The Role of Codeword-to-Class Assignments in Error-Correcting Codes: An Empirical Study

Itay Evron[*] **Ophir Onn**[*] **Tamar Weiss Orzech** **Hai Azeroual** **Daniel Soudry**
Department of Electrical and Computer Engineering, Technion, Israel

Abstract

Error-correcting codes (ECC) are used to reduce multiclass classification tasks to multiple binary classification subproblems. In ECC, classes are represented by the rows of a binary matrix, corresponding to codewords in a codebook. Codebooks are commonly either predefined *or* problem-dependent. Given predefined codebooks, codeword-to-class assignments are traditionally overlooked, and codewords are implicitly assigned to classes arbitrarily. Our paper shows that these assignments play a major role in the performance of ECC. Specifically, we examine similarity-preserving assignments, where similar codewords are assigned to similar classes. Addressing a controversy in existing literature, our extensive experiments confirm that similarity-preserving assignments induce easier subproblems and are superior to other assignment policies in terms of their generalization performance. We find that similarity-preserving assignments make *predefined* codebooks become problem-dependent, without altering other favorable codebook properties. Finally, we show that our findings can improve predefined codebooks dedicated to extreme classification.

1 INTRODUCTION

Error-correcting codes (ECC) have been long used in machine learning as a reduction from multiclass classification tasks to binary classification tasks (Dietterich and Bakiri, 1994). This scheme encodes classes using rows of a binary matrix called a codebook. The codebook columns induce binary partitions of classes, or subproblems, to be learned using any binary classification algorithm.

Recently, error-correcting codes have been used as output embeddings of deep networks (Yang et al., 2015; Rodríguez et al., 2018; Kusupati et al., 2021), on top of features extracted by deep CNNs (Dorj et al., 2018), and as a means to combine ensembles of several networks (Zheng et al., 2018). Moreover, they were recently used for their robustness in adversarial learning (Verma and Swami, 2019; Gupta and Amin, 2021; Song et al., 2021) and for their redundancy in regression tasks (Shah et al., 2022) and heterogeneous domain adaptation (Zhou et al., 2019b).

In extreme multiclass classification, where the number of classes is extremely large, ECC can be particularly beneficial. Several works (Jasinska and Karampatziakis, 2016; Evron et al., 2018) employed ECC to shrink the output space, decreasing the number of learned predictors, as well as the prediction time, to *logarithmic* in the number of classes. In comparison, both one-hot encoding and hierarchical models train a linear number of predictors (even though the latter enjoy a logarithmic prediction time).

The first step in employing ECC consists of selecting a *good* codebook. Some codebook properties are universally important for error correction, e.g., the minimum hamming distance between rows. Other properties are only important in some regimes, e.g., the decoding complexity which is essential mainly in extreme classification.

Roughly, codebooks can be divided into two categories: *predefined codebooks* and *problem-dependent codebooks*. Predefined codebooks are independent of the problem at hand, but offer simplicity (e.g., random codebooks), favorable error-correction properties (e.g., Hadamard codebooks in Zhang et al., 2003 or optimized codebooks in Gupta and Amin, 2022), or regime-specific advantages like fast decoding algorithms (Evron et al., 2018). On the other hand, problem-dependent approaches attempt to induce binary subproblems that are tailored for a given dataset, often by balancing against other codebook properties.

Problem-dependent codebooks are commonly designed by optimizing over codebooks while taking *class-similarity* into account. However, there are two *opposite* intuitions in the literature as to *how* to incorporate class-similarity in the

[*] Equal Contribution.
Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS) 2023, Valencia, Spain. PMLR: Volume 206. Copyright 2023 by the author(s).

design process. Some works follow an intuition that to induce easy subproblems, similar classes should be encoded by *similar* codewords (Zhang et al., 2009; Cissé et al., 2012; Zhao and Xing, 2013; Zhou et al., 2016; Rodríguez et al., 2018). In contrast, other works encode similar classes by *distant* codewords to improve the error correction between hardly-separable classes (Pujol et al., 2008; Martin et al., 2017; Youn et al., 2021; Gupta and Amin, 2021). We examine this *controversy* in depth and provide evidence from multiple regimes that generalization is superior when encoding similar classes by *similar* codewords.

In *predefined* codebooks, the mapping between codewords and classes, i.e., the *codeword-to-class assignment*, is usually set arbitrarily (e.g., using a random assignment). Dietterich and Bakiri (1994) showed that randomly-sampled assignments perform similarly, and since, these assignments have been commonly overlooked.

Our paper shows that *codeword-to-class assignments do matter* and cause a large variation in the performance of many predefined codebooks (Section 4.1.1). We explain this by showing that, given a codebook, some assignments induce substantially easier binary subproblems than other assignments do (Section 4.1.2). Moreover, we show that the easiest subproblems are induced by assigning similar codewords to similar classes (Section 4.1.3).

Finally, we employ our observations on extreme multiclass classification datasets (having 1K to 104K classes). By assigning similar codewords to similar classes, we significantly improve predefined extreme classification codebooks that enjoy fast decoding algorithms (Section 4.2).

To the best of our knowledge, this is the first work to point out the large performance variation explained solely by codeword-to-class assignments, and to explicitly examine these assignments as a means to control the difficulty of the induced learning-subproblems in problem-*independent* predefined codebooks. We conclude that choosing an informed assignment improves predefined codebooks by turning them problem-*dependent* and better suited for the solved task. Importantly, other useful properties of these codebooks are *not* harmed in this process.

2 ERROR-CORRECTING CODES (ECC)

Error-correcting codes are widely used for transmitting messages over noisy channels in communication systems, storage systems, and more. By adding redundant bits to transmitted messages, the receiver can recover messages despite errors caused by a disruptive channel (Roth, 2006).

Training. The seminal work of Dietterich and Bakiri (1994) employed error-correcting codes to encode the K classes of a classification dataset. They set a binary codebook $\mathbf{M} \in \{-1,+1\}^{K\times\ell}$ with $K \in \mathbb{N}$ codewords (each belonging to one class) and ℓ columns (where $\ell \geq \log_2 K$). Each column induces a *binary subproblem*, i.e., a binary partition of classes. Each such subproblem is learned using a base learner $\mathcal{A}$ (i.e., a binary classification learning algorithm), yielding ℓ predictors $f_1,\dots,f_\ell : \mathcal{X} \to \mathbb{R}$. More formally, given a training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $x_i \in \mathcal{X}$ and $y_i \in [K] \triangleq \{1,\dots,K\}$, the jth predictor is the output of $\mathcal{A}$ when trained using the induced binary labels $M_{y_i,j}$:

$$f_j = \mathcal{A}\left(\{(\mathbf{x}_i, M_{y_i,j})\}_{i=1}^m\right) . \quad (1)$$

Prediction. At prediction time, an example $\mathbf{x} \in \mathcal{X}$ is treated as a transmitted message encoding the unknown class $y \in [K]$. The ℓ predictors' scores for $\mathbf{x}$ constitute the vector $\mathbf{f}(\mathbf{x}) \triangleq [f_1(\mathbf{x}),\dots,f_\ell(\mathbf{x})]^\top$. These scores can be prediction margins from a linear model, confidences from a probabilistic model, outputs of a neural network, etc.

Finally, the prediction vector $\mathbf{f}(\mathbf{x})$ is *decoded* into a codeword belonging to a class. The simplest approach is *hard decoding* that consists of finding the nearest neighbor, that is, the codeword closest (in Hamming distance) to the thresholded prediction vector, $\text{sign}(\mathbf{f}(\mathbf{x})) \in \{-1,+1\}^\ell$.

Hard decoding ignores the score magnitudes which entail valuable information for prediction. As a remedy, *soft decoding*, or loss-based decoding (Allwein et al., 2000), minimizes a decoding loss $\mathcal{L} : \mathbb{R} \to \mathbb{R}_{\geq 0}$:

$$\hat{y}(\mathbf{x}) = \underset{y\in[K]}{\arg\min} \sum\nolimits_{j=1}^{\ell} \mathcal{L}\left(M_{y,j} f_j(\mathbf{x})\right) . \quad (2)$$

Two popular decoding losses are the hinge loss $\mathcal{L}(z) = \max\{0, 1-z\}$ and the exponential loss $\mathcal{L}(z) = e^{-z}$. Notice that soft decoding generalizes hard decoding with $\mathcal{L}(z) = \frac{1-\text{sign}(z)}{2}$.

We illustrate the entire ECC scheme in App. A.

Multiclass error upper bound. Allwein et al. (2000) proved an insightful upper bound[1] that will facilitate our discussion throughout this paper. Let

$$\varepsilon \triangleq \varepsilon(\mathbf{M}, \mathcal{L}) = \frac{1}{m\ell} \sum\nolimits_{i=1}^{m} \sum\nolimits_{j=1}^{\ell} \mathcal{L}\left(M_{y_i,j} f_j(\mathbf{x}_i)\right) \quad (3)$$

be the *average binary loss* of the binary predictors on a given training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$ with respect to a codebook $\mathbf{M}$ and a decoding loss $\mathcal{L}$. Assume $\mathcal{L}$ satisfies mild conditions (e.g., convexity is sufficient). Then, the *multiclass training error* when decoding with $\mathcal{L}$ is upper bounded as:

$$\frac{1}{m} \sum\nolimits_{i=1}^{m} \mathbb{I}[y_i \neq \hat{y}(\mathbf{x}_i)] \leq \frac{\ell\varepsilon}{\rho\mathcal{L}(0)} , \quad (4)$$

where $\rho \triangleq \rho(\mathbf{M}) = \min_{a\neq b} \frac{1}{2} \left\|\mathbf{M}_{a,:} - \mathbf{M}_{b,:}\right\|_1$ is the codebook's *minimum inter-row Hamming distance* ($\mathbf{M}_{a,:}$ being the ath row of $\mathbf{M}$) and $\mathcal{L}(0)$ is a scaling factor of $\mathcal{L}$.

[1] Zhou et al. (2019a) derived a bound for more general N-ary codes (where subproblems are also multiclass instead of binary), but this remains out of our scope in this work.

2.1 Properties of a Good Codebook

We now review favorable properties of error-correcting codebooks. The first two properties are discussed more often in the literature (e.g., Dietterich and Bakiri, 1994; Zhang et al., 2003), while the latter two are seldom addressed despite their importance. In many cases improving one property comes at the expense of another.

1. **High minimum row distance ρ (between codewords).** With hard decoding (i.e., nearest neighbor), the maximal number of prediction errors the scheme can recover from is $\lfloor (\rho - 1) / 2 \rfloor$. Using soft decoding, a high minimum distance is still vital for error correction, as seen from the error bound (4).

2. **Low column correlation (between subproblems).** Intuitively, if two binary predictors often make errors on the same inputs, their mistakes become twice as hard to correct. Thus, uncorrelated columns (that yield uncorrelated binary subproblems) are generally considered advantageous.

3. **Efficient decoding algorithm.** Traditionally ignored in many ECC works, the complexity of decoding prediction scores into codewords becomes essential in extreme classification tasks with thousands of codewords or more. Recently, Jasinska and Karampatziakis (2016) and Evron et al. (2018) utilized codebooks with a special structure to allow soft decoding using any decoding loss in a time complexity that depends only on the codebook width ℓ (which can be logarithmic in the number of codewords K). In contrast, exact soft decoding of arbitrary codebooks (e.g., random or optimized ones) requires a time complexity at least linear in K.

4. **Easy binary subproblems (low average loss ε).** The binary subproblems yield binary predictors with an average binary loss ε. The lower this loss is, the better the multiclass accuracy of the scheme becomes (see (4)). One way to lower ε is to use high-capacity base learners (e.g., kernel SVMs), but such rich models are often prone to overfitting or require more computation.

   A proper codebook design can lower ε, by making the subproblems *easier*, even for low-capacity learners. Following are design choices that can achieve this.

   (a) **Sparse or imbalanced codebooks.** Allwein et al. (2000) extended the ECC scheme to ternary codes where $\mathbf{M} \in \{-1, 0, +1\}^{K \times \ell}$. They showed that sparse columns generalize the one-vs-one scheme and that imbalanced columns generalize the one-vs-all scheme. Both options can be seen as ways to create easier subproblems at the expense of the row distance or column correlation.
   See Zhou et al. (2016) and Section 6 in Allwein et al. (2000) for further discussion.

   (b) **Problem-dependent aspects.** Many papers design codebooks that are specifically suitable for the problem at hand while implicitly tuning the difficulty of the binary subproblems.
   Most of these works are guided by notions of class similarity. Some try (implicitly or explicitly) to create codebooks where similar classes have similar codewords (e.g., Cissé et al., 2012) in order to create easier subproblems. Others try the opposite (e.g., Martin et al., 2017) in order to enhance error correction between classes that are hard to separate, at the expense of harder subproblems.
   Notably, most methods balance preserving the similarity against other codebook properties (e.g., the codeword distance between two very similar classes is encouraged to be 1, whereas ρ is encouraged to be maximal). They create codebooks from scratch or alter existing ones. On the other hand, our observations next allow making *pre-defined* codebooks more problem-dependent, by simply assigning codewords to classes in an informed manner, and without harming other codebook properties which may be important.

3 CODEWORD-TO-CLASS ASSIGNMENTS

The error-correcting scheme implicitly assigns codewords to classes. Both during training and during decoding, we arbitrarily assumed that the kth row in the codebook belongs to the kth class (see (1) and (2)). In an attempt to show robustness to codeword-to-class assignments, Dietterich and Bakiri (1994) (Section 3.3.2 therein) experimented on several random assignments and reported no significant accuracy variation. However, they did not rule out the possibility that some assignments *are* better than others.

We hypothesize that some assignments are *significantly* better than others. We first notice that given a codebook, different assignments induce different binary subproblems, potentially changing their difficulty and consequently the average binary loss ε. Next, we define a scoring function that measures the extent to which close codewords are assigned to close classes. This score later helps us conclude that *similarity-preserving* assignments (i.e., similar codewords to similar classes) are preferable.

Class-codeword score. Consider a class metric in the form of a distance matrix $\mathbf{D}_{\text{cls}} \in \mathbb{R}_{\geq 0}^{K \times K}$. For instance, $\mathbf{D}_{\text{cls}}$ can be (inversely proportional to) a symmetrized confusion matrix, a matrix of distances between class embeddings, or a matrix of distances between classes on a hierarchy tree. Define the codeword distance matrix $\mathbf{D}_{\mathbf{M}} \in \mathbb{R}_{\geq 0}^{K \times K}$ where $(\mathbf{D}_{\mathbf{M}})_{a,b} \triangleq \frac{1}{2} \|\mathbf{M}_{a,:} - \mathbf{M}_{b,:}\|_1$. To account for the different scales of these matrices, we normalize them such that $\|\mathbf{D}_{\text{cls}}\|_{\text{F}} = \|\mathbf{D}_{\mathbf{M}}\|_{\text{F}} = 1$.

Notice that an assignment corresponds to reordering, or permuting, the rows of the codebook $\mathbf{M}$ using a $K \times K$ permutation matrix $\mathbf{P}$. Consequently, such an assignment corresponds to permuting the rows *and* columns of the distance matrix $\mathbf{D}_{\mathbf{M}}$.

Given a codebook $\mathbf{M}$ and a class metric $\mathbf{D}_{\text{cls}}$. We assess an assignment, or a permutation $\mathbf{P}$ of the rows in $\mathbf{M}$, by defining the *class-codeword score* as the Frobenius distance between $\mathbf{D}_{\text{cls}}$ and the permuted $\mathbf{D}_{\mathbf{M}}$:

$$s_{\text{cc}}(\mathbf{P}) \triangleq \|\mathbf{D}_{\text{cls}} - \mathbf{D}_{\mathbf{PM}}\|_{\text{F}} = \left\|\mathbf{D}_{\text{cls}} - \mathbf{P}\mathbf{D}_{\mathbf{M}}\mathbf{P}^{\top}\right\|_{\text{F}}. \quad (5)$$

Intuitively, an extreme case where $s_{\text{cc}}(\mathbf{P}) = 0$ means that $\mathbf{D}_{\text{cls}}$ and the permuted $\mathbf{D}_{\mathbf{M}}$ completely "agree", i.e., similar codewords are assigned to similar classes, and dissimilar codewords are assigned to dissimilar classes (realistically, given $\mathbf{D}_{\text{cls}}$ and $\mathbf{D}_{\mathbf{M}}$, the minimum is often larger than zero).

Synthetic dataset. App. B illustrates some of the above ideas using a synthetic dataset. For a specific codebook, we show that only *one* assignment can perfectly fit the data, while *all* other $(K! - 1)$ assignments fail. Moreover, the only successful assignment assigns similar codewords to similar classes.

4 EXPERIMENTS

We test our hypothesis and demonstrate the validity of our claims in two regimes. First, in Section 4.1 we run extensive experiments on small datasets and illustrate how codeword-to-class assignments vary greatly in their accuracy. We show that this variation is mostly explained by the average binary loss ε from (3) and the class-codeword score from (5). We conclude that similarity-preserving assignments are vital for inducing easy binary subproblems. Then, in Section 4.2 we employ similarity-preserving assignments on codebooks for extreme classification. We show how the structure of specific predefined codebooks facilitates finding good assignments and improve performance on datasets with up to 104K classes.

4.1 Exhaustive Experiments

Datasets. We start by testing our hypothesis on 3 small datasets with $K = 10$ classes: `MNIST` (LeCun et al., 1998), `CIFAR-10` (Krizhevsky et al., 2009), and `yeast` (Dua and Graff, 2017).

Table 1: Exhaustive Experiments' Datasets

Dataset	Area	Feat.	Train	Test	Model
`MNIST`	Vision	784	60K	10K	Linear
`CIFAR-10`	Vision	3,072	50K	10K	Linear
`yeast`	Life	8	1,284	200	DT

Codebooks. We experiment on 3 predefined codebooks: Two random dense codebooks (generated like in Allwein et al., 2000) of widths $\ell = 8, 15$ having row distances of $\rho = 3, 5$ (respectively) and a truncated Hadamard matrix (see Hoffer et al., 2018) with $\ell = 15$ and $\rho = 8$.

Experimental setup. Working with only $K = 10$ classes allows us to extensively validate our claims on **all** possible $K! \approx 3.6M$ assignments of each combination of a dataset and a predefined codebook. Notice that given such a combination, we need not *train* $K!$ assignments from scratch. Instead, we train only $2^{K-1} = 512$ binary predictors and construct every possible assignment from them. This technique saves time and decreases the variance of the evaluated test accuracy (details in App. C.1).

To demonstrate the flexibility of our observations, we use two different base learners. For `MNIST` and `CIFAR-10`, we train ℓ linear predictors using the (soft-margin) SVM algorithm. For `yeast`, each binary predictor is a decision tree (built by the Gini splitting criterion and a minimum of 3 samples to split a node). Hyperparameters were tuned using cross-validation (details in App. C.2).

In the decoding step (2), we use the hinge loss, corresponding also to the loss minimized by the SVM used for training the linear base learners.

4.1.1 Variation in performance of assignments

Figure 1 illustrates the large variation in performance for different assignments of given codebooks. For instance, in `MNIST` we observe that using the random dense codebook of width $\ell = 8$, the worst assignment achieves $\approx 77\%$ test accuracy, while the best assignment achieves $\approx 88.5\%$.

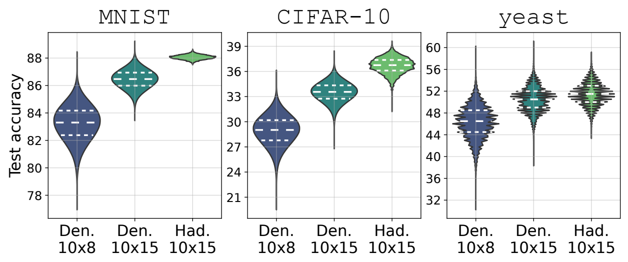
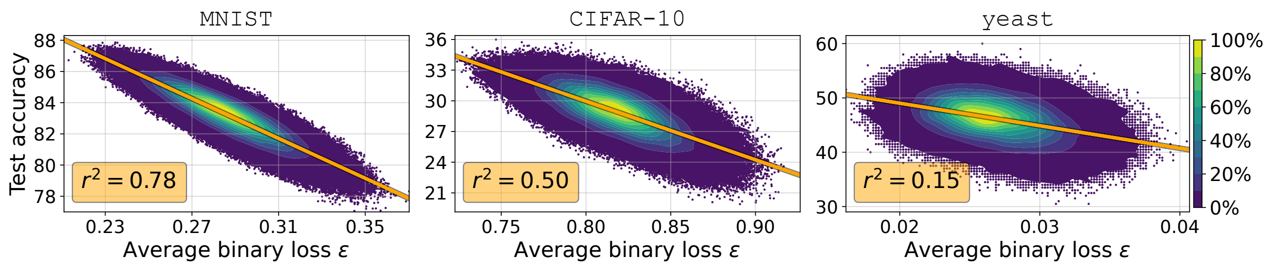
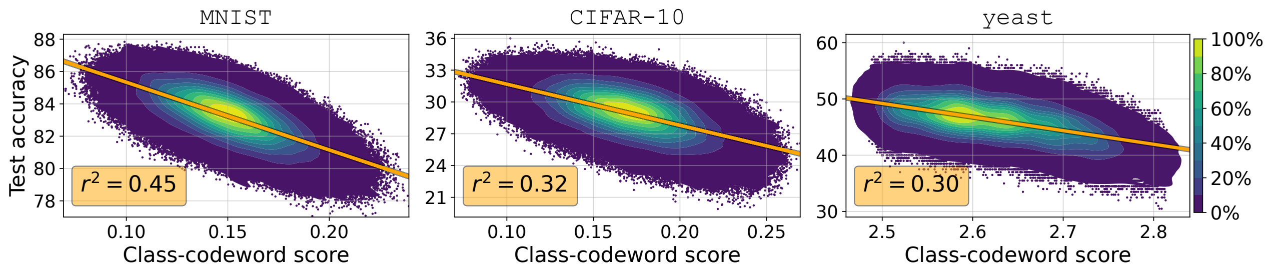


Figure 1: Variation of test accuracy across all $10! \approx 3.6M$ assignments of 3 codebooks on 3 datasets. Dashed lines indicate quartiles (except where the plot is too narrow). There is a large variation in performance across different assignments of the same codebooks.

In all 3 datasets, the narrow $(\ell < K)$ codebook exhibits higher variation in performance. This can be explained by the low minimum distance $(\rho = 3)$ which does not allow for meaningful error correction, making the average binary loss ε a more dominant factor in performance.



(a) Correlation between average binary (train) loss and test accuracy. Assignments that induce easier subproblems perform better.



(b) Correlation between class-codeword score (created by confusion matrices). Similarity-preserving assignments perform better.

Figure 2: The empirical distributions of *all* the 3.6M assignments of the random 10×8 codebook on the 3 datasets. **Top:** Test accuracy vs. average binary (train) loss from (3). **Bottom:** Test accuracy vs. the class-codeword score from (5). Each level set contains $\approx 10\%$ of the assignments. The 10^{-3} least probable assignments are scattered as individual points. Regressors computed on all assignments are plotted in orange. Also written are the coefficients of determination (r^2).

Equidistant codebooks. The low variation in the Hadamard codebook (especially in MNIST) probably stems from it being an *equidistant codebook* (every two codewords are in the same distance from each other). In such codebooks, the class-codeword score (5) remains constant across all assignments (since $\forall \mathbf{P}: \mathbf{D_M} = \mathbf{P}\mathbf{D_M}\mathbf{P}^\top$). This also supports the following findings (Section 4.1.3) that the class-codeword score is a lead factor in the observed performance variation.

4.1.2 Some assignments induce easier subproblems

Figure 2a shows the correlation between the average binary train loss ε and the test accuracy. We plot the empirical distribution (using kernel density estimation) of all 3.6M assignments ran on the 3 datasets using the 10×8 random dense codebook.

For MNIST (top left), the correlation between the test accuracy and ε is the highest ($r^2 = 0.78$). The other two datasets exhibit lower correlations, but large performance gaps are still explained by ε which roughly quantifies the difficulty of subproblems induced by each assignment.

We observe a similar behavior in another 10×8 codebook and a wider 10×15 codebook as well (App. D).

The observed correlation between performance and the average binary loss ε is itself not surprising and can be expected from the error bound in (4). However, our results stress that different *assignments* of the *same* codebook induce binary subproblems of different difficulty.

4.1.3 Similarity-preserving assignments are better

We now test the effect of class similarity on an assignment's performance. We use the class-codeword score (5) to assess how close are codewords of similar classes.

Sources of class similarity. Our class-codeword score requires a matrix $\mathbf{D}_{\text{cls}}$ corresponding to a class metric. Here, we use two *different* class metrics to strengthen our findings. First, we use the (training) confusion matrices of one-vs-all predictors, assuming that confusable classes are semantically similar (a common assumption; see Zhou et al., 2016). Then, in App. D, we use Euclidean distances between the means of raw features of each class. App. C.3 explains how we turn a confusion matrix (a similarity matrix) into a distance matrix.

Results. Figure 2b shows the correlation between our class-codeword score and test accuracy. We use the same random dense 10×8 codebook as before, and compute the class-codeword score from confusion matrices (see above).

For example, the plot on the bottom-middle shows the distribution of all 3.6M assignments ran on CIFAR-10. On average, assigning similar codewords to similar classes (thus minimizing the class-codeword score) improves the test accuracy from $\approx 29\%$ to $\approx 32.5\%$. Moreover, assigning similar codewords to *dissimilar* classes evidently worsens the performance significantly (to $\approx 25.5\%$)

We observe a similar behavior in another 10×8 codebook and a wider 10×15 codebook as well (App. D).

4.1.4 Summary

Some assignments of *the same* codebook induce much easier binary subproblems than others do. Our class-codeword score largely explains the performance of an assignment.

Computing the class-codeword score of one assignment is cheap and mainly requires calculating the distance between two $K \times K$ matrices. Thus, when $K = 10$, exhaustively iterating *all* 3.6M assignments to find the one minimizing that score, takes only a few minutes on a single CPU. Overall, a similarity-preserving assignment found exhaustively *before* training should yield a much better test accuracy than a random assignment.

In App. E we show that the class-codeword score also controls performance in a larger dataset (CIFAR-100), where any exhaustive experiment becomes intractable. We demonstrate that similarity-preserving assignments, originating from the distances between fastText embeddings of *class names*, significantly improve performance.

4.2 Extreme Multiclass Classification (XMC)

We now utilize our understanding that similar codewords should be assigned to similar classes on four XMC benchmarks trained using XMC-dedicated codebooks. We show that in the extreme regime as well — similarity-preserving assignments are significantly better than random ones.

Datasets. We experiment on four XMC preprocessed benchmarks – LSHTC1, LSHTC2 (Partalas et al., 2015), aloi.bin (Rocha and Goldenstein, 2013; Yen et al., 2016), and ODP (Bennett and Nguyen, 2009). The datasets are described briefly below and in detail in App. F.

Table 2: Extreme Benchmarks. Further details in App. F.

Dataset	Area	Classes	Features	Similarity
aloi	Vision	1K	637K	Clustering
LSHTC1	Text	12K	1.2M	Given
LSHTC2	Text	27K	575K	Given
ODP	Text	104K	423K	Clustering

Sources of class similarity. For all datasets, our algorithm below uses class taxonomies given in a form of a tree. These taxonomies are either known in advance (in LSHTC1 and LSHTC2) *or computed* by a simple hierarchical clustering algorithm on class means (in aloi.bin and ODP). Again, using multiple sources of class similarities corroborates the soundness of our findings below.

Experimental setup. We use the code from the publicly available repository of Evron et al. (2018) to learn using their WLTLS codebooks. To use our similarity-preserving codeword-to-class assignments, we edit their scripts to allow for fixed assignments (rather than random ones).[2] We also use the same learning setup — as a base learner, we use AROW (Crammer et al., 2009), which is an online algorithm for learning linear classifiers, and we also use the exponential loss for the soft decoding step in (2). We run all experiments sequentially on a single i7 CPU. In practice, each binary predictor can be trained on a separate CPU.

For each dataset, we train several WLTLS codebooks of various widths ℓ. Each codebook is learned 5 times using random assignments and 5 times using similarity-preserving assignments, found as described below (here, randomness stems from shuffling the training set).

For comparison, we also train one-vs-all (OVA) models using the same base learner – AROW. Our OVA results are better than the ones reported in Evron et al. (2018), since we apply oversampling (Ling and Li, 1998) to overcome the high imbalance in each OVA subproblem.

Finding similarity-preserving assignments. We exploit the graph structure of WLTLS codebooks which embed K codewords on source-to-target paths of a directed acyclic graph (DAG) with exactly K such paths. Since the class taxonomies are also DAGs, a quick-and-simple algorithm arises for assigning similar codewords to similar classes.

Algorithm Sketch: Naive assignment for WLTLS

Input:

1. The dataset's class taxonomy (given or learned)
2. A WLTLS coding DAG suitable for K classes

Algorithm:

```
1. Traverse the class tree with DFS to obtain an ordering
   $(a_1, \dots, a_K)$ of leaves (i.e., classes);
2. Recursively iterate all $K$ paths in the coding DAG, to ob-
   tain an ordering $(b_1, \dots, b_K)$ of paths (i.e., codewords);
3. Assign class $a_i$ to codeword $b_i$.
```

The proposed algorithm preserves similarities by assigning similar classes to similar codewords. Intuitively, in most cases classes a_i and a_{i+1} are close on the taxonomy and paths b_i and b_{i+1} are similar on the codebook's DAG. We illustrate this algorithm in App. F.2.

Despite its simplicity, the algorithm finds assignments with exceptionally low class-codeword scores (5) compared to the scores of random assignments. For example, for the smallest codebook of LSHTC1 ($\ell = 56$), random assignments exhibit an average score of $\widehat{s_{\text{cc}}} \approx 0.061$ with an empirical standard deviation of $1.33 \cdot 10^{-5}$; while the assignment our algorithm finds has a score of $s_{\text{cc}} \approx 0.049$. That is, compared to random assignments, our algorithm decreases the score by more than 900 standard deviations (!).

[2]The updated GitHub repository is available on https://github.com/ievron/wltls

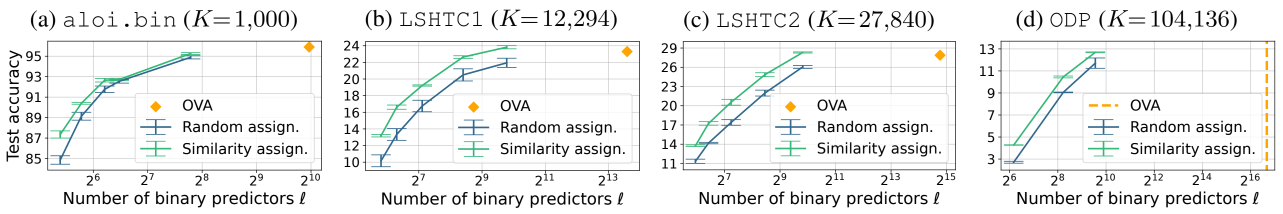


Figure 3: Results on extreme datasets. We run `WLTLS` codebooks using different predictor numbers ℓ. Errorbars indicate ± 2 empirical standard deviations of 5 runs). Results are available in a tabular form in App. F.3. Due to computational infeasibility, we do not report the performance of one-vs-all (OVA) on the largest dataset (`LSHTC2`), but just mark its number of binary predictors instead (where $\ell = K$). In all datasets, assigning similar codewords to similar classes significantly improves performance.

Results. Figure 3 demonstrates the advantage of similarity-preserving codeword-to-class assignments. For each dataset, we compare the test accuracy of random assignments to that of similarity-preserving assignments across various codebook widths ℓ.

We plot the test accuracy averages of the 5 runs of each combination of a codebook width and an assignment method, accompanied by 2 empirical standard deviations (full result tables are given in App. F.3). In almost all cases, similarity-preserving assignments lead to a statistically-significant improvement over random assignments. Moreover, in 16 out of 18 cases, similarity-preserving assignments exhibit a lower variance. In `LSHTC1` and `LSHTC2`, similarity-preserving assignments make the codebooks competitive with OVA while training up to 32 times fewer predictors.

In the two larger codebooks of `aloi.bin`, our assignments do not improve much over random ones. This probably happens because when ℓ approaches K, the underlying `WLTLS` codebooks become almost equidistant.

Summary. Similarity-preserving assignments significantly improve codebooks dedicated to extreme classification. By exploiting class semantics, such assignments turn *predefined* codebooks with regime-specific advantages (e.g., fast decoding algorithms) into problem-dependent codebooks, without losing those advantages.

5 RELATED WORK

Our work is of a retrospective nature and calls for an elaborate discussion of its connections with decades of existing research on error-correcting codes.

Codebooks with easy subproblems are obviously preferable. Bai et al. (2016) design a codebook by selecting a subset of the easiest columns out of all possible columns. They exhaustively train on *all* these columns and select a column subset based on the trained predictors' accuracy. This works well but does not scale gracefully (e.g., for merely $K = 10$ classes, it requires *training* $2^{K-1} = 512$ predictors). Instead, many works (including ours) exploit extra knowledge on classes to create easy subproblems.

Codebook design methods. While we point out that similarity-preserving assignments improve a *predefined* codebook by making it *problem dependent*, most works try to design the *entire* codebook. Given a dataset, designing optimal codebooks is a hard problem due to their discrete nature (Crammer and Singer, 2002). As a remedy, some papers take greedy approaches, e.g., sequentially adding optimized columns (Pujol et al., 2008) or solving integer programming formulations (Gupta and Amin, 2021, 2022); while others take approximate approaches, like solving relaxed continuous optimization problems (e.g., Zhang et al., 2009; Rodríguez et al., 2018).

The class-similarity controversy. Many papers incorporate different notions of class similarity into their design process. Interestingly, some encode similar classes with *similar* codewords (Zhang et al., 2009; Cissé et al., 2012; Zhao and Xing, 2013; Zhou et al., 2016; Rodríguez et al., 2018; McVay, 2020), whereas others encode similar classes with *dissimilar* codewords (Pujol et al., 2008; Martin et al., 2017; Jaiswal et al., 2020; Gupta and Amin, 2021; Wan et al., 2022). For instance, Martin et al. (2017) look for a codebook $\mathbf{M} \in \{-1, +1\}^{K \times \ell}$ that *minimizes* $\left\|\mathbf{D}_{\text{cls}} - \mathbf{M}\mathbf{M}^{\top}\right\|_{\text{F}}^{2}$, while balancing against other codebook properties. In fact, they *maximize* our score (5) instead of minimizing it, since $\mathbf{M}\mathbf{M}^{\top} = \ell \mathbf{1}_{K \times K} - \mathbf{D}_{\mathbf{M}}$.

Existing literature on adversarial robustness has thus far considered assigning *dissimilar* codewords to similar classes (e.g., Gupta and Amin (2021); Wan et al. (2022)). in order to improve the error-correcting capabilities between easily-confusable classes, especially in the presence of an adversary. On the other hand, our study shows that similarity-preserving assignments improve the separability and classification performance in traditional settings. An interesting future direction should be to perform adequate ablation studies in the adversarial learning regime and examine the tradeoff between separability (maximized by similarity-preserving assignments) and robustness (maximized by similarity-breaking ones).

Class similarity in extreme classification (XMC). In Section 4.2 we use a class taxonomy to improve a codebook that requires training very few predictors compared to one-vs-all or hierarchical models. A closely related work (Cissé et al., 2012) designs XMC-codebooks using a learned class-similarity. However, their codebooks do not allow fast decoding like the ones we use. Other related approaches learn hierarchical models using a given (or learned) class taxonomy, to either benefit from a $\mathcal{O}(\log K)$ prediction time (Bengio et al., 2010), or to alleviate the computation of the softmax while training a deep network (Morin and Bengio, 2005). Another approach directly builds a codebook from a class taxonomy (Pujol et al., 2006). However, these approaches train $\mathcal{O}(K)$ predictors, implying longer training and *linear* space requirements. Recently, Mittal et al. (2021) incorporated label metadata in the training of *deep* extreme classification models (much larger than the linear WLTLS models we use). Finally, Rahman et al. (2018) use class semantics to improve zero-shot performance, which may be relevant to XMC tasks which often suffer from a long tail of classes (Babbar et al., 2014), some having few to no training examples.

Ordinal classification and regression tasks can also be tackled with ECC. Interestingly, successful assignments used implicitly in these areas often follow a similar rule-of-thumb like we do – they encode target labels that are similar (i.e., close on the real line) using similar codewords. For instance, see the Unary and HEXJ codebooks in Shah et al. (2022) (the first codebook is equivalent to the underlying codebook in Li and Lin, 2006) or the random ordered-splits in Huhn and Hüllermeier (2008). However, similarities in these areas (i.e., distances on the real line) are much simpler than the inter-class relations examined in our paper.

Nested dichotomies (ND) offer another reduction from multiclass tasks to binary ones. Basically, ND models split classes recursively in a binary hierarchical structure, where each tree node corresponds to a binary classification subproblem. One could either use a single tree (Fox, 1997) or an ensemble of trees (Frank and Kramer, 2004). The resulting models can be seen as a special case of ECC.

Melnikov and Hüllermeier (2018) conduct an experiment that is closely related to our variation demonstration in Section 4.1.1. They show that the assignment of classes to leaves of a *single* ND tree greatly affects the model's performance, and report a high variation in the performance of *randomly-sampled* NDs (the tree structure was also shown to be important in Mnih and Hinton, 2008). However, their tree corresponds to a codebook with a minimum Hamming distance of $\rho = 1$ (i.e., a prediction mistake in *one* inner node necessarily results in a multiclass error). Thus, it is not immediate that their findings generalize to codebooks with higher error-correcting capabilities (like the ones we use). Importantly, we do not only point out the performance variation of codeword-to-class assignments, but also clearly show it is explained by class-similarity (Section 4.1.3).

Model capacity. Codeword-to-class assignments control the difficulty of the binary subproblems, which is naturally more crucial when the base learners are weaker (see the ε/ρ factor in (4)). Related phenomena have been exhibited in ordinal classification (Huhn and Hüllermeier, 2008) and nested dichotomies (Melnikov and Hüllermeier, 2018) as well. In this paper, we demonstrated our findings using relatively weak linear models and decision trees over raw features (Section 4.1) and preprocessed ones (Section 4.2; App. E).

Even high-capacity models like neural networks are likely to favor similarity-preserving assignments. Zhai and Wu (2019) show that a deep classification network implicitly performs *metric learning* — training embeds the classes' weight vectors in the last linear layer (preceding the softmax) in a way that reflects underlying class semantics (see also Kusupati et al. (2021)). Similarity-preserving codebooks can be seen as fixing the last layer using a matrix that already reflects such semantics at initialization (see also Sec. 3.3 in Hoffer et al., 2018).

Notably, complex models can attain a very low average training binary loss ε such that the training error bound (4) becomes $< 1/m$, implying no *training* mistakes. However, this does not make assignments unimportant. If, for example, we train and decode using an exponential loss, then complex learners can obtain an extremely low loss ε, but never 0. In such cases, similarity-preserving assignments should *still* yield a lower ε. In turn, a lower training loss, even when the error is already 0, is linked to better generalization, both theoretically and practically (e.g., Soudry et al. (2018)).

Limitations of design methods. Similarity-preserving assignments can enhance almost *any* predefined codebook, while design methods are often restricted to codebooks with certain properties. For instance, the spectral method (Zhang et al., 2009) creates only narrow codebooks (where $\ell \leq K$) and does not explicitly take the minimum row distance ρ into account, which may not be best suited for small datasets (e.g., on CIFAR-10 with $\ell = 8$, their method yielded two identical rows). Other methods scale poorly with the number of classes (Bai et al., 2016; Escalera et al., 2008). Some are more suitable for creating balanced dense columns (Zhang et al., 2009; Rodríguez et al., 2018) while others focus on sparse columns (Pujol et al., 2006).

Limitations of finding informed assignments. Designing problem-dependent codebooks from scratch is naturally more flexible than only assigning classes to predefined ones. Objective scores can be optimized more freely when the codebook itself is not fixed like in predefined codebooks. However, predefined codebooks can have favorable properties like fast decoding algorithms, hence it is important to be able to find informed assignments for them.

We use our class-codeword score $\left\|\mathbf{D}_{\text{cls}} - \mathbf{P}\mathbf{D}_{\mathbf{M}}\mathbf{P}^{\top}\right\|_{\text{F}}$ mainly to demonstrate the superiority of similarity-preserving assignments (Section 4.1.3). One could also employ this score as a surrogate to control the difficulty of subproblems, and directly minimize it on a given codebook to find an optimal similarity-preserving assignment. However, finding this optimum corresponds to solving a weighted graph-matching problem, which does not have a known efficient algorithm (Umeyama, 1988). Instead, one could settle for assignments with a low (but possibly sub-optimal) score. We exemplify this using a local search on a CIFAR-100 codebook (App. E). An exception where our score is constant and assignments are less impactful, is in equidistant codebooks (e.g., Hadamard, OVA, OVO; see Section 4.1.1). This suggests that equidistant codebooks are perhaps more suitable when no class semantics are available. They can also be expected to yield smaller variation (see Figure 1). See also James and Hastie (1998) who linked such codebooks to Bayes optimality. As a downside, these codebooks must be wide ($\ell \geq K$), which is unacceptable in many cases such as extreme classification.

Greedy assignment policies. After submitting our paper, we became aware of two recent works closely related to ours that also improve the performance of a given codebook using codeword-to-class assignments. McVay (2020) exploits a sparse class-similarity matrix to greedily assign similar codewords to similar classes. Wan et al. (2022) employ ECC for adversarial learning, by altering a Hadamrd codebook (to break its equidistance property) and using a confusion matrix to greedily assign *dissimilar* codewords to similar classes (in contrast to our policy; see the discussion on this controversy above).

Both these works focus on specific greedy assignment policies for specific codebooks. We on the other hand extensively test our hypotheses on many codebooks and demonstrate the superiority of similarity-preserving assignments over similarity-breaking ones in traditional classification settings. We exhaustively evaluate *all* possible assignments in several codebooks on three small datasets (see Figure 2 and App. D); and also evaluate different greedy assignment policies on larger datasets (see 4.2 and App. E).

6 CONCLUSION

Codeword-to-class assignments matter because they vary greatly in the difficulty of subproblems they induce, even for a *predefined* codebook. In classification tasks (of both small and large scales), similarity-preserving assignments lead to easier subproblems and better generalization performance. Predefined codebooks can be advantageous when certain properties are crucial, e.g., specific minimum distance ρ and number of predictors ℓ, a given sparsity level, or an efficient decoding algorithm. Choosing an informed assignment according to class semantics, allows for improving predefined codebooks by making them more problem-dependent.

Further research might discover that different usages require different assignment policies. For instance, perhaps similarity-preserving assignments benefit generalization, while similarity-breaking assignments benefit robustness (see the discussion in Section 5).

Acknowledgements

We thank Koby Crammer and Thomas G. Dietterich for the fruitful discussions. The research of DS was Funded by the European Union (ERC, A-B-C-Deep, 101039436). Views and opinions expressed are however those of the author only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency (ERCEA). Neither the European Union nor the granting authority can be held responsible for them. DS also acknowledges the support of Schmidt Career Advancement Chair in AI. Finally, we thank the Control Robotics & Machine Learning (CRML) Lab at the Technion for their support.

References

Erin L Allwein, Robert E Schapire, and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of machine learning research*, 1(Dec):113–141, 2000. (cited on p. 2, 3, 4, 13, 21)

Rohit Babbar, Cornelia Metzig, Ioannis Partalas, Eric Gaussier, and Massih-Reza Amini. On power law distributions in large-scale taxonomies. *ACM SIGKDD explorations newsletter*, 16(1):47–56, 2014. (cited on p. 8)

Xiaolong Bai, Swamidoss Issac Niwas, Weisi Lin, Bing-Feng Ju, Chee Keong Kwoh, Lipo Wang, Chelvin C Sng, Maria C Aquino, and Paul TK Chew. Learning ecoc code matrix for multiclass classification with application to glaucoma diagnosis. *Journal of medical systems*, 40 (4):78, 2016. (cited on p. 7, 8, 15)

Samy Bengio, Jason Weston, and David Grangier. Label embedding trees for large multi-class tasks. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23, pages 163–171. Curran Associates, Inc., 2010. (cited on p. 8)

Paul N Bennett and Nam Nguyen. Refined experts: improving classification in large taxonomies. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 11–18, 2009. (cited on p. 6, 23)

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016. (cited on p. 21)

Moustapha Cissé, Thierry Artieres, and Patrick Gallinari. Learning compact class codes for fast inference in large multi class classification. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 506–520. Springer, 2012. (cited on p. 2, 3, 7, 8)

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995. (cited on p. 15)

Koby Crammer and Yoram Singer. On the learnability and design of output codes for multiclass problems. *Machine learning*, 47(2-3):201–233, 2002. (cited on p. 7)

Koby Crammer, Alex Kulesza, Mark Dredze, et al. Adaptive regularization of weight vectors. In *NIPS*, volume 22, pages 414–422. Citeseer, 2009. (cited on p. 6)

Thomas G Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of artificial intelligence research*, 2:263–286, 1994. (cited on p. 1, 2, 3)

Ulzii-Orshikh Dorj, Keun-Kwang Lee, Jae-young Choi, and Malrey Lee. The skin cancer classification using deep convolutional neural network. *Multimedia Tools and Applications*, 77, 04 2018. doi: 10.1007/s11042-018-5714-1. (cited on p. 1)

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`. (cited on p. 4)

Sergio Escalera, David MJ Tax, Oriol Pujol, Petia Radeva, and Robert PW Duin. Subclass problem-dependent design for error-correcting output codes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30 (6):1041–1054, 2008. (cited on p. 8)

Itay Evron, Edward Moroshko, and Koby Crammer. Efficient loss-based decoding on graphs for extreme classification. *Advances in Neural Information Processing Systems*, 31:7233–7244, 2018. (cited on p. 1, 3, 6, 23, 24)

John Fox. *Applied regression analysis, linear models, and related methods.* Sage Publications, Inc, 1997. (cited on p. 8)

Eibe Frank and Stefan Kramer. Ensembles of nested dichotomies for multi-class problems. In *Proceedings of the twenty-first international conference on Machine learning*, page 39, 2004. (cited on p. 8)

Samarth Gupta and Saurabh Amin. Integer programming-based error-correcting output code design for robust classification. In *Uncertainty in Artificial Intelligence*, pages 1724–1734. PMLR, 2021. (cited on p. 1, 2, 7)

Samarth Gupta and Saurabh Amin. Scalable design of error-correcting output codes using discrete optimization with graph coloring. *Advances in Neural Information Processing Systems*, 35, 2022. (cited on p. 1, 7)

Elad Hoffer, Itay Hubara, and Daniel Soudry. Fix your classifier: the marginal value of training the last weight layer. In *International Conference on Learning Representations*, 2018. (cited on p. 4, 8)

Jens C Huhn and Eyke Hüllermeier. Is an ordinal class structure useful in classifier learning? *International Journal of Data Mining, Modelling and Management*, 1 (1):45–67, 2008. (cited on p. 8)

Mayoore S Jaiswal, Bumsoo Kang, Jinho Lee, and Minsik Cho. Mute: Inter-class ambiguity driven multi-hot target encoding for deep neural network design. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 754–755, 2020. (cited on p. 7)

Gareth James and Trevor Hastie. The error coding method and picts. *Journal of Computational and Graphical statistics*, 7(3):377–387, 1998. (cited on p. 9)

Kalina Jasinska and Nikos Karampatziakis. Log-time and log-space extreme classification. *arXiv preprint arXiv:1611.01964*, 2016. (cited on p. 1, 3)

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images, 2009. (cited on p. 4, 21)

Aditya Kusupati, Matthew Wallingford, Vivek Ramanujan, Raghav Somani, Jae Sung Park, Krishna Pillutla, Prateek Jain, Sham Kakade, and Ali Farhadi. Llc: Accurate, multi-purpose learnt low-dimensional binary codes. *Advances in Neural Information Processing Systems*, 34, 2021. (cited on p. 1, 8)

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. (cited on p. 4)

Ling Li and Hsuan-Tien Lin. Ordinal regression by extended binary classification. *Advances in neural information processing systems*, 19, 2006. (cited on p. 8)

Charles X Ling and Chenghui Li. Data mining for direct marketing: Problems and solutions. In *Kdd*, volume 98, pages 73–79, 1998. (cited on p. 6)

Miguel Angel Bautista Martin, Oriol Pujol, Fernando De la Torre, and Sergio Escalera. Error-correcting factorization. *IEEE transactions on pattern analysis and machine intelligence*, 40(10):2388–2401, 2017. (cited on p. 2, 3, 7)

Paul Robert McVay. *Generalization Bounds for Compressed Learning with Hard Support Vector Machines, and Multiclass Learning with Error Correcting Output Codes*. PhD thesis, Texas A&M University, 2020. (cited on p. 7, 9)

Vitalik Melnikov and Eyke Hüllermeier. On the effectiveness of heuristics for learning nested dichotomies: an empirical analysis. *Machine Learning*, 107(8):1537–1560, 2018. (cited on p. 8)

Anshul Mittal, Kunal Dahiya, Sheshansh Agrawal, Deepak Saini, Sumeet Agarwal, Purushottam Kar, and Manik Varma. Decaf: Deep extreme classification with label features. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 49–57, 2021. (cited on p. 8)

Andriy Mnih and Geoffrey E Hinton. A scalable hierarchical distributed language model. *Advances in neural information processing systems*, 21, 2008. (cited on p. 8)

Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252. Citeseer, 2005. (cited on p. 8)

Ioannis Partalas, Aris Kosmopoulos, Nicolas Baskiotis, Thierry Artières, George Paliouras, Éric Gaussier, Ion Androutsopoulos, Massih-Reza Amini, and Patrick Gallinari. LSHTC: A benchmark for large-scale text classification. *CoRR*, abs/1503.08581, 2015. (cited on p. 6, 23)

Oriol Pujol, Petia Radeva, and Jordi Vitria. Discriminant ecoc: A heuristic method for application dependent design of error correcting output codes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28 (6):1007–1012, 2006. (cited on p. 8)

Oriol Pujol, Sergio Escalera, and Petia Radeva. An incremental node embedding technique for error correcting output codes. *Pattern Recognition*, 41(2):713–725, 2008. (cited on p. 2, 7)

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007. (cited on p. 23)

Shafin Rahman, Salman Khan, and Fatih Porikli. Zero-shot object detection: Learning to simultaneously recognize and localize novel concepts. In *Asian Conference on Computer Vision*, pages 547–563. Springer, 2018. (cited on p. 8)

Anderson Rocha and Siome Klein Goldenstein. Multiclass from binary: Expanding one-versus-all, one-versus-one and ecoc-based approaches. *IEEE Transactions on Neural Networks and Learning Systems*, 25(2):289–302, 2013. (cited on p. 6, 23)

Pau Rodríguez, Miguel A Bautista, Jordi Gonzalez, and Sergio Escalera. Beyond one-hot encoding: Lower dimensional target embedding. *Image and Vision Computing*, 75:21–31, 2018. (cited on p. 1, 2, 7, 8)

Ron M Roth. Introduction to coding theory. *IET Communications*, 47, 2006. (cited on p. 2)

Deval Shah, Zi Yu Xue, and Tor Aamodt. Label encoding for regression networks. In *International Conference on Learning Representations*, 2022. (cited on p. 1, 8)

Yang Song, Qiyu Kang, Wee Peng Tay, and Y Tay. Error-correcting output codes with ensemble diversity for robust learning in neural networks. In *AAAI*, pages 9722–9729, 2021. (cited on p. 1)

Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. The implicit bias of gradient descent on separable data. *JMLR*, 2018. (cited on p. 8)

Shinji Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE transactions on pattern analysis and machine intelligence*, 10(5): 695–703, 1988. (cited on p. 9, 16)

Gunjan Verma and Ananthram Swami. Error correcting output codes improve probability estimation and adversarial robustness of deep neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 8646–8656. Curran Associates, Inc., 2019. (cited on p. 1)

Li Wan, Tansu Alpcan, Emanuele Viterbo, and Margreta Kuijper. Efficient error-correcting output codes for adversarial learning robustness. In *ICC 2022-IEEE International Conference on Communications*, pages 2345–2350. IEEE, 2022. (cited on p. 7, 9)

Chen Xing, Negar Rostamzadeh, Boris Oreshkin, and Pedro O O. Pinheiro. Adaptive cross-modal few-shot learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. (cited on p. 21)

Shuo Yang, Ping Luo, Chen Change Loy, Kenneth W Shum, and Xiaoou Tang. Deep representation learning with target coding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015. (cited on p. 1)

Ian En-Hsu Yen, Xiangru Huang, Pradeep Ravikumar, Kai Zhong, and Inderjit Dhillon. Pd-sparse: A primal and dual sparse approach to extreme multiclass and multilabel classification. In *International Conference on Machine Learning*, pages 3069–3077. PMLR, 2016. (cited on p. 6, 23)

Hwiyoung Youn, Soonhee Kwon, Hyunhee Lee, Jiho Kim, Songnam Hong, and Dong-Joon Shin. Construction of error correcting output codes for robust deep neural networks based on label grouping scheme. In *2021 7th IEEE International Conference on Network Intelligence and Digital Content (IC-NIDC)*, pages 51–55. IEEE, 2021. (cited on p. 2)

Andrew Zhai and Hao-Yu Wu. Classification is a strong baseline for deep metric learning. In *BMVC*, 2019. (cited on p. 8)

Aijun Zhang, Zhi-Li Wu, Chun-Hung Li, and Kai-Tai Fang. On hadamard-type output coding in multiclass learning. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 397–404. Springer, 2003. (cited on p. 1, 3)

Xiao Zhang, Lin Liang, and Heung-Yeung Shum. Spectral error correcting output codes for efficient multiclass recognition. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1111–1118. IEEE, 2009. (cited on p. 2, 7, 8, 18)

Bin Zhao and Eric P Xing. Sparse output coding for large-scale visual recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3350–3357, 2013. (cited on p. 2, 7)

Jiewan Zheng, Xianbin Cao, Baochang Zhang, Xiantong Zhen, and Xiangbo Su. Deep ensemble machine for video classification. *IEEE transactions on neural networks and learning systems*, 30(2):553–565, 2018. (cited on p. 1)

Jindeng Zhou, Yun Yang, Mingjie Zhang, and Haibo Xing. Constructing ecoc based on confusion matrix for multiclass learning problems. *Science China Information Sciences*, 59(1):1–14, 2016. (cited on p. 2, 3, 5, 7)

Joey Tianyi Zhou, Ivor W Tsang, Shen-Shyang Ho, and Klaus-Robert Müller. N-ary decomposition for multiclass classification. *Machine Learning*, 108(5):809–830, 2019a. (cited on p. 2)

Joey Tianyi Zhou, Ivor W Tsang, Sinno Jialin Pan, and Mingkui Tan. Multi-class heterogeneous domain adaptation. *Journal of Machine Learning Research*, 2019b. (cited on p. 1)

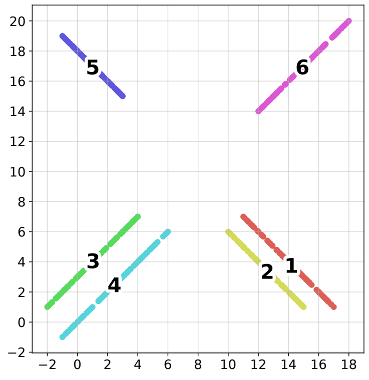
The Role of Codeword-to-Class Assignments in Error-Correcting Codes: Supplementary Materials

A Error Correcting Codes: Illustration

To make our paper more approachable for readers who are less familiar with the Error-Correcting Codes scheme (Section 2), we now present a brief illustration of the entire scheme. For further explanations, we recommend Section 3 in Allwein et al. (2000).

In this section and the next, we use a synthetic dataset with $K = 6$ classes and $m = 600$ training samples (100 per class). The dataset is illustrated in Figure 4a.

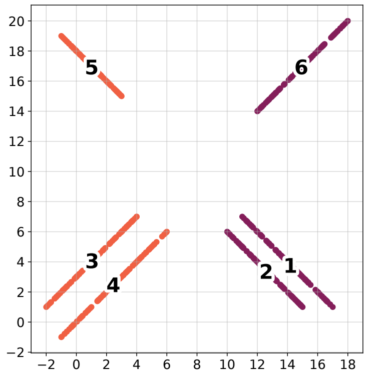
For simplicity, in Figure 4b we present a small codebook with $\ell = 3$ columns and no redundancy. Each column of the codebook $\mathbf{M}$ induces a binary subproblem. One such subproblem, corresponding to the leftmost column, is depicted in Figure 4c and requires separating classes $1, 2, 6$ from $3, 4, 5$. Each binary subproblem is learned by a model of choice, e.g., SVM or a decision tree, yielding ℓ binary predictors $f_1, \ldots, f_\ell$.



(a) The synthetic dataset

$$\begin{array}{ccc} \mathbf{M}_{:,1} & \mathbf{M}_{:,2} & \mathbf{M}_{:,3} \\ \hline -1 & -1 & -1 \\ -1 & -1 & 1 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \\ -1 & 1 & -1 \end{array}$$

(b) A given codebook $\mathbf{M}$



(c) Subproblem #1

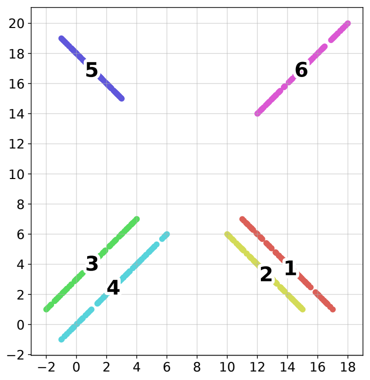
Figure 4: Given the synthetic dataset (left) and a given codebook (center), assignments vary greatly in their accuracy (right). The best assignments achieve 100% accuracy, while the worst achieve 37.17%.

At test time, given an input $\mathbf{x}$, the binary predictors output a prediction vector $\mathbf{f}(\mathbf{x}) \triangleq [f_1(\mathbf{x}), \ldots, f_\ell(\mathbf{x})]^\top$. In turn, the final prediction is made either by thresholding $\mathbf{f}(\mathbf{x})$ and looking for the nearest neighbor (row) of the codebook $\mathbf{M}$, or by a more sophisticated decoding scheme that takes into account the prediction magnitudes as well (see (2)). For instance, if $\mathbf{f}(\mathbf{x}) = [0.1, -3, 2.4]^\top$, then a hard decoding scheme, which is equivalent to nearest-neighbor decoding, will compute $\text{sign}\,(\mathbf{f}(\mathbf{x})) = [1, -1, 1]^\top$, and the prediction would be $\hat{y}(\mathbf{x}) = 4$ (see the fourth row in Figure 4b).

B Synthetic dataset

Here we illustrate the importance of codeword-to-class assignments using the synthetic dataset from the previous section ($K = 6$ classes, $m = 600$ training samples). For simplicity, we use a small codebook with $\ell = 3$ columns and no redundancy.

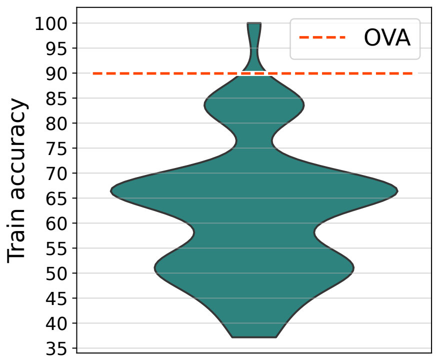
For this codebook, only 12 out of 720 assignments can fit the data perfectly with a linear predictor. These assignments all correspond to the same codebook (since the column order does not matter in ECC schemes and since complementary binary partitions are equivalent). These assignments also beat one-vs-all (OVA) trained with a (tuned) linear SVM that achieves only 89.83% (setting the Soft-SVM's C as 0.89). Finally, the best assignments apparently preserve similarity (see Figure 6 and compare the codewords of the neighboring classes #1 and #2 to those of #4 and #6).



(a) The synthetic dataset

$\mathbf{M}_{:,1}$	$\mathbf{M}_{:,2}$	$\mathbf{M}_{:,3}$
-1	-1	-1
-1	-1	1
1	1	1
1	-1	1
1	1	-1
-1	1	-1

(b) A given codebook $\mathbf{M}$



(c) Accuracy variation of assignments

Figure 5: Given the synthetic dataset (left) and a given codebook (center), assignments vary greatly in their accuracy (right). The best assignments achieve 100% accuracy, while the worst achieve 37.17%.

Now we illustrate *why* the subproblems induced by the best assignment are inherently easier than the ones induced by the worst assignment (using the same codebook).

class	$\mathbf{M}_{:,1}$	$\mathbf{M}_{:,2}$	$\mathbf{M}_{:,3}$
#1	-1	-1	-1
#2	-1	-1	1
#3	1	1	1
#4	1	-1	1
#5	1	1	-1
#6	-1	1	-1

(a) Assigned codebook

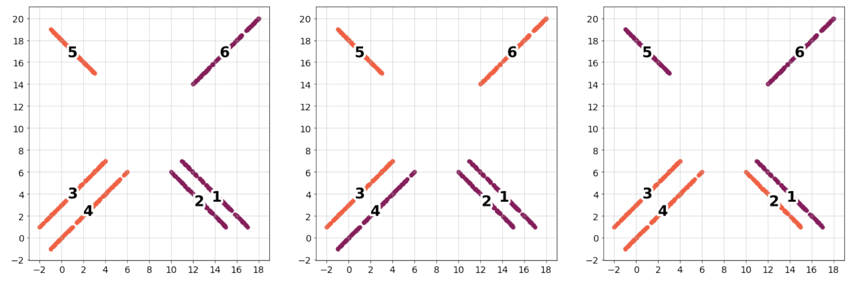

(b) Subproblem #1 (c) Subproblem #2 (d) Subproblem #3

Figure 6: Subproblems induced by the *best* assignment (acc. = 100%) are linearly *separable*.

class	$\mathbf{M}_{:,1}$	$\mathbf{M}_{:,2}$	$\mathbf{M}_{:,3}$
#1	-1	-1	-1
#2	-1	-1	1
#6	1	1	1
#3	1	-1	1
#4	1	1	-1
#5	-1	1	-1

(a) Assigned codebook

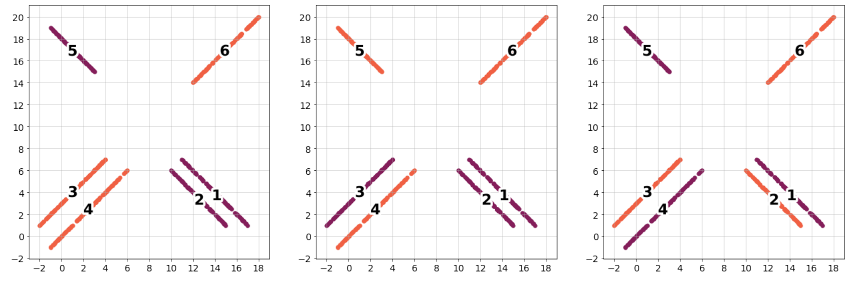

(b) Subproblem #1 (c) Subproblem #2 (d) Subproblem #3

Figure 7: Subproblems induced by the *worst* assignment (acc. = 37.17%) are linearly *inseparable*.

C Training details for the exhaustive experiments in Section 4.1

C.1 Evaluating all possible assignments

The exhaustive experiments require obtaining the test accuracy of every possible codeword-to-class assignment of given codebooks. Following are the training details of our experimental setup. This exhaustive setup resembles of the setup in Bai et al. (2016), but we use it as a means for simply showing that similar codewords should be assigned to similar classes, while they propose it as a practical approach for small datasets (with very few classes).

The datasets used in the exhaustive experiments have $K = 10$ classes each. This means that each dataset has $K! \approx 3.6M$ assignments. Instead of training every assignment from scratch, we notice that there are at most $2^K = 1,024$ possible binary columns with $K = 10$ rows. We further notice that a column $\mathbf{M}_{:,j}$ and its complementary column $-\mathbf{M}_{:,j}$ create the same binary classification task (with opposite labels), thus reducing the number of possible binary partitions to $2^{K-1} = 512$. Finally, columns consisting of only $+1$ (or -1) induce meaningless partitions. Hence, the number of binary partitions we actually need to train on is $2^{K-1} - 1 = 511$.

The aforementioned 511 columns constitute every possible codebook with $K = 10$ codewords. Specifically, given a codebook, all possible assignments correspond to all possible row permutations of the codebook. Thus, instead of training 3.6M codebooks, we train only 511 columns, construct every possible assignment (permuted codebook) from the pretrained columns, and finally, merely check the test accuracy of the resulting codebook.

Clearly, the trick above reduces the time required for our experiments (since training is much more expensive than inference). Moreover, it reduces the variation in the test accuracies stemming from the training itself, since a binary partition that appears in multiple assignments is only trained once. This makes the observed variation in Section 4.1.1 more significant.

C.2 Hyperparameter tuning

We now explain how we train the 511 binary partitions of each dataset.

C.2.1 `MNIST` and `CIFAR-10`

For these two datasets, we use the (soft-margin) SVM algorithm (Cortes and Vapnik, 1995) as the base learner. The SVM problem is

$$\min_{\boldsymbol{w},b} \frac{1}{2} \|\boldsymbol{w}\|^2 + C \sum_{i=1}^{m} \max\{0,\, 1 - y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)\} \,,$$

where $C > 0$ is a regularization parameter that requires tuning.

To tune C, we perform 3-fold cross-validation on the training sets (only), evaluating the performance of $C \in \{10^{-3}, 10^{-2}, \ldots, 10^{2}, 10^{3}\}$. Consequently, we choose $C = 10$ for `MNIST` and $C = 1$ for `CIFAR-10`.

C.2.2 `yeast`

For this dataset, the base learners are decision trees with the Gini splitting criterion. Nodes are split until they are pure or until they contain less than 3 examples. The minimum sample-number for splitting (i.e., 3) was chosen from $\{1, 3, 5, 7\}$ after we found it yielded the best validation performance.

C.3 Building class metrics

The class-codeword score (5) described in Section 3 requires a class metric in the form of a distance matrix $\mathbf{D}_{\text{cls}} \in \mathbb{R}_{\geq 0}^{K\times K}$. In Section 4.1.3 we construct the aforementioned class distance matrix in two ways: (a) using a confusion matrix, and (b) using class means (of raw features). Later in App. E, we also use word embeddings of class names. Details follow.

C.3.1 Using a confusion matrix

We use confusion matrices as a similarity measure on classes, assuming confusable classes are semantically similar. For each dataset, we train a one-vs-all (OVA) classifier and compute its confusion matrix. For `MNIST` and `CIFAR-10`, we compute the confusion matrix on the training set itself (since there are sufficient errors for the matrix to be informative). For `yeast`, there are very few training samples and the models have a high capacity (decision trees), resulting in very few training errors and a sparse confusion matrix. Thus, we split the training set of `yeast` into 8 folds. We train on one fold and compute the confusion matrix on the other 7 folds. Finally, we sum the 8 resulting confusion matrices.

Each confusion matrix $\mathbf{C}_{\text{cls}}$ is an asymmetric similarity matrix (where the sum of all entries is 1), while we require $\mathbf{D}_{\text{cls}}$ to be a symmetric distance matrix. At first, we considered using

$$\mathbf{A} \triangleq \mathbf{1}_{K\times K} - \frac{1}{2}\left(\mathbf{C}_{\text{cls}} + \mathbf{C}_{\text{cls}}^{\top}\right) , \tag{6}$$

which is a symmetric dissimilarity measure as required. However, since the entries of $\mathbf{C}_{\text{cls}}$ are often very close to 0, the above transformation yields a matrix that is very close to a rank-one matrix which is substantially different from $\mathbf{D}_{\mathbf{M}}$ (the eigenvalue analysis in (Umeyama, 1988) shows why this is problematic).

To induce reasonable spectra, we used the following matrix (simpler matrices yield mostly similar results in our experiments):

$$\begin{aligned} &\forall i : D_{i,i} = 0, \\ &\forall i \neq j : D_{i,j} = \log\left(A_{i,j}\right) - 1.1 \min_{i' \neq j'} \log\left(A_{i',j'}\right) , \end{aligned} \tag{7}$$

which yields a symmetric dissimilarity matrix, with a vast spectrum of eigenvalues, as depicted below. In practice, the $\mathbf{D}_{\text{cls}}$ matrices stemming from the above approach are very informative and create valuable class-codeword scores which are highly correlated with the test accuracy (see Section 4.1.3).

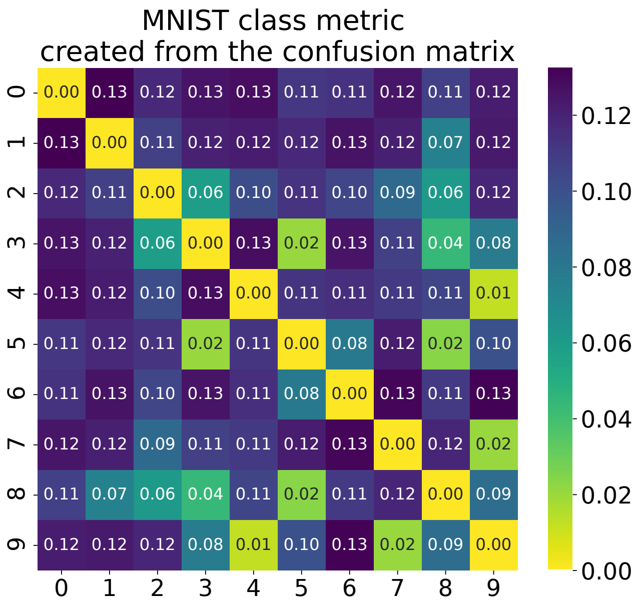
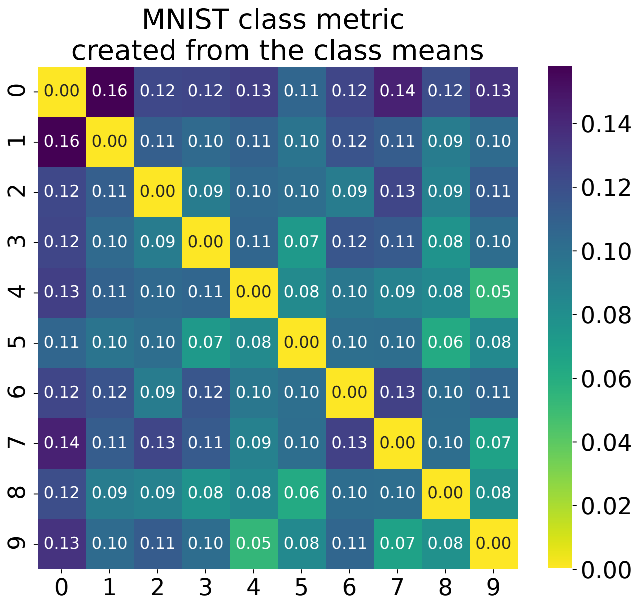


Figure 8: The class metric matrix $\mathbf{D}_{\text{cls}}$ of `MNIST`, constructed from the confusion matrix as described above. Notice how this matrix quantifies visual class semantics (similarities). For instance, see how 4 has a small distance from 9 and how 3 is close to $2, 5, 8$, etc.

C.3.2 Using class means of raw features

For each dataset, we embed each class in a high-dimensional Euclidean space by computing the means of the *raw* features of all *training* samples of that class. Then, we set $\mathbf{D}_{\text{cls}}$ as the matrix of Euclidean distances between these embeddings. The resulting matrices are symmetric and the class-codeword scores stemming from them are informative, as seen from the apparent correlations to the test accuracy (see App. D).

Importantly, these embeddings do not require actually training other models (unlike confusion matrices). Moreover, since we use simple base learner (e.g., linear models), one should expect that similarity-preserving assignments according to the proposed $\mathbf{D}_{\text{cls}}$ will "concentrate" the samples of each binary class in each of the induced binary subproblems in the Euclidean space, thus creating easier subproblems.



	0	1	2	3	4	5	6	7	8	9
0	0.00	0.16	0.12	0.12	0.13	0.11	0.12	0.14	0.12	0.13
1	0.16	0.00	0.11	0.10	0.11	0.10	0.12	0.11	0.09	0.10
2	0.12	0.11	0.00	0.09	0.10	0.10	0.09	0.13	0.09	0.11
3	0.12	0.10	0.09	0.00	0.11	0.07	0.12	0.11	0.08	0.10
4	0.13	0.11	0.10	0.11	0.00	0.08	0.10	0.09	0.08	0.05
5	0.11	0.10	0.10	0.07	0.08	0.00	0.10	0.10	0.06	0.08
6	0.12	0.12	0.09	0.12	0.10	0.10	0.00	0.13	0.10	0.11
7	0.14	0.11	0.13	0.11	0.09	0.10	0.13	0.00	0.10	0.07
8	0.12	0.09	0.09	0.08	0.08	0.06	0.10	0.10	0.00	0.08
9	0.13	0.10	0.11	0.10	0.05	0.08	0.11	0.07	0.08	0.00

Figure 9: The class metric matrix $\mathbf{D}_{\text{cls}}$ of `MNIST`, constructed from the class raw feature means as described above. Notice how this matrix quantifies visual class semantics (similarities) as well, and how it resembles the metric created from the confusion matrix (Figure 8).

D All correlation graphs for the exhaustive experiments in Section 4.1

Now we show similar results and correlations to the ones shown in Section 4.1 for additional two codebooks — Random dense 10×15 and Spectral 10×8 (built using the method from Zhang et al., 2009). Moreover, we also report results using class-codeword scores stemming from the means of the raw features of each class rather than confusion matrices (see App. C.3.2).

Our observations and findings evidently apply to many codebooks and class metrics (see App. E for an additional metric).

How to understand the plots? Each level set contains $\approx 10\%$ of all possible 3.6M assignments. The 10^{-3} least probable assignments are scattered as individual points. Regressors computed on all assignments are plotted in orange. Also written are the coefficients of determination (r^2).

D.1 `MNIST`

In `MNIST`, the average binary loss ε is highly correlated with the test accuracy for the three tested codebooks. Our method for constructing $\mathbf{D}_{\text{cls}}$ from confusion matrices (described in App. C.3.1) yields class-codeword scores that are correlated to the test accuracy (but less than the average binary loss). Finally, the class-codeword scores stemming from the means of the raw features (see App. C.3.2) are the least correlated to the test accuracy, but are still informative.

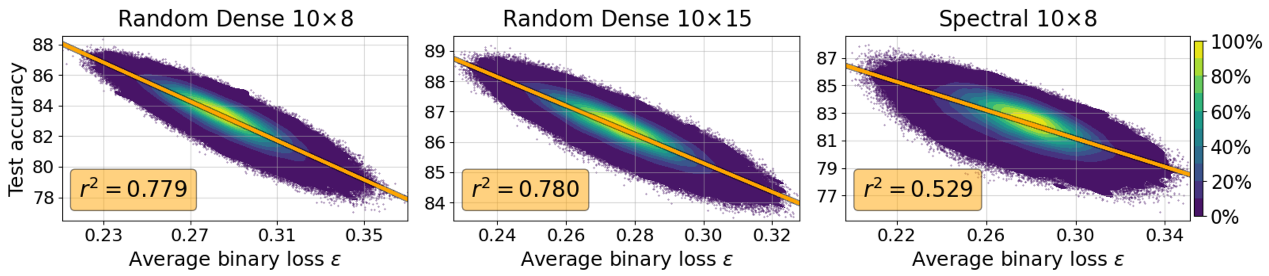


Figure 10: Test accuracy vs. Average binary loss in `MNIST`.

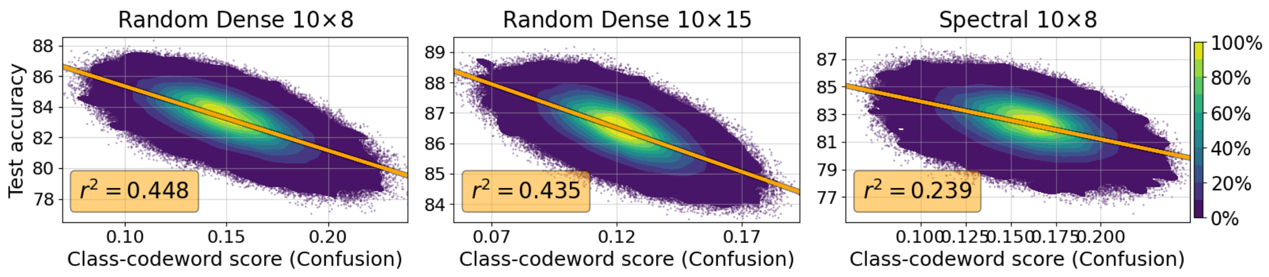


Figure 11: Test accuracy vs. Class-codeword scores in `MNIST`, using the confusion matrix to construct $\mathbf{D}_{\text{cls}}$.

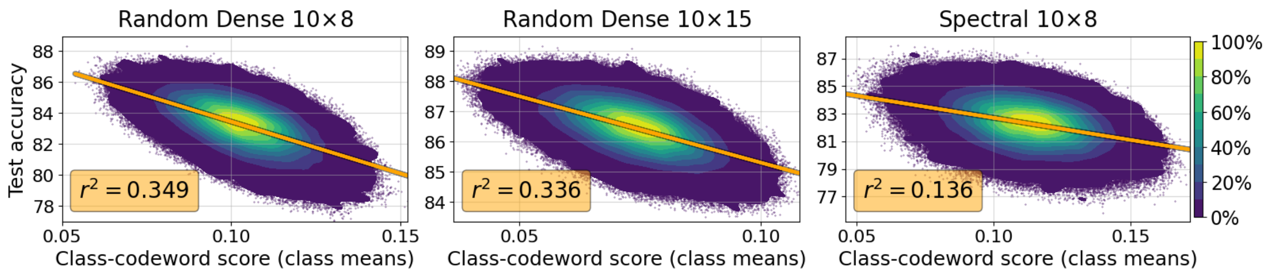


Figure 12: Test accuracy vs. Class-codeword scores in `MNIST`, using class (feature) means to construct $\mathbf{D}_{\text{cls}}$.

D.2 CIFAR-10

In CIFAR-10, the test accuracy is less correlated to the average binary loss compared to the correlation in MNIST. However, the class-codeword scores computed using the confusion matrices are very informative (sometimes even comparable with the average binary loss). The class means are again slightly less informative than the confusion matrices.

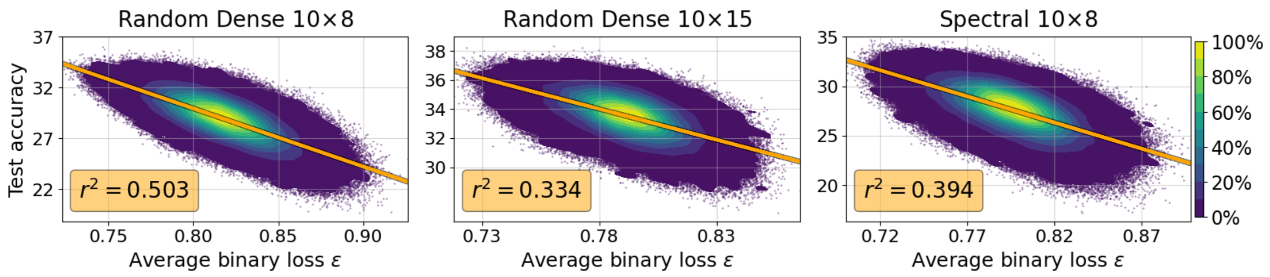


Figure 13: Test accuracy vs. Average binary loss in CIFAR-10.

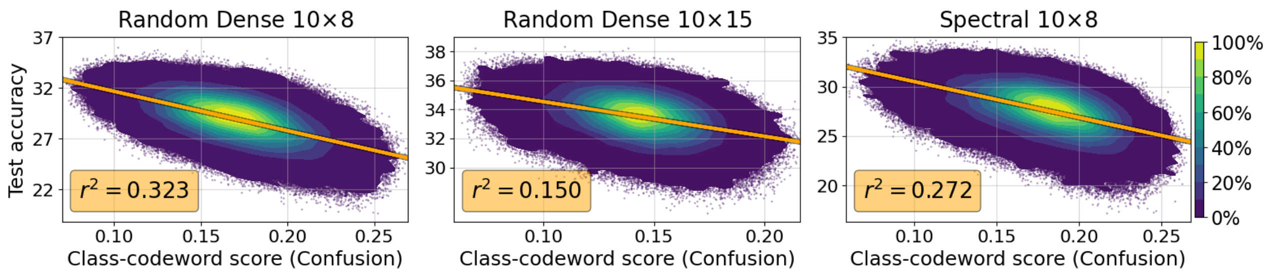


Figure 14: Test accuracy vs. Class-codeword scores in CIFAR-10, using the confusion matrix to construct $\mathbf{D}_{\text{cls}}$.

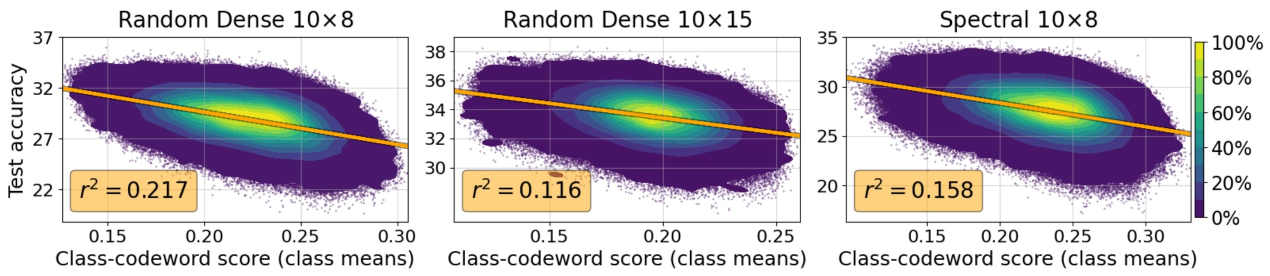


Figure 15: Test accuracy vs. Class-codeword scores in CIFAR-10, using class (feature) means to construct $\mathbf{D}_{\text{cls}}$.

D.3 `yeast`

`yeast` exhibits the worst correlations among the three datasets, but the assignments still evidently vary, and their accuracy is mildly controlled by the class-codeword scores. One should also notice that this dataset is much smaller than the other two (in both the number of training examples and number of features, see Table 1), which might explain the larger variation and lower correlations it exhibits.

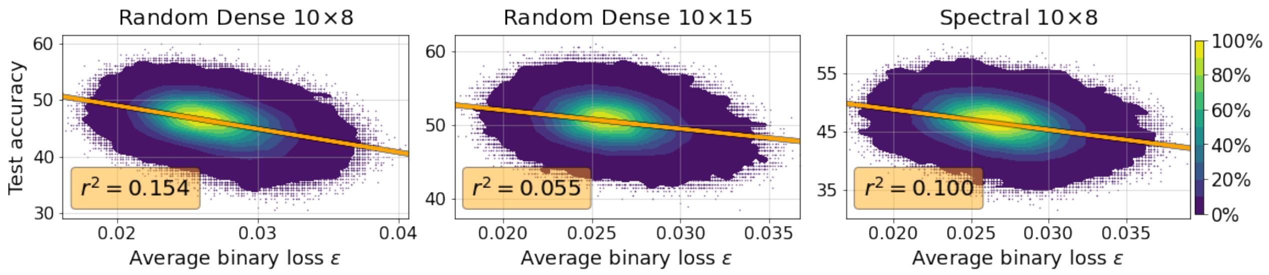


Figure 16: Test accuracy vs. Average binary loss in `yeast`.

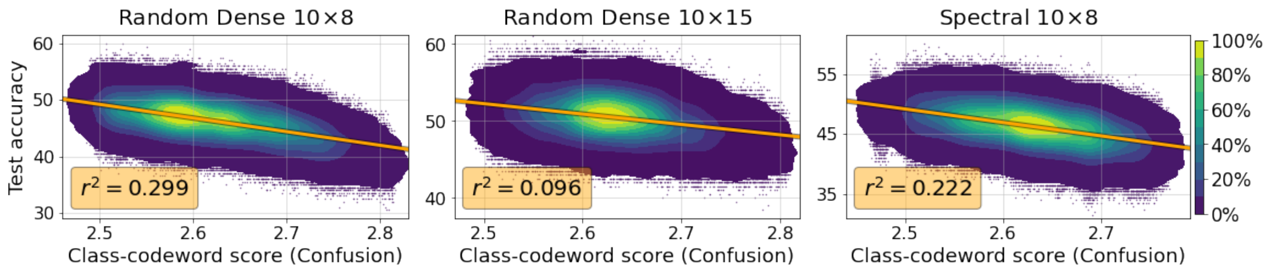


Figure 17: Test accuracy vs. Class-codeword scores in `yeast`, using the confusion matrix to construct $\mathbf{D}_{\text{cls}}$.

E Intermediate scale: CIFAR-100

In this section, we use CIFAR-100 (Krizhevsky et al., 2009) that have $K = 100$ classes to demonstrate that the class-codeword score is correlated with the test accuracy in a larger dataset than the ones we use in Section 4.1. Moreover, we show that as we discuss in Section 5, the class-codeword score (5) can also be used to efficiently *find* a similarity-preserving assignment in practice. So far in our other experiments in Section 4, we did not explicitly use the score to find good assignments. In the small-scale experiments in Section 4.1, we exhaustively computed the test accuracy of all $K!$ possible assignments to empirically *prove* the correlation to the class-codeword score. In the extreme experiments in Section 4.2, we employed a special structure of the WLTLS codebooks and the class taxonomy of the extreme datasets we used (but the class-codeword score was not *explicitly* minimized). Here on the other hand, given a codebook and a general class metric (not in a tree structure), we explicitly minimize the class-codeword score using a local search algorithm to obtain a similarity-preserving assignment.

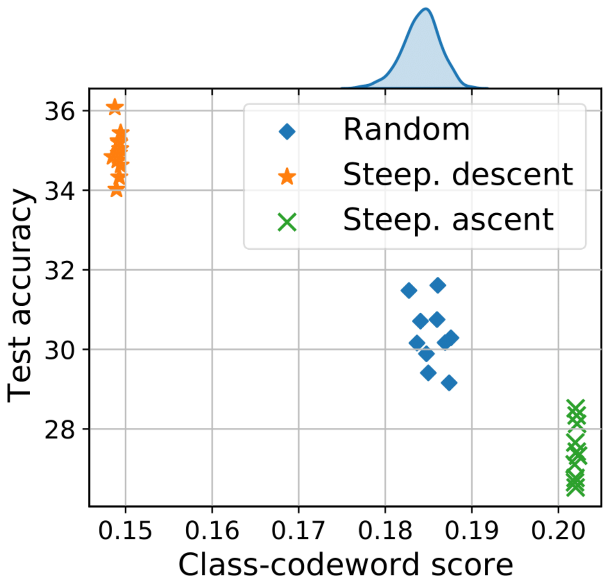


Figure 18: Results on a Dense 10×20 codebook for CIFAR-100. On top is the empirical marginal distribution of class-codeword scores of random assignments. The class-codeword score is clearly correlated with the test accuracy. The local search algorithm finds similarity-preserving assignments that significantly improve on random assignments. Further discussion below.

Codebook. For this experiment, we use a 100×20 random dense codebook. Like Allwein et al. (2000), we choose the random codebook by randomizing 10^4 dense codebooks and taking the codebook with the largest minimal Hamming distance ρ.

Feature extraction. We use a simple publicly available pretrained convolutional neural network[3] for feature extraction. This allows us to train, like before, simple linear predictors on top of the extracted features.

Class metric. As a metric between classes, we use the Euclidean distances between word embeddings of class names. Specifically, we use a publicly available fastText (Bojanowski et al., 2016) model, pretrained on Common Crawl and Wikipedia. These embeddings serve as a cheap heuristic of a semantic metric between classes, which can possibly approximate the visual class similarity. Class names' embeddings were previously used for computer vision tasks (e.g., for few-shot learning; Xing et al., 2019). We use this semantic metric since it can very easily be acquired in many real-world scenarios where classes have known names. Note that any other metric between classes should work here since this is already the third class-(dis)similarity measure we explore in the paper (together with confusion matrices and class means; see App. C.3).

[3] https://github.com/aaron-xichen/pytorch-playground/

Optimizing the class-codeword score (Steepest descent hill-"climbing" local search). Given a codebook and a dataset, finding a class assignment (out of $100! \approx 9.33 \cdot 10^{157}$ assignments) with a low class-codeword score is a hard task (see discussion in Section 5). However, we are able to find assignments with a low class-codeword score by performing a simple (discrete) steepest descent algorithm.[4] This allows us to quickly find many assignments that are more than 10 standard deviations (!) farther from the mean class-codeword score.

Comparing different assignments. We train the scheme on the following codeword-to-class assignments:

1. 10 random assignments;
2. 10 similarity-preserving assignments (i.e., having a low class-codeword score), found by using 10 random restarts of steepest descent;
3. 10 similarity-breaking assignments (i.e., having a high class-codeword score), found similarly using steepest *ascent*.

All 30 assignments are learned separately, and their test accuracy is plotted in the above Figure 18 against their class-codeword score.

Discussing Figure 18. The figure demonstrates how similarity-preserving assignments significantly improve the performance of a predefined codebook on intermediate scales of K as well. The distribution on top of the plot is the empirical marginal distribution of class-codeword scores of random assignments. Note that the assignments found by the local search algorithms could not have been found by simply sampling assignments.

We obviously cannot run an exhaustive search on the entire 100! possible assignments in order to test the correlation, but these 30 assignments agree with our empirical findings from Section 4.1.3 that a lower class-codeword score implies better multiclass performance.

[4] Start from a random assignment. Search all $\binom{K}{2} = 4,950$ assignments obtained by swapping the codewords of two classes only. Pick the assignment with the lowest class-codeword score and repeat until convergence.

F Supplementary material for the extreme classification (XMC) experiments in Section 4.2

F.1 Extended dataset descriptions

The dataset properties are brought in the table below.

Table 3: Extreme Benchmarks' Extended Properties.

				Split Details			WLTLS Arguments		
Dataset	Area	Classes	Features	Train	Val.	Test	Epochs	Early Stop.	Similarity
aloi	Vision	1K	637K	90K	10K	8K	8	Yes	Clustering
LSHTC1	Text	12K	1.2M	83.8K	5K	5K	5	Yes	Given
LSHTC2	Text	27K	575K	330K	15K	39.2K	3	Yes	Given
ODP	Text	104K	423K	867K	-	493K	5	No	Clustering

Now we elaborate on these benchmarks for the sake of completeness and reproducibility.

aloi.bin (Rocha and Goldenstein, 2013). Downloaded from the PD-Sparse (Yen et al., 2016) repository.[5] The dataset was created by applying Random Binning Features (Rahimi and Recht, 2007) on the images of the original `aloi` dataset. See Yen et al. (2016) for more details.

`LSHTC1` (Partalas et al., 2015). Also called `LSHTC2010` or `Dmoz2010`. Downloaded from the PD-Sparse repository.[5]

`LSHTC2` (Partalas et al., 2015). Also called `LSHTC2011` or `Dmoz2011`. Originally this is not a multi-class dataset but a multi-label dataset. However, only 11,121 out of 394,756 training samples have more than one label. We thus remove these samples and randomly split the remaining 384K samples into train, validation, and test sets. A similar process was used to create the more common XMC dataset `Dmoz` (used for example in Yen et al., 2016; Evron et al., 2018). However, the leaves of `LSHTC2` were merged to create `Dmoz`, and so it has only 12K labels instead of 27K like in the dataset we use.

`ODP` (Bennett and Nguyen, 2009). Downloaded from the Vowpal Wabbit repository.[6] Only 867K out of 1.08M training samples and 394K out of 493K test samples are non-empty. We remove the empty training samples but keep the empty test samples.

[5] https://github.com/a061105/ExtremeMulticlass
[6] https://github.com/VowpalWabbit/vowpal_wabbit/tree/master/demo/recall_tree

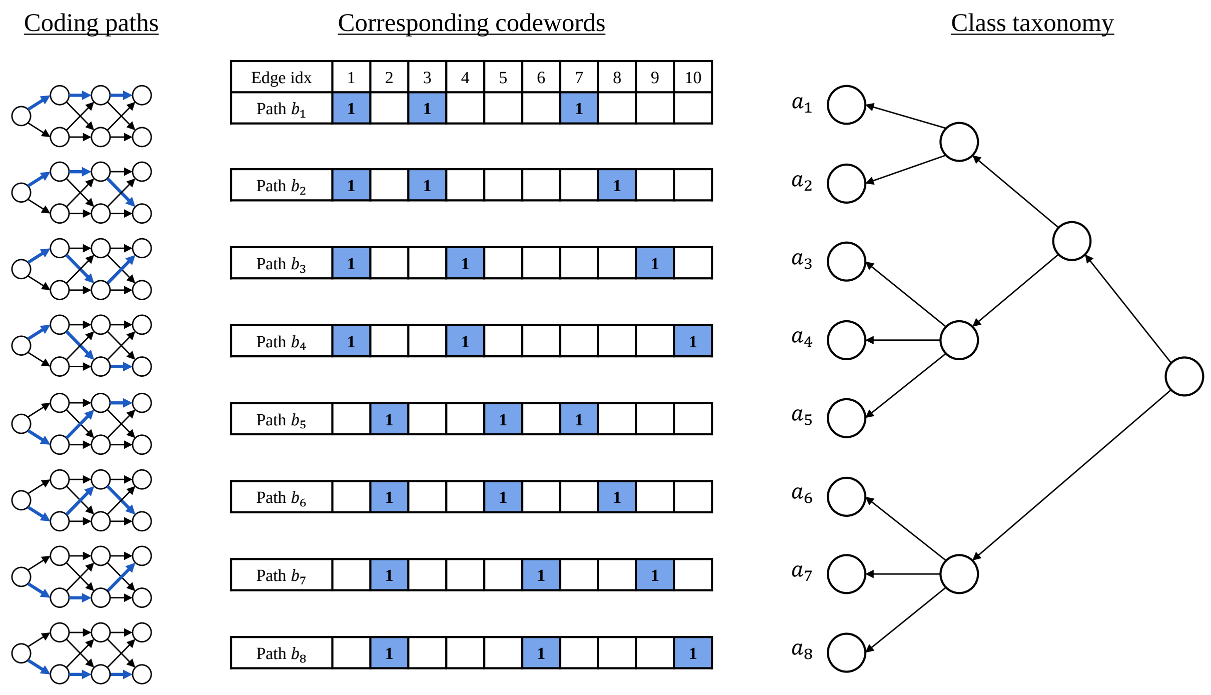
F.2 Additional algorithmic details for the naive assignment algorithm

We will now illustrate more thoroughly the assignment algorithm described in Section 4.2 and how it yields similarity-preserving assignments.

Understanding the coding graphs of `WLTLS`. On the left side of the figure, we illustrate the coding graph used in `WLTLS` (Evron et al., 2018) for $K = 8$ classes. The graph induces an error-correcting codebook as follows:

1. Each source-sink path in the graph corresponds to one class (notice that there are 8 such paths).
2. Each edge in the graph corresponds to one column (i.e., bit) in the codebook. That is, each edge induces one binary subproblem, separating the paths (classes) that use this edge from the ones that do not.

Overall, we see in the center of the figure that each path corresponds to a codeword whose +1 bits correspond to the edges used in the path.



Edge idx	1	2	3	4	5	6	7	8	9	10
Path b_1	1		1				1			
Path b_2	1		1					1		
Path b_3	1			1					1	
Path b_4	1			1						1
Path b_5		1			1		1			
Path b_6		1			1			1		
Path b_7		1				1			1	
Path b_8		1				1				1

Understanding the assignment algorithm. We explain the algorithm according to its original steps:

1. Given some class taxonomy (either known-in-advance or computed by hierarchical clustering for instance), the algorithm traverses the taxonomy to obtain an ordering $(a_1, \ldots, a_K)$ of leaves (classes). Notice that in most cases, classes a_i and a_{i+1} should be close on the taxonomy (i.e.,, these classes should be similar). See the right side of the figure.
2. Then, the algorithm recursively traverses the coding path (starting from the source; each node is visited *many* times) to obtain an ordering $(b_1, \ldots, b_K)$ of paths. Due to the recursion, in most cases, paths b_i and b_{i+1} should be similar in edges (i.e.,, their codewords should be close). See the left and center sides of the figure.
3. Finally, since classes a_i and a_{i+1} should be similar and so should codewords b_i and b_{i+1}, then by assigning class a_i to codeword b_i, we create a similarity-preserving assignment.

Like we explain in Section 4.2, despite its simplicity, the illustrated algorithm succeeds in creating similarity-preserving assignments. For instance, in `LSHTC1` with $\ell = 56$ edges, the algorithm found an assignment whose class-codeword score improves over the average score of random assignments by more than 900 standard deviations. Importantly, in all datasets, the multiclass accuracy significantly improves by using assignments found by the algorithm.

F.3 Tabular results

Here we summarize the extreme classification experiments of Section 4.2. For both datasets, similarity-preserving assignments consistently and significantly beat the random assignments. For all datasets, the accuracies are averaged over 5 runs. Two empirical standard deviations (of the runs, not their means) are reported as well.

F.3.1 `aloi.bin` results of Figure 3a

Assignment method	$\ell = 42$ ($b = 2$)	$\ell = 55$ ($b = 3$)	$\ell = 74$ ($b = 4$)	$\ell = 89$ ($b = 5$)	$\ell = 221$ ($b = 10$)
Random	84.86 ± 0.41	89.14 ± 0.39	91.77 ± 0.32	92.54 ± 0.17	94.89 ± 0.16
Similarity preserving	87.35 ± 0.35	90.40 ± 0.09	92.66 ± 0.17	92.73 ± 0.12	$\mathbf{95.26} \pm 0.09$

Table 4: Test accuracy (%) for different codebook widths ℓ (or WLTLS graph widths b) on `aloi.bin` ($K = 12,294$).
One-vs-all achieves 95.9% with $\ell = K = 1,000$ binary predictors.
Similarity-preserving assignments significantly improve the test performance compared to random assignments.

F.3.2 `LSHTC1` results of Figure 3b

Assignment method	$\ell = 56$ ($b = 2$)	$\ell = 79$ ($b = 3$)	$\ell = 138$ ($b = 5$)	$\ell = 338$ ($b = 10$)	$\ell = 879$ ($b = 20$)
Random	10.17 ± 0.71	13.33 ± 0.71	16.75 ± 0.69	20.50 ± 0.73	21.95 ± 0.55
Similarity preserving	13.19 ± 0.15	16.62 ± 0.25	19.22 ± 0.08	22.64 ± 0.16	$\mathbf{23.82} \pm 0.19$

Table 5: Test accuracy (%) for different codebook widths ℓ (or WLTLS graph widths b) on `LSHTC1` ($K = 12,294$).
One-vs-all achieves 23.3% with $\ell = K = 12,294$ binary predictors.
Similarity-preserving assignments beat OVA with only 879 binary predictors ($\times 14$ less).

F.3.3 `LSHTC2` results of Figure 3c

Assignment method	$\ell = 62$ ($b = 2$)	$\ell = 86$ ($b = 3$)	$\ell = 151$ ($b = 5$)	$\ell = 351$ ($b = 10$)	$\ell = 904$ ($b = 20$)
Random	11.34 ± 0.33	14.18 ± 0.11	17.39 ± 0.44	21.97 ± 0.46	26.06 ± 0.23
Similarity preserving	13.77 ± 0.12	17.24 ± 0.28	20.54 ± 0.45	24.84 ± 0.31	$\mathbf{28.31} \pm 0.09$

Table 6: Test accuracy (%) for different codebook widths ℓ (or WLTLS graph widths b) on `LSHTC2` ($K = 27,840$).
One-vs-all achieves 27.88% with $\ell = K = 27,840$ binary predictors.
Similarity-preserving assignments beat OVA with only 904 binary predictors ($\times 30.8$ less).

F.3.4 `ODP` results of Figure 3d

Assignment method	$\ell = 72$ ($b = 2$)	$\ell = 299$ ($b = 8$)	$\ell = 752$ ($b = 15$)
Random	2.71 ± 0.09	9.05 ± 0.03	11.71 ± 0.47
Similarity preserving	4.27 ± 0.02	10.45 ± 0.08	$\mathbf{12.69} \pm 0.04$

Table 7: Test accuracy (%) for different codebook widths ℓ (or WLTLS graph widths b) on `ODP` ($K = 104,136$).
Training a one-vs-all model for 104K classes on 423K features is too costly, hence we do not report its performance for this dataset.