding algorithm performs inference in two steps. First, it assigns (per input $x$ to be classified) weights $\{w_j(x)\}_{j=1}^\ell$ to the edges $\{e_j\}_{j=1}^\ell$ of the trellis graph. Second, it finds the shortest path (instead of the heaviest) $P^*_k$ by an efficient dynamic programming (Viterbi) algorithm and predicts the class $k^*$. Unlike [19], our strategy for assigning edge weights ensures that for any class $k$, the weight of the path assigned to this class, $w(P_k) \triangleq \sum_{j:e_j \in P_k} w_j(x)$, equals the total loss $\sum_{j=1}^\ell L(M_{k,j}f_j(x))$ for the classified input $x$. Therefore, finding the shortest path on the trellis graph is equivalent to minimizing the total loss, which is the aim in loss-based decoding. In other words, we design a new weighting scheme that links loss-based decoding to the shortest path in a graph.

We now describe our algorithm in more detail for the case when the number of classes $K$ is a power of 2 (see Appendix A.2 for extension to arbitrary $K$). Consider a directed edge $e_j \in E$ and denote by $(u_j, v_j)$ the two vertices it connects. Denote by $S(e_j)$ the set of edges outgoing from the same vertical slice as $e_j$. Formally, $S(e_j) = \{(u, u') : \delta(u) = \delta(u_j)\}$, where $\delta(v)$ is the shortest distance from the source vertex to $v$ (in terms of number of edges). For example, in Figure 1.1 $S(e_0) = S(e_1) = \{e_0, e_1\}$, $S(e_2) = S(e_3) = S(e_4) = S(e_5) = \{e_2, e_3, e_4, e_5\}$.

Given a loss function $L(z)$ and an input instance $x$, we set the weight $w_j$ for edge
as following,

\[ w_j(x) = L(1 \times f_j(x)) + \sum_{j': e_{j'} \in S(e_j) \setminus \{e_j\}} L((-1) \times f_{j'}(x)) . \tag{2.1} \]

For example, in Figure 1.1 we have,

\[ w_0(x) = L(1 \times f_0(x)) + L((-1) \times f_1(x)) \]
\[ w_2(x) = L(1 \times f_2(x)) + L((-1) \times f_3(x)) + L((-1) \times f_4(x)) + L((-1) \times f_5(x)) . \]

The next theorem states that for our choice of weights, finding the shortest path in the weighted graph is equivalent to loss-based decoding. Thus, algorithmically we can enjoy fast decoding (i.e. inference), and statistically we can enjoy better performance by using loss-based decoding.

**Theorem 2.1.** Let \( L(z) \) be any loss function of the margin. Let \( T \) be a trellis graph with an underlying coding matrix \( M \). Assume that for any \( x \in \mathcal{X} \) the edge weights are calculated as in Eq. (2.1). Then, the weight of any path \( P_k \) equals to the loss suffered by predicting its corresponding class \( k \), i.e.

\[ w(P_k) = \sum_{j=1}^{\ell} L(M_{k,j} f_j(x)) . \]

**Proof** For any class \( k \) we have,

\[
\begin{align*}
    w(P_k) &= \sum_{j \in \mathcal{P}_k} w_j(x) = \sum_{j \in \mathcal{P}_k} \left[ L(1 \times f_j(x)) + \sum_{j': e_{j'} \in S(e_j) \setminus \{e_j\}} L((-1) \times f_{j'}(x)) \right] \\
    &= \sum_{j \in \mathcal{P}_k} \left[ \sum_{j' \in \mathcal{S}(e_j)} L(M_{k,j'} \times f_{j'}(x)) \right] = \sum_{j=1}^{\ell} L(M_{k,j} f_j(x)).
\end{align*}
\]

The third equality in the proof follows from the path codeword representation, see for example Figure 1.1.

In the next lemma we claim that LTLS decoding is a special case of loss-based decoding with the squared loss function.

**Lemma 2.2.** Denote the squared loss function by \( L_{sq}(z) \triangleq (1 - z)^2 \). Given a trellis graph represented using a coding matrix \( M \in \{-1, +1\}^{K \times \ell} \), and \( \ell \) func-
tions \( f_j(x) \), for \( j = 1 \ldots \ell \), the decoding method of LTLS (mentioned in Section 1.4) is a special case of loss-based decoding with the squared loss, that is \( \arg\max_k w(P_k) = \arg\min_k \left\{ \sum_j L_{sq}(M_{k,j} f_j(x)) \right\} \).

**Proof** Indeed, for LTLS we have,

\[
\arg\max_k w(P_k) = \arg\max_k \left\{ \sum_{j, \omega_j \in P_k} w_j(x) \right\} = \arg\max_k \left\{ \sum_{j; M_{k,j} = 1} f_j(x) \right\}
\]

\[
= \arg\max_k \left\{ 2 \sum_{j; M_{k,j} = 1} f_j(x) - \sum_{j} f_j(x) \right\} = \arg\max_k \left\{ \sum_{j; M_{k,j} = 1} f_j(x) - \sum_{j; M_{k,j} = -1} f_j(x) \right\}
\]

\[
= \arg\max_k \left\{ \sum_j M_{k,j} f_j(x) \right\} = \arg\max_k \left\{ \sum_j M_{k,j}^2 + \sum_j f_j(x) - \sum_j (M_{k,j} - f_j(x))^2 \right\}
\]

\[
= \arg\min_k \left\{ \sum_j (M_{k,j} - f_j(x))^2 \right\} = \arg\min_k \left\{ \sum_j (1 - M_{k,j} f_j(x))^2 \right\}
\]

\[
= \arg\min_k \left\{ \sum_j L_{sq}(M_{k,j} f_j(x)) \right\} .
\]

Note that we used the fact that \( M_{k,j} \in \{-1, +1\} \).

In Section 2.3.1, we show empirically that the above efficient loss-based decoding algorithm, which generalizes LTLS from the squared loss to any loss function, can be used to enhance the multiclass accuracy of previously trained binary functions.

We show next how to build on the framework of Allwein et al. [1] to design graphs that achieve a better multiclass accuracy.
2.2 Wide-LTLS (W-LTLS)

Allwein et al. [1] derived error bounds for loss-based decoding with any convex loss function $L$. They showed that the training multiclass error with loss-based decoding is upper bounded by:

$$\frac{\ell \times \varepsilon}{\rho \times L(0)}$$

(2.2)

where $\rho$ is the minimum Hamming distance of the code and

$$\varepsilon = \frac{1}{m\ell} \sum_{i=1}^{m} \sum_{j=1}^{\ell} L \left( M_{y_{i,j}, f_{j}(x_{i})} \right)$$

(2.3)

is the average binary loss on the training set of the learned functions $\{ f_{j} \}_{j=1}^{\ell}$ with respect to a coding matrix $M$ and a loss $L$. One approach to reduce the bound, and thus hopefully also the multiclass training error (and under some conditions also the test error) is to reduce the total error of the binary problems $\ell \times \varepsilon$. We now show how to achieve this by generalizing the LTLS framework to a more flexible architecture which we call W-LTLS.

Motivated by the error bound of [1], we propose a generalization of the LTLS model. By increasing the slice widths of the trellis graph, and consequently increasing the number of edges between adjacent vertical slices, the induced subproblems become less balanced and potentially easier to learn (see Remark [3]). For simplicity we choose a fixed slice width $b \in \{2, \ldots, K\}$ for the entire graph (e.g. see Figure 2.1). In such a graph, most of the induced subproblems are $\frac{1}{b^2} K\text{-vs-rest}$ (corresponding to edges between adjacent slices) and some are $\frac{1}{b} K\text{-vs-rest}$ (the ones connected to the source or to the sink). As $b$ increases, the graph representation becomes less compact and requires more edges, i.e. $\ell$ increases. However, the induced subproblems potentially become easier, improving the multiclass accuracy. This suggests that our model allows an accuracy vs model size tradeoff.

In the special case where $b = K$ we get the widest graph containing $2K$ edges (see Figure 2.1). All the subproblems are now $1\text{-vs-rest}$: the $k$th path from the source to the sink contains two edges (one from the source and one to the sink) which are not a part of any other path. Thus, the corresponding two columns in the underlying coding matrix are identical – having 1 at their $k$th entry and $(-1)$ at the

---

1 Code is available online at https://github.com/ievron/wltls/
rest. This implies that the distinct columns of the matrix could be rearranged as the diagonal coding matrix corresponding to OVR, making our model when $b = K$ an implementation of OVR.

In Section 2.3 we show empirically that W-LTLS improves the multiclass accuracy of LTLS. We also show that the binary subproblems indeed become easier, i.e. we observe a decrease in the average binary loss $\epsilon$, lowering the bound in (2.2). Note that the denominator $\rho \times L(0)$ is left untouched – the minimum distance of the coding matrices corresponding to different architectures of W-LTLS is still 4, like in the original LTLS model (see Section 1.4.2).

### 2.2.1 Time and space complexity analysis

W-LTLS requires training and storing a binary learner for every edge. For most linear classifiers (with $d$ parameters each) we get a total model size complexity and an inference time complexity of $O(d|E|)$. Hence, we first turn to analyze the number of edges.

**Lemma 2.3.**

The number of vertices in the inner slices is at most $(\lfloor \log_b K \rfloor + 1) b$.

**Proof** Follows immediately from the construction in Appendix A.1.
Corollary 2.4. The number of edges is upper bounded:

$$|E| \leq (b + 1) (\lceil \log_b K \rceil + 1) b + b = \mathcal{O} \left( \frac{b^2 \log b}{\log b} \log K \right).$$

**Proof** Each vertex in the inner slices can have at most $b + 1$ outgoing edges. We use Lemma 2.3 and count also the $b$ edges outgoing from the source.

The total model size complexity is hence $\mathcal{O} \left( d \cdot |E| \right) = \mathcal{O} \left( \frac{b^2 \log b}{\log b} \log K \right)$.

Inference consists of four steps:

1. Computing the value (margin) of all binary functions on the input $x$. This requires $\mathcal{O} \left( d \cdot |E| \right) = \mathcal{O} \left( \frac{d \cdot b^2 \log b}{\log b} \log K \right)$ time.

2. Computing the edge weights $\{w_i(x)\}_{i=1}^\ell$ as explained in Section 2.1. This can be performed in $\mathcal{O} \left( |V| + |E| \right) = \mathcal{O} \left( \frac{b^2 \log b}{\log b} \log K \right)$ time using a simple dynamic programming algorithm (e.g. implementing back recursion).

3. Finding the shortest path in the trellis graph with respect to $\{w_i(x)\}_{i=1}^\ell$ using the Viterbi algorithm in $\mathcal{O} \left( |V| + |E| \right) = \mathcal{O} \left( \frac{b^2 \log b}{\log b} \log K \right)$ time.

4. Decoding the shortest path to a class. As explained in Section 1.4.1 the inference requires a mapping function from path to code. Using data structures such as a binary tree, this can be performed in a $\mathcal{O} \left( |E| \right) = \mathcal{O} \left( \frac{b^2 \log b}{\log b} \log K \right)$ time complexity.

Moreover, many extreme classification datasets are sparse – the average number of non-zero features in a sample is $d_e \ll d$. The inference time complexity thus decreases to $\mathcal{O} \left( d_e \frac{b^2 \log b}{\log b} \log K \right)$.

We get that the total inference time complexity is $\mathcal{O} \left( d \cdot |E| \right) = \mathcal{O} \left( d_e \frac{b^2 \log b}{\log b} \log K \right)$. This is a significant advantage: while inference with loss-based decoding for general matrices requires $\mathcal{O} \left( d_e \ell + K \ell \right)$ time, our model performs it in only $\mathcal{O} \left( d_e \ell + \ell \right) = \mathcal{O} \left( d_e \ell \right)$ by exploiting the structure of the trellis graphs.

Since training requires learning $\ell$ binary subproblems, the training time complexity is also sublinear in $K$. These subproblems can be learned separately on $\ell$ cores, leading to major speedups.

\footnote{Clearly, when $b \approx \sqrt{K}$ our method cannot be regarded as sublinear in $K$ anymore. However, our empirical study shows that high accuracy can be achieved using much smaller values of $b$.}
2.2.2 Wider graphs induce sparse models

The high sparsity typical to extreme classification datasets (e.g. the Dmoz dataset has \( d = 833,484 \) features, but on average only \( d_e = 174 \) of them are non-zero), is heavily exploited by previous works such as PD-Sparse [14], PPDSparse [35], and DiSMEC [2], which all learn sparse OVR models (mentioned before in Section 1.3 as linear methods).

Indeed, we find that for sparse datasets, our algorithm typically learns a model with a low percentage of non-zero weights. Moreover, the percentage of non-zero decreases significantly as the slice width \( b \) is increased (see Appendix B.2). This allows us to employ a simple post-pruning of the learned weights. For some threshold value \( \lambda \), we set to zero all learned weights in \([−\lambda, \lambda]\), yielding a sparse model. Similar approaches were taken by [2][14][19] either explicitly or implicitly.

In Section 2.3.4 we show that the above scheme successfully yields highly sparse models.
2.2.3 Connection to other coding schemes

We note that other codes could also be considered (as already mentioned in Section 1.3). In coding theory, there is a wide usage of codes with much better error correction properties, i.e. higher minimum distances $\rho$. This potentially decreases the upper bound presented in (2.3), and may significantly improve the accuracy results achieved by LTLS and W-LTLS.

Indeed, Dietterich and Bakiri [13] have already considered such a code – the BCH code (Bose-Chaudhuri-Hocquenghem; [6, 17]). For the same coding matrix widths ($\ell$), BCH codes ensure notably higher minimum distances $\rho$ than the fixed minimum distance in the underlying coding matrix of W-LTLS ($\rho = 4$). That being said, the binary subproblems BCH codes induce are perfectly balanced, i.e. each bit partition the label set into two even subsets ($\frac{1}{2}K$-vs-$\frac{1}{2}K$), in contrast to W-LTLS, where as explained in Section 2.2, the induced subproblems are $\frac{1}{2}K$-vs-rest.

We already stated that the more balanced subproblems get, the more potentially difficult they become (as supported by our experiments in Section 2.3.3). This means that despite having a high minimum distance $\rho$, BCH codes would have an increased average binary error $\varepsilon$, which might deteriorate the training error bound (2.2).

BCH codes allow efficient hard decoding using their algebraic properties (e.g. using the Berlekamp-Massey algorithm [4, 24]), in a time complexity of $O(\ell)$ (which for our needs is $O(\log K)$). However, those efficient hard decoding algorithms are limited – they can fix only $\frac{\rho-1}{2}$ mistakes, i.e. find the closest codeword up to a distance of $\frac{\rho-1}{2}$ from the predicted binary word (arising from the binary learners’ responses). Another option is to run a complete nearest neighbor search (which is equivalent to the Hamming decoding). This however, as previously explained, has a higher time complexity in comparison to W-LTLS (i.e. $O(K^\alpha)$ for $\alpha \in (0.5, 1)$ [26], but not logarithmic in $K$ like W-LTLS), and often even implies higher space complexities.

In Section 2.3.5 we show experimental result comparing W-LTLS and BCH codes. Our results show that W-LTLS is superior to BCH codes decoded with efficient algorithms, that is, decoding algorithms which are logarithmic in $K$. Using the BCH codes with Hamming decoding (i.e. by finding nearest neighbors) worsens the time and space complexities but improves the accuracy, emphasizing the importance of the minimum Hamming distance. We also compare the BCH codes to the pruned (sparse) W-LTLS models, demonstrating that after pruning, W-LTLS
is superior to BCH codes with any decoding algorithm. Moreover, unlike other fast decoding schemes, ours allows efficient loss-based decoding using any loss function (e.g., as opposed to only the Hamming loss or any other specific loss function). This is an exciting new direction, which paves the way to more research on coding schemes designated for extreme classification usages.
2.3 Experiments

We tested our algorithms on 5 extreme multiclass datasets previously used in [14], having approximately $10^2$, $10^3$, and $10^4$ classes (see Table 2.1). We used AROW [10] to train the binary functions $\{f_j\}$ of W-LTLS. Its online updates are based on the squared hinge loss $L_{SH}(z) \equiv (\max(0, 1 - z))^2$. For each dataset, we built wide-graphs with multiple slice widths (depending on the number of classes of the specific dataset). For each configuration (dataset and graph) we performed five runs using random sample shuffling in every epoch, and a random path assignment (as explained in Section 1.4.1, unlike the greedy policy used in [19]), and report averages over these five runs.

Unlike [19], we train the $\ell$ binary learners independently rather than in a joint (structured) manner. This allows parallel independent training, as common for training binary learners for ECOC, with no need to perform full multiclass inference during training.

2.3.1 Loss-based decoding

We run the W-LTLS model with different loss functions for loss-based decoding: the exponential loss, the squared loss (used by LTLS, see Lemma 2.2), the log loss, the hinge loss, and the squared hinge loss.

The results appear in Figure 2.2. We observe that decoding with the exponential loss works the best on all five datasets. For the two largest datasets (Dmoz and LSHTC1) we report significant accuracy improvement when using the exponential loss for decoding in graphs with large slice widths ($b$), over the squared loss used implicitly by LTLS. Indeed, for these larger values of $b$, the subproblems are easier.
(see Section 2.3.3 for detailed analysis). This should result in larger prediction margins $|f_j(x)|$, as we indeed observe empirically (shown in Section 2.3.1). The various loss functions $L(z)$ differ significantly for $z < 0$, potentially explaining the larger accuracy differences as $b$ increases when decoding with different loss functions.

Figure 2.2: First row (per dataset): Multiclass test accuracy as a function of the model size (MBytes) for loss-based decoding with different loss functions. Second row: Relative increase in multiclass test accuracy compared to decoding with the squared loss used implicitly in LTLS. The secondary x-axes (top axes, blue) indicate the slice widths ($b$) used for the W-LTLS trellis graphs.
Average predictions margin

As discussed in Section 2.3.1, the following Figure 2.3 shows that for larger values of $b$, the predictions’ average margin, i.e. $\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{l} |f_j(x_i)|$, increases.

![Graph of Average absolute margin vs Slice width](image)

Figure 2.3: The average absolute margin vs the slice width $b$. 
2.3.2 Multiclass test accuracy

We compare the multiclass test accuracy of W-LTLS (using the exponential loss for decoding) to the same baselines presented in [19]. Namely we compare to LTLS [19], LOMTree [8] (results quoted from [19]), FastXML [27] (run with the default parameters on the same computer as our model), and OVR (binary learners trained using AROW). For convenience, the results are also presented in a tabular form in Appendix B.1.

![Graphs showing multiclass test accuracy vs model size and prediction time for different datasets.](image)

Figure 2.4: First row (per dataset): Multiclass test accuracy vs model size. Second row: Multiclass test accuracy vs prediction time. A 95% confidence interval is shown for the results of W-LTLS.
Accuracy vs Model size

The first row of Figure 2.4 (best seen in color) summarizes the multiclass accuracies vs model size.

Among the four competitors, LTLS enjoys the smallest model size, LOMTree and FastXML have larger model sizes, and OVR is the largest. LTLS achieves lower accuracies than LOMTree on two datasets, and higher ones on the other two. OVR enjoys the best accuracy, yet with a price of model size. For example, in Dmoz, LTLS achieves 23% accuracy vs 35.5% of OVR, though the model size of the latter is ×200 larger than of the former.

In all five datasets, an increase in the slice width of W-LTLS (and consequently in the model size) translates almost always to an increase in accuracy. Our model is often better or competitive with the other algorithms that have logarithmic inference time complexity (LTLS, LOMTree, FastXML), and also competitive with OVR in terms of accuracy, while we still enjoy much smaller model sizes.

For the smallest model sizes of W-LTLS (corresponding to \(b = 2\)), our trellis graph falls back to the one of LTLS. The accuracies gaps between these two models may be explained by the different binary learners the experiments were run with – LTLS used averaged Perceptron as the binary learner whilst we used AROW. Also, LTLS was trained in a structured manner with a greedy path assignment policy while we trained every binary function independently with a random path assignment policy (see Section 2.2.1). In our runs we observed that independent training achieves accuracy competitive with to structured online training, while usually converging much faster. It is interesting to note that for the ImageNet dataset the LTLS model cannot fit the data, i.e the training error is close to 1 and the test accuracy is close to 0. The reason is that the binary subproblems are very hard, as was also noted by [19]. By increasing the slice width (\(b\)), the W-LTLS model mitigates this underfitting problem, still with logarithmic time and space complexity.

We also observe in the first row of Figure 2.4 that there is a point where the multiclass test accuracy of W-LTLS starts to saturate (except for ImageNet). Our experiments show that this point can be found by looking at the training error and its bound only. We thus have an effective way to choose the optimal model size for the dataset and space/time budget at hand by performing model selection (width of the graph in our case) using the training error bound only (see detailed analysis in Section 2.3.3).
Accuracy vs Prediction time

In the second row of Figure 2.4 we compare prediction (inference) time of W-LTLS to other methods. LTLS enjoys the fastest prediction time, but suffers from low accuracy. LOMTree runs slower than LTLS, but sometimes achieves better accuracy. Despite being implemented in Python, W-LTLS is competitive with FastXML, which is implemented in C++.
2.3.3 Multiclass training error

Average binary training loss

To better understand the above results, we validate our hypothesis that wider graphs lead to easier binary subproblems. In the first row (per dataset) of Figure 2.5 we plot the average binary training loss $\varepsilon$ as a function of model size. The average is both over the induced binary subproblems and over the five runs.

![Figure 2.5: First row (per dataset): Average binary loss ($\varepsilon$) vs Model size. Second row: Multiclass training error and multiclass training error bound (on a logarithmic scale) vs Model size. The secondary x-axes (top axes, blue) indicate the slice widths (b) used for the W-LTLS trellis graphs.](image)
In all datasets we observe a decrease of the average error as the slice width $b$ grows. The decrease is sharp for low values of $b$ and then practically almost converges (to zero). These plots validate our claim – as the subproblems become more unbalanced they also become easier.

**Multiclass training error**

In the second row (per dataset) of Figure 2.5 we plot the multiclass training error (when using loss-based decoding defined in (1.1) with the squared hinge loss) and its bound (2.2) for different model sizes $3$ (MBytes).

For the bound, we set the minimum distance $\rho = 4$ as explained in Section 2.2 and $L_{SH}(0) = 1$. The average binary loss $\varepsilon$ was computed as in (2.3).

For all datasets, the multiclass training error follows qualitatively its bound. For the two larger datasets, shown in the two bottom panels, both the error and its bound decrease to some point, and then start to increase. This can be explained as follows: at some point, the increase in the slice widths (and $\ell$ and the model size), stops to significantly decrease $\varepsilon$ (see first row of Figure 2.5), such that the term $\ell \times \varepsilon$ appears in the training error bound (2.2) overall starts increasing (recall that the denominator $\rho \times L(0)$ is constant). By comparing these plots to the multiclass test accuracy plots in Figure 2.4 we observe that at the same point where the training error and its bound start to increase, the test accuracy does not increase significantly anymore. For example, for LSHTC1 and Dmoz datasets, the training error bounds start to increase at around model size of $2^{12}$, and at the same time the test accuracy stops increasing significantly. This suggests that model size of $2^{12}$ is a good point in terms of accuracy/model size tradeoff.

---

$3$ The model size is linear in the number of predictors, which in turn depends on the slice width like $\frac{\ell^2}{\log b}$.
2.3.4 Exploiting the sparsity of the datasets

We now demonstrate that the post-pruning proposed in Section 2.2.2, which zeroes the weights in \([-\lambda, \lambda]\), is highly beneficial. Since ImageNet is not sparse at all, we do not consider it in this section.

We tune the threshold \(\lambda\) so that the degradation in the multiclass validation accuracy is at most 1\% (tuning the threshold is done after the cumbersome learning of the weights, and does not require much time).

In Figure 2.6 we plot the multiclass test accuracy versus model size for the non-sparse W-LTLS, as well as the sparse W-LTLS after pruning the weights as explained above. We compare ourselves to the aforementioned sparse competitors: DiSMEC, PD-Sparse, and PPDSparse (all results quoted from [35]). Since the aforementioned FastXML [27] also exploits sparsity to reduce the size of the learned trees, we consider it here as well (we run the code supplied by the authors for various numbers of trees). For convenience, all the results are also presented in a tabular form in Appendix B.2.

![Figure 2.6](image)

Figure 2.6: Multiclass test accuracy vs model size for sparse models. Lines between two W-LTLS plots connect the same models before and after the pruning. The secondary x-axes (top axes, blue) indicate the slice widths (\(b\)) used for the (unpruned) W-LTLS trellis graphs.
We observe that our method can induce very sparse binary learners with a small degradation in accuracy. In addition, as expected, the wider the graphs (larger $b$), the more beneficial is the pruning. Interestingly, while the parameter number increases, the actual storage space for the pruned sparse models may even decrease, as observed for the sector and aloi.bin datasets.

Finally, we note that although PD-Sparse [14] and DiSMEC [2] perform better on some model size regions of the datasets, their worse case space requirement during training is linear in the number of classes $K$, whereas our approach guarantees (adjustable) logarithmic space for training.

**How much can we prune, and why?**

In order to shed light on the reasons why larger slice widths allow pruning more weights without accuracy degradation, we focus on the resulting binary classifiers at the end of training, before and after pruning. In the following Figure 2.7 we show that wider graphs induce models which use less features, even before pruning. Moreover, the classifiers induced by wider graphs even achieve non-zero percentage of around 1% after pruning, without a significant accuracy degradation.

![Figure 2.7: Percentage of non-zero weights at end of training, before and after pruning, vs the slice width $b$.](image)

We already seen in Section 2.3.3 that the average binary loss $\epsilon$ decreases as the graph grow wider, implying that the binary subproblems become easier and more suitable for linear separation. Intuitively, easier subproblems can be tackled using less features (especially when the dataset is very sparse). Practically, they lead to less prediction mistakes during training, which in turn mean less updates to the model when learning with AROW like we do. Since the samples are extremely sparse, performing fewer updates means using fewer features per classifier and results in less and less non-zero weights at the end of training.

Low magnitude weights are assumed to be less important for classification, and pruning them effectively performs a type of regularization on the model.
2.3.5 Comparison to BCH codes

For the sake of completeness, we now turn to empirically compare the accuracy of our model to the accuracy of BCH codes (see Section 2.2.3).

In the first row (per dataset) of Figure 2.8 we evaluate and compare the performance of W-LTLS (blue) and sparse W-LTLS (orange) to BCH codes using two decoding schemes – the fast Berlekamp-Massey algorithm (green), which runs in linear time in the code length (ℓ) but can only fix \( \left\lfloor \frac{\rho - 1}{2} \right\rfloor \) binary prediction errors, and the "full" Hamming decoding, i.e. using algorithms for finding the nearest neighbor (red).

While the time complexity of both our method and the Berlekamp-Massey decoding algorithm is linear in the number of predictors ℓ, our method significantly outperforms BCH codes using this decoding algorithm. Using the Hamming decoding, BCH codes outperform W-LTLS without pruning. However, as previously discussed in Section 2.2.3 and Section 1.3, nearest neighbor algorithms cannot guarantee a worst-case time complexity logarithmic in K like our method does, and often hold space complexity implications or special hardware requirements.

It is also interesting to note that since BCH codes induce perfectly balanced subproblems (\( \frac{1}{2} - \text{vs} - \frac{1}{2} \)), these subproblems are harder to learn. This is shown in the second row (per dataset) of Figure 2.8. The overall higher accuracy of BCH codes (with the less efficient Hamming decoding) suggests that their high minimum distance \( \rho \) compensates for their inferior binary loss.

The comparison to the pruned (sparse) W-LTLS models tells a different story. Due to the reasons previously discussed in Section 2.3.4, the imbalance binary subproblems of W-LTLS are especially suitable for pruning and inducing sparse models. Since the high minimum distance \( \rho \) of BCH codes comes with the price of inducing perfectly balanced subproblems, simple pruning schemes are not applicable on these codes. This significant drawback of BCH codes makes them inferior to W-LTLS, even when performing Hamming decoding. Performing loss-based decoding instead of hard decoding will surely improve the performance of BCH codes, but comes with a great cost to prediction times (since even fast nearest neighbor algorithms are not applicable to it).
Figure 2.8: Comparison of W-LTLS to BCH codes. First row (per dataset): Accuracy comparison. Second row: Average binary loss ($\rho$) comparison. The secondary x-axes (top axes, blue) indicate the number of predictors ($\ell$) of the corresponding coding matrices. imageNet is not a sparse dataset, hence we do apply the pruning scheme on it.
Chapter 3

Discussion and Future Work

We propose a new efficient loss-based decoding algorithm, which we call W-LTLS, that works for any loss function. Motivated by a general error bound for loss-based decoding [1], we show how to build on the log-time log-space (LTLS) framework [19] and employ a more general type of trellis graph architectures. We show how to design error correcting output codes that induce easier learning problems and sparse models. Our method offers a tradeoff between multiclass accuracy, model size and prediction time, and achieves better multiclass accuracies under logarithmic time and space guarantees.

We turn to discuss some of the main ideas arising from our work, and offer directions for future work.

3.1 The average binary loss $\epsilon$ can also be controlled

Recall that the training error bound of ECOC schemes, presented by Allwein et al. [1], is given by $\frac{\ell \times \epsilon}{\rho \times L(0)}$, as explained in Section 2.2. The width of the code $\ell$ and the minimum Hamming distance $\rho$ are known to be important for ECOC schemes in coding theory and communication. However, the average binary loss $\epsilon$ is mostly unique to machine learning usages, and might therefore be overlooked.

By employing wider graphs (and codes), W-LTLS controls the learnability of the binary subproblems and induce easier ones. We show experimentally that this tradeoff between $\ell$ and $\epsilon$ can be beneficial even when the minimum Hamming distance $\rho$ remains constant. This stands in contrast to other codes like BCH, that induce better $\frac{\ell}{\rho}$ ratios but have higher losses $\epsilon$ due to their balanced subproblems.
3.2 Error correcting codes can also induce sparse models

Many of the recent work on extreme classification [2][14][35] focused on producing sparse models in order to reduce the restrictively high model sizes used for extreme classification.

We explain in Section 2.2.2 how imbalanced subproblems can exploit the sparsity of extreme classification datasets to create sparse ECOC models. We propose a straightforward post-pruning scheme that yields highly sparse model without a significant accuracy loss. Potentially, a regularization term can be presented to our learning scheme, already during training, to allow a more efficient post-pruning, like was done in DiSMEC [2].

3.3 Improving the minimum Hamming distance $\rho$

One could try to improve the restrictively low minimum Hamming distance $\rho$ of W-LTLS discussed in Section 1.4.2. We show in Section 2.3.5 that this is a promising direction – codes with a better minimum distance lead to higher accuracy, even when performing hard decoding. Finding a way to employ graphs that achieve higher minimum distance whilst still enjoying fast loss-based decoding (logarithmic in $K$ even in the worst case) would be a great contribution. Notice however that this should be done without deteriorating the average binary loss $\varepsilon$ too much, for the reasons discussed throughout the entire work.
3.4 Exploiting the graph’s structure to ease the binary subproblems

We notice that the bijective mapping between classes and paths forces parameter sharing across different classes. For instance, if two classes are mapped to two paths that differ by only four edges, all the binary learners corresponding to the agreement edges (i.e., all edges except of those four) should partition the feature space such that samples from these two classes will be classified similarly. If these classes are very dissimilar, the described partitions would most likely be hard to train (the same goes for very similar classes mapped to two completely disjoint paths).

This suggests it is beneficial for $\varepsilon$ to map similar classes to similar paths (which will require some sort of a similarity measure between classes). Similar ideas have been employed by Bengio et. al. in [3] on trees for hierarchical extreme classification, and by Cisse et. al. [9] on ECOC for extreme classification.

Recall the greedy policy for assigning paths to classes (i.e., creating bijective the mapping) proposed in [19] and described in Section 1.4.1. They proposed to employ a policy suitable for online learning, where during training, a sample whose class is yet unassigned with a path, is assigned to the heaviest unassigned path. This suggests that a class is mapped to a path whose edges already respond positively to a sample from this class. Considering that positive responses from those edges suggest that they were trained by similar samples from other classes, this policy can be seen as an attempt to map similar paths to similar classes.

However, we believe this greedy policy is faulty for several reasons. First, due to its online nature, a class is assigned with a path based on a single sample from this class, which might be arbitrarily outlying. Moreover, the first few seen classes are assigned based on very few samples, meaning that they might be assigned arbitrarily close or far from one another, regardless to the actual similarity between them. A complementary phenomenon occurs for the last assigned classes – when very few paths are yet unassigned, a class might be assigned with an improper path. Finally, this policy showed little to none advantage over the random mapping for most datasets in several experiments we held.

To conclude, this is an interesting direction which holds potential benefits for our method.
3.5 Edges weighting scheme

Escalera et. al. [15] proposed the loss-weighted decoding approach, where a weighting matrix $M^W$ is calculated to allow adjusting the importance given to each binary learner per class – i.e. an input $x$ is predicted based on

$$k^* = \arg\min_k \sum_{j=1}^{\ell} M^W_{k,j} L(M_{k,j} f_j(x)).$$

This approach generalizes the standard loss-based decoding scheme (i.e. the one implemented in W-LTLS), where one simply sets $M^W_{k,j} = 1$, $\forall k, j$. They showed that this approach considerably increases the performance of ECOC.

It may be interesting to consider similar weighting schemes in the context of trellis graphs and W-LTLS. The two methods cannot be trivially combined such that the decoding would still be done efficiently by exploiting the graph structure. However, adjusting the weights of every edge (i.e. bit in the induced coding matrix) uniformly for all classes, can be done by simply scaling the weights assigned to each edge given the input, before finding the heaviest path. Whether or not a more complex weighting scheme exists such that the decoding time complexity is still linear in the number of edges, remains an open question and is deferred to future work.

3.6 Other types of graphs and graph algorithms

Finally, we believe that many ideas in this paper can be extended for other types of graphs and graph algorithms, e.g. shortest paths on other types of DAGs, or minimum spanning tree of undirected weighted graph.

Such algorithms could potentially help tackle some of the aforementioned issues and directions. Moreover, it is possible that such algorithms will allow extending to ternary codes as proposed in [1], i.e. sparse code matrices with easier subproblems.
Appendix A

Extension to arbitrary K

A.1 Graph construction

The following algorithm construct graphs with any number of paths, and is not restricted to powers of 2. See Figure[A.1] for examples.

Algorithm 1: Graph construction with an arbitrary K

<table>
<thead>
<tr>
<th>Input :</th>
<th>Number of labels K and slice width b.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Convert K to a base-b representation.</td>
</tr>
<tr>
<td>2</td>
<td>Store the reverse representation in an array A of size $\lceil \log_b K \rceil + 1$ (i.e. the least significant b-ary digit is in A[0]).</td>
</tr>
<tr>
<td>3</td>
<td>Build a trellis graph with $\lceil \log_b K \rceil + 1$ inner slices, each with b vertices.</td>
</tr>
<tr>
<td>4</td>
<td>Add a source vertex s and connect it to the vertices of the first inner slice.</td>
</tr>
<tr>
<td>5</td>
<td>Add a sink vertex t.</td>
</tr>
<tr>
<td>6</td>
<td>For every inner slice $i = 0 \ldots \lfloor \log_b K \rfloor$, connect $A[i]$ vertices of the slice to the sink.</td>
</tr>
<tr>
<td>7</td>
<td>Delete any vertices from which the sink is unreachable.</td>
</tr>
</tbody>
</table>

Below we show that this construction indeed produces a graph with exactly K paths from source to sink. We start with a technical lemma.

Lemma A.1. Let $v \in V \setminus \{t\}$ be a vertex in the $i$th inner slice (where $i \in \{0 \ldots \lceil \log_b K \rceil \}$), and let $\mathcal{N}(v)$ be the number of paths from the source to $v$. Then $\mathcal{N}(v) = b^i$.

Proof We show this by induction on the slice index $i = 0 \ldots \lceil \log_b K \rceil$. In the
Figure A.1: An illustration of the graph construction algorithm for two different values of $K$, using $b = 3$. The darker edges are the ones created on stage 6 of Algorithm 1. Note that in the upper construction, the minimum distance between any two path is actually $\rho = 3$ and not $\rho = 4$ like previously noted. This sometimes holds minor accuracy implications.

Theorem A.2. The number of paths from source to sink is $K$, i.e. $\mathcal{N}(t) = K$.

Proof Let $A \in \{0 \ldots b-1\}^{\lfloor \log_b K \rfloor +1}$ be the array of the reverse base-$b$ representation of $K$. Using the decomposition of the $b$-ary representation, i.e. $K = \sum_{i=0}^{\lfloor \log_b K \rfloor} A[i] \cdot b^i$, and Lemma A.1, we get:

$$\begin{align*}
\mathcal{N}(t) &= \sum_{v \in \text{in}(t)} \mathcal{N}(v) \\
&= \sum_{i=0}^{\lfloor \log_b K \rfloor} \left[ \sum_{v \in \text{slice} \cap \text{in}(t)} \mathcal{N}(v) \right]_{b^i} \\
&= \sum_{i=0}^{\lfloor \log_b K \rfloor} \left[ \text{slice} \cap \text{in}(t) \right]_{b^i} \\
&= \sum_{i=0}^{\lfloor \log_b K \rfloor} A[i] \cdot b^i = K.
\end{align*}$$
A.2 Loss-based decoding generalization

We now show how to adjust the generalization of loss-based decoding for graphs with an arbitrary number of paths $K$, constructed using Algorithm 1.

The idea of the reduction in (2.1) is that going through an edge $e_j$ incurs the loss of turning its corresponding bit on (i.e. $L (1 \times f_j (x))$), but also the loss of turning off the bits corresponding to other edges between its slices (i.e. $\sum_{j' \in S(e_j) \setminus \{e_j\}} L \left( (-1) \times f_{j'} (x) \right)$), which cannot coappear with $e_j$ in the same path (i.e. a codeword).

The only change in the general case is that an edge $e_j = (u_j, v_j)$ that is connected to the sink $t$ cannot coappear with any other edge outgoing from a vertex in the same vertical slice as $u_j$, or that is reachable from $u_j$.

Let $\delta (v)$ be the shortest distance from the source to $v$ (in terms of number of edges).

We define an updated $S$ function (which is a generalization of the one defined in Section 2.1), where for every edge $e_j = (u_j, v_j) \in E$ we set:

$$S (e_j) = \begin{cases} \{ (u, u') : \delta (u) = \delta (u_j) \} & v_j \neq t \\ \{ (u, u') : \delta (u) \geq \delta (u_j) \} & v_j = t \end{cases} \quad (A.1)$$

and the weights are set as in (2.1) but with the new $S$ function defined in (A.1),

$$w_j (x) = L (1 \times f_j (x)) + \sum_{j' \in S(e_j) \setminus \{e_j\}} L \left( (-1) \times f_{j'} (x) \right) . \quad (A.2)$$
Figure A.2: An illustration of a graph with $K = 9$ and $b = 2$.

For example, in Figure A.2 we have,

- $S(e_6) = \{e_6, e_7, e_8, e_9\}$
- $S(e_{11}) = \{e_{10}, e_{11}\}$
- $S(e_{13}) = \{e_2, \ldots, e_{13}\}$
- $S(e_2) = \{e_2, e_3, e_4, e_5, e_{13}\}$
- $S(e_{12}) = \{e_{12}\}$
- $S(e_{13}) = \{e_2, \ldots, e_{13}\}$

Below we show that Theorem 2.1 is correct for any $K$. Let $P$ be a path on the trellis graph from the source $s$ to the sink $t$. We start with the following lemma.

**Lemma A.3.** Let $e_q = (u_q, t)$ be the last edge in $P$. For every edge $e_j = (u_j, v_j) \in P \setminus \{e_q\}$ we have $\delta(v_j) = \delta(u_j) + 1$.

**Proof** Following immediately from the graph construction – other than edges to the sink, there are only edges between adjacent slices (without cycles). Therefore, any vertex $v$ in a vertical slice $i$ (where the leftmost slice containing $s$ is the first one, i.e. $i = 0$) holds $\delta(v) = i$, and every edge $e_j = (u_j, v_j) \in P \setminus \{e_q\}$ holds $\delta(v_j) = \delta(u_j) + 1$.

The next corollaries follow immediately.

**Corollary A.4.** For every two different vertices $u, v \in V \setminus \{t\}$ along $P$ we have $\delta(u) \neq \delta(v)$.

**Corollary A.5.** Let $e_q = (u_q, t)$ be the last edge in $P$. For every edge $e_j = (u_j, v_j) \in P \setminus \{e_q\}$ we have $\delta(u_j) < \delta(u_q)$.

Clearly, since the graph is directed and acyclic, we get the next lemma.

**Lemma A.6.** Every vertex in $V$ can have at most one incoming edge and one outgoing edge in $P$.
By using Lemma A.6, Corollary A.4 and Lemma A.3 we have the next corollary.

**Corollary A.7.** Set an edge $e_j = (u_j, v_j)$ along the path $P$. Then, $\forall e' \in S(e_j) \setminus \{e_j\}$ we have $e' \notin P$.

**Proof** Let $e' = (u_j', v_j')$ be any edge in $S(e_j) \setminus \{e_j\}$, and assume $e' \in P$. We consider two cases:

1. $v_j \neq t$:
   By the definition of $S(e_j)$, we get $\delta(u_j') = \delta(u_j)$. By negating Corollary A.4 we get $u_j' = u_j$. Therefore $e_j, e'$ are two edges in $P$ outgoing from the same vertex $u_j$, in contradiction to Lemma A.6.

2. $v_j = t$:
   By definition, $e_j \neq e'$. By Lemma A.6 we get that $u_j \neq u_j'$. Also, since $t$ is a sink it has no outgoing edges, thus $u_j' \neq t$.
   By the definition of $S(e_j)$, we get $\delta(u_j') \geq \delta(u_j)$. Since $u_j' \neq u_j$, we can use Corollary A.4 to rule out equivalence and get $\delta(u_j') > \delta(u_j)$.
   Following Lemma A.3, $u_j'$ must appear later than $u_j$ in $P$. However, $t$ is the only vertex in $P$ to appear after $u_j$ ($t$ is a sink), and since $u_j' \neq t$, we get a contradiction.

$\blacksquare$
We conclude with the main result of this section, which states that Theorem 2.1 is correct for any $K$.

**Theorem A.8.** Following the notations of Theorem 2.1, assume the weights of the edges are calculated as in Eq. (A.2) with the $S$ function defined in (A.1). Then, the weight of any path $P_k$ corresponding to class $k$ equals to the loss suffered by predicting class $k$, i.e. $w(P_k) = \sum_{j=1}^{f} L(M_{k,j}f_j(x))$.

**Proof** For any class $k$, we denote the last edge in $P_k$ by $e_q = (u_q, t)$. We have,

$$w(P_k) = \sum_{j \in P_k} w_j(x)$$

$$= \sum_{j \in P_k} \left[ L\left( \frac{1}{M_{k,j}} \times f_j(x) \right) + \sum_{j' \neq j' \in S(e_j) \backslash \{e_j\}} \right]$$

$$= \sum_{j \in P_k} \left[ \sum_{j' \in S(e_j)} L\left( M_{k,j} \times f_j(x) \right) \right]$$

$$= \sum_{j \in P_k \backslash \{e_q\}} \left[ \sum_{j' \in S(e_j)} L\left( M_{k,j} \times f_j(x) \right) \right]$$

$$= \sum_{j \in P_k \backslash \{e_q\}} \left[ \sum_{j' \in S(e_j)} L\left( M_{k,j} \times f_j(x) \right) \right]$$

$$= \sum_{j \in P_k \backslash \{e_q\}} \left[ \sum_{j' \in S(e_j) \backslash \{e_j\}} L\left( M_{k,j} \times f_j(x) \right) \right]$$

$$= \sum_{j \in E} L\left( M_{k,j} \times f_j(x) \right) = \sum_{j=1}^{f} L\left( M_{k,j}f_j(x) \right).$$

$\blacksquare$
Appendix B

Experiments appendix

B.1 Experimental results of the multiclass test accuracy experiments

In the following Table B.1 are the results of Section 2.3.2 organized in a tabular form – the model sizes and prediction times of the tested algorithms and their test accuracy. The results of W-LTLS are averaged on five runs.
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Algorithm</th>
<th>Model size (Bytes)</th>
<th>Prediction time (sec)</th>
<th>Test accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECTOR</td>
<td>LTLS</td>
<td>5.9 MB</td>
<td>0.14</td>
<td>88.45</td>
</tr>
<tr>
<td></td>
<td>W-LTLS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = 2</td>
<td>5.9 MB</td>
<td>0.14</td>
<td>91.63</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = 4</td>
<td>9.7 MB</td>
<td>0.16</td>
<td>93.92</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = 5</td>
<td>11.6 MB</td>
<td>0.16</td>
<td>94.32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = 7</td>
<td>15.4 MB</td>
<td>0.18</td>
<td>94.86</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = 10</td>
<td>26.5 MB</td>
<td>0.23</td>
<td>94.88</td>
</tr>
<tr>
<td></td>
<td>LOMTree</td>
<td>17.0 MB</td>
<td>0.16</td>
<td>82.10</td>
</tr>
<tr>
<td></td>
<td>FASTXML</td>
<td>8.9 MB</td>
<td>0.32</td>
<td>86.26</td>
</tr>
<tr>
<td></td>
<td>OVR</td>
<td>41.3 MB</td>
<td>-</td>
<td>94.79</td>
</tr>
<tr>
<td>ALON.BIN</td>
<td>LTLS</td>
<td>102.1 MB</td>
<td>1.0</td>
<td>82.24</td>
</tr>
<tr>
<td></td>
<td>W-LTLS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = 2</td>
<td>102.1 MB</td>
<td>1.4</td>
<td>85.03</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = 3</td>
<td>133.6 MB</td>
<td>1.5</td>
<td>89.39</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = 5</td>
<td>216.2 MB</td>
<td>1.7</td>
<td>92.49</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = 7</td>
<td>328.0 MB</td>
<td>2.1</td>
<td>93.70</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = 10</td>
<td>537.0 MB</td>
<td>2.7</td>
<td>94.92</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = 15</td>
<td>777.5 MB</td>
<td>3.5</td>
<td>94.93</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = 20</td>
<td>1.1 GB</td>
<td>5.0</td>
<td>95.16</td>
</tr>
<tr>
<td></td>
<td>LOMTree</td>
<td>106.0 MB</td>
<td>1.6</td>
<td>89.47</td>
</tr>
<tr>
<td></td>
<td>FASTXML</td>
<td>522.0 MB</td>
<td>5.3</td>
<td>95.38</td>
</tr>
<tr>
<td></td>
<td>OVR</td>
<td>2.4 GB</td>
<td>-</td>
<td>95.90</td>
</tr>
<tr>
<td>IMAGE.NET</td>
<td>LTLS</td>
<td>0.16 MB</td>
<td>15.0</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>W-LTLS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = 2</td>
<td>0.16 MB</td>
<td>21.4</td>
<td>0.42</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = 5</td>
<td>0.34 MB</td>
<td>26.3</td>
<td>1.59</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = 10</td>
<td>0.84 MB</td>
<td>40.3</td>
<td>4.36</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = 15</td>
<td>1.2 MB</td>
<td>52.6</td>
<td>4.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = 20</td>
<td>1.8 MB</td>
<td>76.2</td>
<td>7.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = 30</td>
<td>3.7 MB</td>
<td>194.2</td>
<td>12.60</td>
</tr>
<tr>
<td></td>
<td>LOMTree</td>
<td>35.0 MB</td>
<td>37.7</td>
<td>5.37</td>
</tr>
<tr>
<td></td>
<td>FASTXML</td>
<td>1.6 GB</td>
<td>172.3</td>
<td>6.84</td>
</tr>
<tr>
<td></td>
<td>OVR</td>
<td>3.8 MB</td>
<td>-</td>
<td>15.60</td>
</tr>
<tr>
<td>Dataset</td>
<td>Algorithm</td>
<td>Model Size (Bytes)</td>
<td>Prediction Time (sec)</td>
<td>Test Accuracy (%)</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>--------------------</td>
<td>-----------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>DMOZ</td>
<td>LTLS</td>
<td>193.9 MB</td>
<td>5.2</td>
<td>23.04</td>
</tr>
<tr>
<td></td>
<td>b = 2</td>
<td>193.9 MB</td>
<td>7.5</td>
<td>25.76</td>
</tr>
<tr>
<td></td>
<td>b = 3</td>
<td>248.0 MB</td>
<td>8.1</td>
<td>29.56</td>
</tr>
<tr>
<td></td>
<td>b = 5</td>
<td>429.2 MB</td>
<td>9.8</td>
<td>33.92</td>
</tr>
<tr>
<td></td>
<td>b = 10</td>
<td>1.1 GB</td>
<td>16.6</td>
<td>37.44</td>
</tr>
<tr>
<td></td>
<td>b = 20</td>
<td>2.8 GB</td>
<td>41.4</td>
<td>38.43</td>
</tr>
<tr>
<td></td>
<td>b = 30</td>
<td>4.3 GB</td>
<td>73.6</td>
<td>38.89</td>
</tr>
<tr>
<td></td>
<td>b = 40</td>
<td>6.3 GB</td>
<td>164.5</td>
<td>38.81</td>
</tr>
<tr>
<td></td>
<td>b = 50</td>
<td>9.0 GB</td>
<td>332.3</td>
<td>38.89</td>
</tr>
<tr>
<td></td>
<td>LOMTree</td>
<td>1.8 GB</td>
<td>28.0</td>
<td>21.27</td>
</tr>
<tr>
<td></td>
<td>FASTXML</td>
<td>1.2 GB</td>
<td>60.7</td>
<td>38.58</td>
</tr>
<tr>
<td></td>
<td>OVR</td>
<td>38.0 GB</td>
<td>-</td>
<td>35.50</td>
</tr>
<tr>
<td>LSHTC1</td>
<td>LTLS</td>
<td>256.3 MB</td>
<td>0.65</td>
<td>9.50</td>
</tr>
<tr>
<td></td>
<td>b = 2</td>
<td>256.3 MB</td>
<td>1.1</td>
<td>10.56</td>
</tr>
<tr>
<td></td>
<td>b = 3</td>
<td>361.6 MB</td>
<td>1.1</td>
<td>13.72</td>
</tr>
<tr>
<td></td>
<td>b = 5</td>
<td>631.6 MB</td>
<td>1.3</td>
<td>16.58</td>
</tr>
<tr>
<td></td>
<td>b = 10</td>
<td>1.5 GB</td>
<td>2.2</td>
<td>20.44</td>
</tr>
<tr>
<td></td>
<td>b = 20</td>
<td>4.0 GB</td>
<td>5.1</td>
<td>21.76</td>
</tr>
<tr>
<td></td>
<td>b = 30</td>
<td>6.3 GB</td>
<td>9.9</td>
<td>22.08</td>
</tr>
<tr>
<td></td>
<td>b = 40</td>
<td>9.0 GB</td>
<td>20.4</td>
<td>22.30</td>
</tr>
<tr>
<td></td>
<td>b = 45</td>
<td>10.8 GB</td>
<td>29.3</td>
<td>22.47</td>
</tr>
<tr>
<td></td>
<td>LOMTree</td>
<td>744.0 MB</td>
<td>6.8</td>
<td>10.56</td>
</tr>
<tr>
<td></td>
<td>FASTXML</td>
<td>366.6 MB</td>
<td>10.7</td>
<td>21.66</td>
</tr>
<tr>
<td></td>
<td>OVR</td>
<td>56.3 GB</td>
<td>-</td>
<td>21.90</td>
</tr>
</tbody>
</table>

Table B.1: Tabular results of the multiclass test accuracy experiments in Section 2.3.2
## B.2 Experimental results of the sparsity experiments

In the following Table B.2 are the results of Section 2.3.4 organized in a tabular form.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Algorithm</th>
<th>( b = 2 )</th>
<th>( \lambda = 0.324 )</th>
<th>Model size (Bytes)</th>
<th>Test accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECTOR</td>
<td>Sparse W-LTLS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( b = 2 )</td>
<td>( \lambda = 0.303 )</td>
<td>1.4 MB</td>
<td>90.66</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( b = 4 )</td>
<td>( \lambda = 0.327 )</td>
<td>1.1 MB</td>
<td>93.86</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( b = 5 )</td>
<td>( \lambda = 0.341 )</td>
<td>1.0 MB</td>
<td>94.11</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( b = 10 )</td>
<td>( \lambda = 0.358 )</td>
<td>0.74 MB</td>
<td>94.82</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FastXML</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T = 25 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T = 50 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T = 75 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T = 100 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T = 150 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T = 200 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ALOI.BIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sparse W-LTLS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( b = 2 )</td>
<td>( \lambda = 0.015 )</td>
<td>37.3 MB</td>
<td>84.09</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( b = 3 )</td>
<td>( \lambda = 0.015 )</td>
<td>32.7 MB</td>
<td>88.59</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( b = 5 )</td>
<td>( \lambda = 0.017 )</td>
<td>25.4 MB</td>
<td>91.74</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( b = 7 )</td>
<td>( \lambda = 0.016 )</td>
<td>25.9 MB</td>
<td>92.89</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( b = 10 )</td>
<td>( \lambda = 0.013 )</td>
<td>30.3 MB</td>
<td>94.48</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( b = 15 )</td>
<td>( \lambda = 0.014 )</td>
<td>31.2 MB</td>
<td>94.27</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( b = 20 )</td>
<td>( \lambda = 0.016 )</td>
<td>25.4 MB</td>
<td>94.55</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FastXML</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T = 5 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T = 10 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T = 25 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T = 50 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T = 100 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DiSMEC</td>
<td>10.7 MB</td>
<td>96.28</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PD-Sparse</td>
<td>12.7 MB</td>
<td>96.20</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PPDSparse</td>
<td>9.3 MB</td>
<td>96.38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dataset</td>
<td>Algorithm</td>
<td>Model Size (Bytes)</td>
<td>Test Accuracy (%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
<td>--------------------</td>
<td>-------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dmoz</strong></td>
<td><strong>Sparse W-LTLS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$b = 2$ $\lambda = 0.347$</td>
<td>43.2 MB</td>
<td>24.64</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$b = 3$ $\lambda = 0.325$</td>
<td>56.1 MB</td>
<td>28.63</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$b = 5$ $\lambda = 0.314$</td>
<td>69.3 MB</td>
<td>33.04</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$b = 10$ $\lambda = 0.323$</td>
<td>93.7 MB</td>
<td>36.75</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$b = 20$ $\lambda = 0.345$</td>
<td>126.3 MB</td>
<td>37.72</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$b = 30$ $\lambda = 0.385$</td>
<td>145.6 MB</td>
<td>38.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$b = 40$ $\lambda = 0.374$</td>
<td>193.1 MB</td>
<td>38.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$b = 50$ $\lambda = 0.323$</td>
<td>324.3 MB</td>
<td>38.16</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$T = 1$</td>
<td>24.5 MB</td>
<td>27.09</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$T = 2$</td>
<td>49.2 MB</td>
<td>31.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$T = 5$</td>
<td>123.0 MB</td>
<td>35.84</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$T = 10$</td>
<td>246.5 MB</td>
<td>37.47</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$T = 50$</td>
<td>1.2 GB</td>
<td>38.58</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$T = 300$</td>
<td>7.4 GB</td>
<td>38.63</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>FastXML</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$T = 2$</td>
<td>24.5 MB</td>
<td>27.09</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$T = 5$</td>
<td>7.4 GB</td>
<td>38.63</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>DisMEC</strong></td>
<td>259.3 MB</td>
<td>39.38</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>PD-Sparse</strong></td>
<td>453.3 MB</td>
<td>39.91</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>PPD-Sparse</strong></td>
<td>526.7 MB</td>
<td>39.32</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>LSHTC1</strong></td>
<td><strong>Sparse W-LTLS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$b = 2$ $\lambda = 0.288$</td>
<td>24.7 MB</td>
<td>9.48</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$b = 3$ $\lambda = 0.237$</td>
<td>27.6 MB</td>
<td>12.66</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$b = 5$ $\lambda = 0.271$</td>
<td>31.5 MB</td>
<td>15.60</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$b = 10$ $\lambda = 0.226$</td>
<td>46.3 MB</td>
<td>19.78</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$b = 20$ $\lambda = 0.207$</td>
<td>85.0 MB</td>
<td>21.13</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$b = 30$ $\lambda = 0.192$</td>
<td>130.5 MB</td>
<td>21.56</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$b = 40$ $\lambda = 0.183$</td>
<td>184.7 MB</td>
<td>21.81</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$b = 45$ $\lambda = 0.246$</td>
<td>152.1 MB</td>
<td>21.88</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$T = 1$</td>
<td>7.3 MB</td>
<td>12.50</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$T = 5$</td>
<td>36.6 MB</td>
<td>17.98</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$T = 10$</td>
<td>73.3 MB</td>
<td>19.36</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$T = 50$</td>
<td>366.6 MB</td>
<td>21.66</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$T = 150$</td>
<td>1.1 GB</td>
<td>21.94</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$T = 300$</td>
<td>2.2 GB</td>
<td>22.04</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$T = 1200$</td>
<td>8.8 GB</td>
<td>21.92</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>FastXML</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$T = 2$</td>
<td>24.7 MB</td>
<td>9.48</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>DisMEC</strong></td>
<td>94.7 MB</td>
<td>22.74</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>PD-Sparse</strong></td>
<td>58.7 MB</td>
<td>22.46</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>PPD-Sparse</strong></td>
<td>254.0 MB</td>
<td>22.70</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table B.2: Tabular results of the multiclass test accuracy experiments for sparse models in Section 2.3.4
Bibliography


[18] Qixuan Huang, Anshumali Shrivastava, and Yiqiu Wang. MACH: Embarrassingly parallel k-class classification in \( o(d \log K) \) memory and \( o(k \log K + d \log K) \) time, instead of \( o(kd) \), 2018.


now present a generalization of the approach mentioned above, to a wider range of graphs and flexible, which allows for more accurate calculations in a shorter time, while saving memory.

We also generalize the approach, and consequently, achieve theoretical results and improvements on the codes used by us, and even improve the accuracy.

We are able to handle algorithms for small and large classes of graphs and therefore reduce both time and memory costs.

And finally, we present results that clearly show the advantages of the previous methods and the benefits of using the approach described here, which meets the requirements of both time and memory.

This work.
In the field of this study, the divide and conquer approach is applied, characterizing the document as belonging to several categories, which is also done by different researchers. The method relies on dividing the domain into two or more categories and using the divide-and-conquer strategy. In other words, it divides the input by dividing the domain into two or more categories, then uses the divide-and-conquer strategy.

Moreover, the method relies on correcting errors in codes, as previously mentioned. Therefore, the method relies on dividing the domain into two or more categories, then uses the divide-and-conquer strategy.

In this context, the method relies on dividing the domain into two or more categories, then uses the divide-and-conquer strategy. In other words, it divides the input by dividing the domain into two or more categories, then uses the divide-and-conquer strategy.

In this context, the method relies on dividing the domain into two or more categories, then uses the divide-and-conquer strategy. In other words, it divides the input by dividing the domain into two or more categories, then uses the divide-and-conquer strategy.

In this context, the method relies on dividing the domain into two or more categories, then uses the divide-and-conquer strategy. In other words, it divides the input by dividing the domain into two or more categories, then uses the divide-and-conquer strategy.

In this context, the method relies on dividing the domain into two or more categories, then uses the divide-and-conquer strategy. In other words, it divides the input by dividing the domain into two or more categories, then uses the divide-and-conquer strategy.

In this context, the method relies on dividing the domain into two or more categories, then uses the divide-and-conquer strategy. In other words, it divides the input by dividing the domain into two or more categories, then uses the divide-and-conquer strategy.

In this context, the method relies on dividing the domain into two or more categories, then uses the divide-and-conquer strategy. In other words, it divides the input by dividing the domain into two or more categories, then uses the divide-and-conquer strategy.

In this context, the method relies on dividing the domain into two or more categories, then uses the divide-and-conquer strategy. In other words, it divides the input by dividing the domain into two or more categories, then uses the divide-and-conquer strategy.

In this context, the method relies on dividing the domain into two or more categories, then uses the divide-and-conquer strategy. In other words, it divides the input by dividing the domain into two or more categories, then uses the divide-and-conquer strategy.
המחק נעשת בהנחיית פרופסור קובי קרמר
בפקולטה למדעי המחשב

אני מודה לטכניק על התמיכה הכספית הנדרשת בהש◇למוהי.
 ASUS להצלת טכניקות - מכון טכנולוגי לישראל

שון אחרי

איתי עברון

חתםhqי

הושאמש

2018 נובמבר
านון עיסול של קוד שגיאה על גרפים
למשיים של סיון עזבוק

איתי עזרו