Learning Compact Class Codes for Fast Inference in Large Multi Class Classification

M. Cissé, T. Artières, and Patrick Gallinari

Laboratoire d'Informatique de Paris 6 (LIP6), Université Pierre et Marie Curie, Paris, France
`firstname.lastname@lip6.fr`,
`http://www-connex.lip6.fr`

Abstract. We describe a new approach for classification with a very large number of classes where we assume some class similarity information is available, e.g. through a hierarchical organization. The proposed method learns a compact binary code using such an existing similarity information defined on classes. Binary classifiers are then trained using this code and decoding is performed using a simple nearest neighbor rule. This strategy, related to Error Correcting Output Codes methods, is shown to perform similarly or better than the standard and efficient one-vs-all approach, with much lower inference complexity.

1 Introduction

Classification problems with very large number of classes (VLC) now occur in many applications in the web, text, image or video domains. Current problems often deal with tens or hundreds of thousand of classes. For example, for patent classification the number of classes is around 60 000, for image annotation classes are keywords and their number is not limited, the number of classes in large class hierarchies like Dmoz is around 600 000 and still growing.

Scaling algorithms for VLC is a recent research direction compared to scaling wrt the sample size or data dimensionality and this is still a challenging problem [1], [2] [3], [4], [5]. Its specificity lies in the complexity of inference. The inference linear complexity in the number of classes of standard one vs rest approaches is prohibitive for VLC and only sub-linear inference methods are acceptable for practical purpose. Of course, training should also remain feasible. Besides pure scaling problems, classes in VLC problems may evolve, e.g. some classes may become rarely observed. Designing classifiers that do not require full retraining for new classes is also important in many cases.

We focus here on the design of algorithms for dealing with these different issues. In our approach the classification problem is casted into a cost-sensitive framework where a class distance or class similarity information is supposed available. Cost sensitivity reflects an existing latent structure between the classes and these relations will be exploited as complementary knowledge to improve classification performance and to reduce the inference and training complexities.

This information could be provided by existing resources which in our case is a class-taxonomy, but the extension to other class similarity measures is straightforward.

Within this framework, the approach we develop relies on first learning binary class codes using the similarity information between classes, a class will then be represented as a l-dimensional binary code with values in $\{-1, +1\}$, and second in training l binary classifiers, each will predict one bit of the class code. The *dichotomizer* for the j^{th} bit of the code will be trained to distinguish between the samples of all classes whose j^{th} bit is 1 and those whose j^{th} bit is -1. A test example will then be categorized according to a simple nearest neighbor rule between the code computed for this example and learned codes. This method is inspired by Error Correcting Output Codes (ECOC) [6] and Class embeddings [7]. With this strategy, the complexity of inference will become linear in the length of the code instead of the number of classes for computing the output code of an input sample and logarithmic in the number of classes to compute the closest class code. Consequently we aim at designing compact class codes. Besides fast decoding, these codes should be discriminant enough to reach a performance equivalent to or higher than standard classification methods, at a reduced inference complexity.

Our main contribution is an efficient procedure for learning compact binary class codes of length l such that $l << k$ where k stands for the number of classes. The inference requires then computing the output of l classifiers while for the one vs rest (OVR) approach inference requires computing the output of k classifiers. The value of l may be set so as to achieve a compromise between complexity and acuracy. We show experimentally that the value of l required for reaching OVR performance, scales sub-linearly with the number of classes k and that increasing the complexity of the method (i.e. l) allows outperforming OVR. We provide an experimental comparison, with respect to performance and runtimes, of our method with baselines, including OVR, on datasets up to 10 000 classes built from the 2010 Large Scale Hierarchical Text Classification challenge datasets [8].

Finally, beyond its raw performance, we investigate the particular ability of our method for zero-shot learning, i.e. recognizing samples from new classes without any training sample. We show that providing the similarity information for new classes allows recognizing samples from theses classes even in the case when no training samples are available.

The paper is structured as follows. Section 2 reviews related works, section 3 presents our approach for learning compact class codes, and finally section 4 reports experimental results.

2 Related Works

Classification in a large number of classes has received an increasing attention in the last few years. The challenge of designing sub-linear inference complexity algorithms has guided the researchers into two main directions: hierarchical approaches and class nearest neighbor search methods.

Hierarchical Approaches exploiting a tree structured relation among classes are straightforward solutions for reducing the inference complexity from $O(k)$ to $O(\log k)$. Besides, many problems can be formulated as hierarchical classification, and most of the datasets available to the research community are organized hierarchically. Different methods have been proposed and some start from an existing hierarchy while others learn the class hierarchy. The filter tree and the conditional probability tree [9] for example are consistent reduction of multi-class problems to binary that learn a tree of classifiers. Trees offer a natural and efficient solution to the inference complexity problem, on the other hand, it is widely recognized (e.g. [3], [2]), that classifier cascades greatly suffer from the propagation of errors from parent to children. This is why some authors [10], [3] propose to globally train the classifiers in the tree, instead of using local classifiers, and report improved performance at the cost of larger training complexity.

The One-vs-Rest [11] approach is the most popular flat multi-class classifier. Surprisingly, it remains one of the most efficient approaches in terms of accuracy, for VLC [3]. Although the inference complexity is $O(k)$, it is readily parallellizable which might be another way for solving the complexity issue. The one-vs-rest classifier is then a strong contender for large scale classification and often the best classifier for VLC in terms of accuracy.

Taxonomy [7] and Label Embedding [3] are other flat approaches that propose to jointly learn a projection of the data and the classes (or the taxonomy) in a low dimensional latent space where each data will be close to its class representation. The inference procedure is based on a class nearest neighbor search, so that its complexity is potentially $O(\log k)$. This, and the competitive performance reported make these methods appealing for large multi-class problems though their performance is often below that of OVR method (e.g. [3]).

Error Correcting Output Coding [6] has not been used up to now for VLC. Since our method produces ECOCs, we introduce its principle here and will compare our strategy with standard ECOC in the experimental section. ECOC is a general framework for handling multi-class problems and consists in representing each class with a codeword. These codewords are arranged into a coding matrix $M(k \times l)$ where l is the code length and k is the number of classes. ECOC uses a binary coding $M \in \{-1, 1\}^{k \times l}$, each column of the coding matrix defines a partition of the target space and learning consists in training l dichotomizers to predict a codeword for each new instance. Prediction, also called decoding, is done by assigning a new sample to the class having the closest codeword according to a distance measure. The key issue here is designing a coding matrix with good error correcting properties. It is usually required that both rows and columns are well separated. Row separation ensures that a large number of binary classifiers have to make a wrong decision before the decoding process misclassifies a test sample. Column separation ensures that the binary dichotomizers (there is one dichotomizer per column) are uncorrelated. The most popular way for initialiazing M is to choose each m_{ij} to be 1 or -1 with probability 1/2 [12], it is called dense random ECOC. For small number of classes, ECOC might outperform the standard one-vs-rest scheme [6] [12] and the inference complexity is $O(\log k)$.

3 Our Approach: Learned Distributed Representation (LDR)

3.1 Principle

As demonstrated in recent publications one of the best performing method for VLC today is the OVR method [3]. Yet this strategy has inference complexity that scales linearly with the number of classes. Alternatively hierarchical methods allow fast inference but fail to reach the accuracy of OVR, due to error propagation in the tree. We aim here at building a method that allows, both fast inference and high accuracy. To reach this goal we propose a method called Learned Distributed Representation (LDR) that first learns binary low dimensional class codes, then uses binary classifiers to learn each bit of the codes, as in ECOC.

A key issue is to take into account the available relationships between classes (e.g. a hierarchical or a graph organization of classes). We propose to compute low dimensional binary class codes that reflect these relationships. In order to do that we first represent a class as a vector of similarities between the class and all other classes, $\mathbf{s}_i = [s(C_i, C_1), ..., s(C_i, C_k)]$ (see section 4 for an example). Different similarity measures may be used. It may be computed from a hierarchy of classes or from a similarity between samples of the two classes. Then, we learn short class codes that reflect these relationships between classes, by transforming these high k-dimensional representations of classes ($\mathbf{s}_i$) into lower l-dimensional codes ($\mathbf{h}_i$) via a dimension reduction algorithm. This step is explained in details in section 3.2. Once low dimensional (say l-dimensional, with $l << k$) binary class representations are learned, we train l binary classifiers, one for every bit. The binary classifier for the j^{th} bit is a dichotomizer that is learned to separate samples of all classes whose class code has the j^{th} bit set to 1 from the samples of all classes whose class code has the j^{th} bit set to -1. All these binary classifiers are then learned with all training samples from all classes.

Finally at test time, when one wants to decide the class of an input sample x, we use the l classifiers on x to compute a l-length binary word $\mathbf{m} = (m_1, ..., m_l)$ which is compared to the k class codes $\{\mathbf{h}_i, i = 1..k\}$ to find the nearest neighbor.

3.2 Learning Compact Binary Class-Codes

We propose to learn compact class codes with autoencoders which have been widely used for feature extraction and dimensionality reduction [13], [14]. Among many existing dimension reduction methods the advantage of autoencoders lies in the flexibility of the optimization criterion that allows us including additional terms related to class codes separation. An autoencoder is trained by minimizing a squared reconstruction error between the input (here a class representation $\mathbf{s}_i$)

and its reconstruction at the output of the autoencoder, $\widehat{\mathbf{s}}_i$. It may be viewed as an encoder (input $\rightarrow$ hidden layer) followed by a decoder (hidden $\rightarrow$ output layer). Usually it is required that encoding and decoding weights are tied [14], both for linear and non linear encoders, so that if $\mathcal{W}$ is the coding matrix, $\mathcal{W}^T$ is the decoding matrix. We used this strategy here. Training an autoencoder writes (omitting bias terms):

$$\underset{W}{\operatorname{argmin}} \sum_{i=1}^{k} ||\mathbf{s}_i - \mathcal{W}^T \times f(\mathcal{W} \times \mathbf{s}_i)||^2 \tag{1}$$

where $||.||$ is the euclidean distance. The activation function in hidden units f may be a linear function, then the projection learned by the autoencoder is similar to the one learned by a principal component analysis. One can expect to learn more interesting features by using nonlinearities on hidden units, using sigmoid or hyperbolic tangent activation functions (in our implementation, we use hyperbolic tangent activation function hidden units). To perform dimensionality reduction one uses a narrow hidden layer which forces to learn non trivial regularities from the inputs, hence interesting and compact codes on the hidden layer. The vector of activation of hidden units is the learned encoding function. Here the new class code for class C_i is then $\mathbf{h}_i = f(\mathcal{W} \times \mathbf{s}_i)$.

Ideally, new class codes should satisfy two properties. First, similar classes (according to the cost-sensitive information and/or to similar examples) should have close codes $\mathbf{h}_i$. Second, class codes for any pair of classes should be significantly different to ensure accurate classification at the end. The first property is naturally satisfied since an autoencoder actually learns hidden codes that preserve distances in the original space. Next, to ensure minimal separation between class codes we propose to look for a solution of the following constrained problem:

$$\begin{aligned} &\underset{\mathcal{W}}{\operatorname{argmin}} \sum_{i=1}^{k} ||\mathbf{s}_i - \mathcal{W}^T \times f(\mathcal{W} \times \mathbf{s}_i)||^2 \\ &\quad \text{s.t. } \forall (i,j), i \neq j : ||f(\mathcal{W} \times \mathbf{s}_i) - f(\mathcal{W} \times \mathbf{s}_j)|| \geq b \end{aligned} \tag{2}$$

The constraints are inspired from margin based learning and yield to maximize the distance between any pair of class codes up to a given threshold b. We solve this optimization problem by stochastic gradient descent using the unconstrained regularized form:

$$\begin{aligned} \underset{\mathcal{W}}{\operatorname{argmin}}\ &\alpha \sum_{i=1}^{k} ||\mathbf{s}_i - \mathcal{W}^T \times f(\mathcal{W} \times \mathbf{s}_i)||^2 \\ &+ \beta \sum_{i,j=1}^{k} \max(0, b - ||f(\mathcal{W} \times \mathbf{s}_i) - f(\mathcal{W} \times \mathbf{s}_j)||) \\ &+ \frac{\lambda}{2} ||\mathcal{W}||^2 \end{aligned} \tag{3}$$

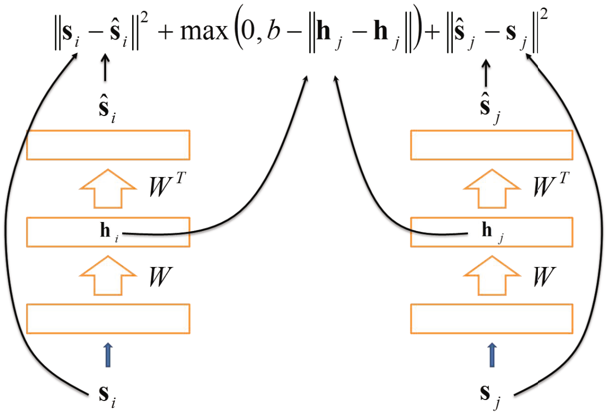
$\|\mathbf{s}_i - \hat{\mathbf{s}}_i\|^2 + \max(0, b - \|\mathbf{h}_j - \mathbf{h}_j\|) + \|\hat{\mathbf{s}}_j - \mathbf{s}_j\|^2$
$\hat{\mathbf{s}}_i$
W^T
$\mathbf{h}_i$
W
$\mathbf{s}_i$
$\hat{\mathbf{s}}_j$
W^T
$\mathbf{h}_j$
W
$\mathbf{s}_j$

Fig. 1. Learning the autoencoder from pairs of input samples (here α and β are considered equal to 1). See Algorithm 1 for details.

where α and β weight the respective importance of the reconstruction error term and of the margin terms, and $||\mathcal{W}||^2$ is a regularization term. Note that α, β, and b (which tunes the margin between two class codes) are set by cross validation.

We learn the autoencoder using stochastic gradient descent by iteratively picking two training samples i and j at random and making a gradient step. Figure 1 illustrates the training process which recalls somehow Siamese architectures used in the past for vision tasks [15]. At the end, in order to get binary class codes, we threshold the learned real valued class codes. This means that the j^{th} component of all class codes $\mathbf{h}_i$ are set to $\mathbf{h}_i(j) = -1$ if $\mathbf{h}_i(j) < \theta_j$, and $\mathbf{h}_i(j) = +1$ otherwise. The threshold value θ_j is chosen so that the prior probability of the j^{th} bit of a class code be $+1$ is equal to 0.5, and this is done by setting θ_j to the median of $\{\mathbf{h}_i(j)|i = 1...k\}$. Although this cut-off it is not learned to optimize classification accuracy, it should be noted that it is defined according to the usual property in ECOC (firing with probability 0.5). Also since similar classes should have close class codes, it is expected that the obtained two class classification problem (i.e. for the j^{th} bit of class codes, separating samples of all classes with $\mathbf{h}_i(j) = +1$ from the samples of all classes with $\mathbf{h}_i(j) = -1$) should be easier to solve than any random two class problem as those defined in traditional ECOC. We will come back to this point in the next section. Algorithm 1 describes the whole algorithm.

3.3 Relations to ECOC

Because each element in the class codes has probability 1/2 of being either $+1$ or -1, our method bares some similarities with the standard dense random ECOC. However, there are two fundamental differences.

The first difference is that by construction, our learned distributed representation is intended to have a reduced tree induced loss compared to randomly generated methods because the autoencoder projects classes that are close in the hierarchy in the same area of the latent space. The second difference, which is somehow related to the first one, is that the binary classification problems induced by the learned class codes should be easier than in random ECOC. Indeed, since similar classes should have close class codes, it is likely that for similar classes most bits are equal. This means that a particular dichotomizer is trained with samples for class +1 and for class -1 that are more homogeneous than if the partitioning of classes was random, as in traditional ECOCs. At the end, if dichotomizers reach higher accuracy, the overall accuracy of the multiclass classifier should also be higher.

Algorithm 1. Learning Compact Binary Class Codes

```
1: Input: ${\mathbf{s}_i \in \mathbb{R}^k | i = 1, ...k}$, $l$, $\epsilon$
2: Output: ${\mathbf{h}_i \in \mathbb{R}^l | i = 1, ...k}$

3: Learn the weights $\mathcal{W}$ of an autoencoder (with $k$ input neurons, $l$ hidden neurons,
   and $k$ output neurons) on ${\mathbf{s}_i \in \mathbb{R}^k | i = 1, ...k}$ to minimize cost in Eq. (3)
4: repeat
5:    Pick randomly two samples $(\mathbf{s}_i, \mathbf{s}_j)$
6:    Make a gradient step : $\mathcal{W} = \mathcal{W} - \epsilon \partial L_{\mathcal{W}}(\mathbf{s}_i, \mathbf{s}_j)/\partial \mathcal{W}$
      with: $L_{\mathcal{W}}(\mathbf{s}_i, \mathbf{s}_j) = \frac{1}{2}\sum_{k\in\{i,j\}} \alpha||\mathbf{s}_k - \mathcal{W}^T \times f(\mathcal{W} \times \mathbf{s}_k)||^2 + \lambda||\mathcal{W}||^2 + \beta \max(0, b -$
      $||f(\mathcal{W} \times \mathbf{s}_i) - f(\mathcal{W} \times \mathbf{s}_j)||)$
7: until convergence criterion is met
8: Compute the learned class codes $\forall i \in [1, k], \mathbf{h}_i = f(\mathcal{W} \times \mathbf{s}_i)$
9: for all $j = 1...l$ do
10:    Compute the median $\theta_j$ of the $j^{th}$ component of $\mathbf{h}_i$'s, $\{\mathbf{h}_i(j) | i = 1, ..., k\}$
11:    Threshold the $j^t h$ component of $\mathbf{h}_i$'s at $\theta_j$ so that $\forall i \in [1, k], \mathbf{h}_i(j) =$
       $\begin{cases} 1 \text{ if } \mathbf{h}_i(j) \leq \theta \\ -1 \text{ otherwise} \end{cases}$
12: end for
13: return Compact binary class codes ${\mathbf{h}_i \in \mathbb{R}^l | i = 1, ...k}$
```

An ECOC coding scheme closer to our method is the discriminative ECOC (DECOC) which learns a discriminative coding matrix by hierarchically partitioning the classes according to a discriminative criteria [16]. The hierarchy is built so as to maximize the mutual information between the data in each partition and the corresponding labels. Our method differs from this in that we are seeking codewords having a sub-linear dependency on the number of classes k while the DECOC method creates codewords of length $k - 1$.

3.4 Training and Inference Complexity

We focus here on complexity issues with respect to the number of classes k, the number of training samples N, the dimension of samples d, and the length of the learned class codes l. Let us denote by $C_T(N)$ the complexity of training one binary classifier with N training samples, and by C_I the complexity of inference for a binary classifier. All complexities in the following will be expressed as a function of C_T and C_I.

We start with our method. Training consists in learning the class codes of length l, then in learning l classifiers. Learning class codes is done by gradient descent whose complexity depends on the number of iterations. Yet since class codes are binarized at the end, one can expect that the method will not be very sensitive to accurate convergence of the autoencoder and one can reasonably assume a fixed and limited number of iterations $\mathcal{I}$ so that learning the autoencoder requires $O(\mathcal{I} \times k^2 \times l)$ (I iterations with k samples every iteration whose forward and backward pass costs roughly $O(k \times l)$). Next, learning the l binary classifiers requires $O(l \times C_T(N))$. At the end training complexity is in $O(\mathcal{I} \times k^2 \times l + l \times C_T(N))$. Inference consists in finding the class code which is most similar (wrt. Hamming distance) to the output code computed for this input sample. Computing the output code requires using the l classifiers, hence $O(l \times C_I)$. Next, using fast nearest neighbor search methods such as ball trees or kd-trees for finding the closest class code may be done (in practice) in $O(\log k)$ comparisons [17], where each comparison costs $O(l)$. Overall, the inference complexity is then $O(l \times (\log k + C_I))$.

We compare these costs to those of the OVR method which is the most accurate technique for large scale classification [3] (see Table 1). Training in OVR method requires $O(k \times C_T(N))$ since one uses k classifiers that are all trained with all training samples, while inference requires $O(k \times C_I)$.

It clearly appears from this discussion that OVR does not extend easily to VLC due to its inference complexity that scales linearly with the number of classes. Compared to these baselines, our method exhibits interesting features. As we will argue from experimental results, it may outperform OVR for $l << k$ and the minimal length l for such a behavior seems to scale strongly sublinearly with k. Furthermore although the training complexity includes a term in $O(k^2)$, it must be clear that in experimental settings such as the ones we investigate in this paper (large number of samples and high dimensionality), the overall training complexity in $O\left(l\mathcal{I}k^2 + lC_T(N)\right)$ is dominated by the second term $O(lC_T(N))$.

Table 1. Comparison of training and inference complexity for our method and for standard methods, OVR and ECOC, as a function of the number of classes k, the dimension of the data d, the size of the class codes l, the learning complexity of a binary classifier with N training samples $C_T(N)$, the inference complexity of a binary classifier C_I, and the number of training iterations I of the autoencoder (LDR method).

	Training	**Inference**
OVR	$O(kC_T(N))$	$O(kC_I)$
ECOC(l)	$O(lC_T(N))$	$O\left(lC_I + l\log k)\right)$
LDR(l)	$O\left(l\mathcal{I}k^2 + lC_T(N)\right)$	$O\left(lC_I + l\log k\right)$

4 Experiments

We performed experiments on three large scale multi-class single label datasets. The proposed method (LDR) is compared to two coding methods, spectral embedding (SPE) and traditional error correcting output coding (ECOC), and to a standard OVR baseline. We first present the datasets, then we explain our experimental setup and finally we present results and analysis.

4.1 Datasets

We used datasets with respectively 1000, 5000 and 10000 classes. Each dataset was created by randomly selecting the corresponding classes from a large scale dataset released for the first PASCAL large scale hierarchical text classification challenge (Kosmopoulos et al., 2010). This dataset was extracted from the open Mozilla directory DMOZ (www.dmoz.org). The classes are organized in a tree hierarchy, classes being at the leaves of the hierarchy and internal nodes being not instantiated classes. Hierarchies are of depth 5 (Kosmopoulos et al., 2010).

The documents were provided as word counts, and then transformed into normalized TF/IDF feature vectors. Considering that for large multi-class text classification every new class is likely to bring specific new words, we did not performed any feature selection although all datasets have very high dimensional feature spaces.

Statistics of the datasets are detailed in Table 2. Each dataset is split into training, validation and testing sets (see Table 2).

We exploited a similarity measure between classes i and j, which is defined as a function of the distance $d_{i,j}$ between the two classes in the hierarchy measured by the length of the shortest path in the tree between the two classes: $\mathbf{s}_i(j) = s(C_i, C_j) = \exp(-d_{i,j}^2/2\sigma^2)$. The tree path distance between two classes is also used in the tree loss used as a classification measure in section 4.3. We systematically used $\sigma = 1$ in our experiments.

Table 2. Statistics of the dataset used in the experiments

Statistics	10^3 classes	$5 * 10^3$ classes	10^4 classes
Nb. training docs	8119	36926	76417
Nb. validation docs	3005	13855	28443
Nb. testing docs	3006	13771	28387
Nb. features	347 255	-	-

4.2 Experimental Setup

Three classifiers were used as baselines: OVR, random ECOC and a Spectral Embedding technique.

Besides ECOC classifiers, we also compared our method to a spectral embedding technique (SPE) which can be used for learning class codes from a similarity matrix and is an alternative to our auto-associator method. Spectral embedding is widely used as a preprocessing step before applying k-means in clustering applications. It has also been used recently for hashing and we exploit a similar idea here. In [18] the authors propose to embed the data for fast retrieval by binarizing the components of the eigenvectors of the similarity matrix Laplacian. This process aims at mapping similar examples in the same regions of a target space. The training complexity of the method is $O(k^3 + lC_T(N))$, which is much larger than LDR or ECOC, and is due to the high complexity if the eigen-decomposition. This method is similar in spirit to LDR and ECOC and is a natural candidate for comparison. The classes here play the same role as data do in spectral hashing.

We chose logistic regression as a base classifier (dichotomizers) for all methods, but any other binary classifier could be used as well. The binary classifiers were trained with a regularization parameter selected from $\lambda \in \{0.001, 0.0005, ..., 10^{-6}\}$ using the validation set.

To train random ECOC classifiers, for a given code length l and a number of class k, we generated several $k \times l$ matrices and discarded those having equal or complementary rows. We then used the coding matrices with best error correcting property (the top 25 matrices for 10^3 classes and the top 10 for $5 * 10^3$ and 10^4 classes) to train an ECOC classifier. Then we kept the model that reached the best performance on the validation set for evaluation on the test set.

We compare the methods using accuracy and tree induced loss which is defined as the average of the length of the shortest path in the hierarchy between the correct class and the predicted class. The tree induced loss measures the ability of the classifier to take into account the hierarchical nature of the classification problem, and the class proximity according to this metric. A low tree loss means that confusions are made between neighboring classes, while a high tree loss signifies that confusions occur among distant classes.

4.3 Comparison of the Methods

We investigate here the behavior of the different methods on the three datasets and explore how the performance evolves with respect to the class code length. Comparisons with all methods are performed on the 10^3 and $5 * 10^3$ classes corpora, while on the larger 10^4 classes dataset, only OVR vs LDR were tested. Figure 2 reports accuracies on the first two datasets for code length in {200, 300, 400, 500, 600}. First it can be seen that LDR outperforms systematically the two other coding methods (SPE and ECOC) whatever the dataset, and whatever the class code length. Second, the performance of the three coding methods (LDR, SPE and ECOC) increases, with some fluctuation, with the code length. A higher code is needed when the number of classes increases. This behavior is intuitive. Finally one can see that LDR reaches and even exceeds the performance of OVR on these two datasets, while ECOC and SPE stay under the performance of OVR, even when increasing the code length l.

Table 3 compares the different methods using their best accuracy score[1], and the corresponding tree induced loss on the same two datasets. It can be seen that the best performance of the different methods are quite close, LDR being systematically higher and providing a clear speedup wrt OVR. For example, for 1 000 classes, with a code length of 200 LDR achieves an accuracy of 67.49% while OVR's accuracy is 66.50%. In this case, the number of classifiers used by the OVR method is 5 times that of LDR.

We come back to our previous observation that LDR is consistently better than random error correcting output coding (ECOC) (Figure 2), which holds whatever the code length. Our main explanation of this phenomenon is that the binary problems are probably easier to solve with LDR. It has been observed since the early use of ECOCs [6] that the dichotomies induced by the codes where more difficult to solve than the initial OVR dichotomies. Here, neighbor classes in the tree, are forced to have similar codes. The data for these classes are often closer one to the other than that of distant classes, so that similar inputs will most often be required to be classified similarly. On the opposite, classical ECOCs where codes are designed at random do not share this property. To investigate this, we compared the mean accuracy of the binary classifiers induced by our method to the mean accuracy of classifiers in a random ECOC scheme. The mean accuracy remains between 72% and 75% for LDR while it is constant at about 69% for ECOC which confirms the hypothesis that learned dichotomizers induce easier problems. Also we think that the learning criteria of the autoencoder helps creating better class codes than those produced by the spectral embedding method.

At last we compare LDR and OVR on classification tasks with up to 10 000 classes. Figure 3 shows the performance of LDR vs OVR for the three datasets ($10^3, 5 * 10^3$ and 10^4 classes) for a code length of size 500. LDR outperforms OVR whatever the number of classes. Speedup are more and more important as the number of classes increases. For 10^4 classes LDR achieves an accuracy of 36.81% (with a code length of 500) while the OVR's performance is 35.20%. This performance is achieved while using 20 times less classifiers than the number of classes. This corresponds to a speedup of 46 wrt OVR (measured by runtimes). Such a speedup is not only due to the smaller number of classifiers used by LDR, but also to fast bitcounts routines that exploit the binary representation of codes for nearest neighbour search.

4.4 Zero-Shot Learning

A few approaches have been proposed in the literature to answer the *zero-shot learning* problem [19], [20], i.e. designing a classifier that is able to discriminate between classes for which we do not have instances in the training set. One particular approach proposes the use of a rich semantic encoding of the classes [20]. Our approach is close to this idea since the codes of classes

[1] For each method, one uses the parameterization, including the value of l, leading to the best score.

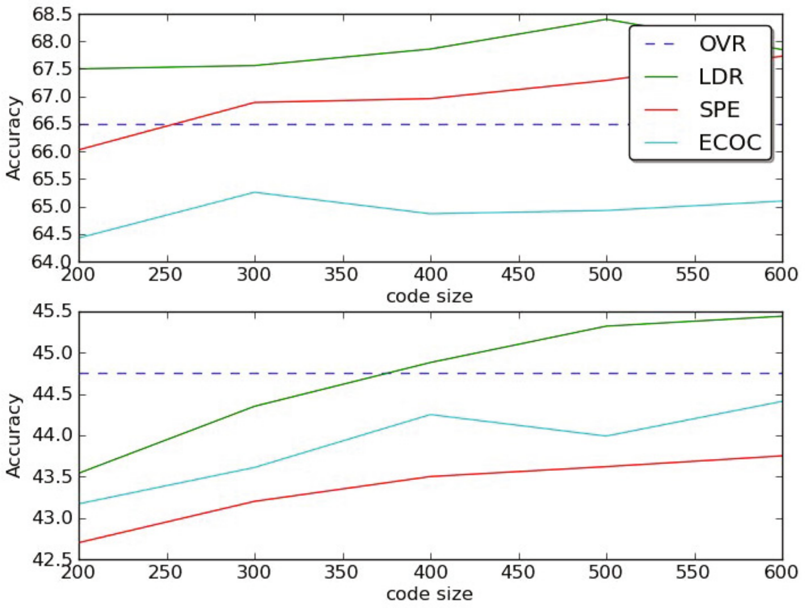

Fig. 2. Accuracy of our method (LDR), random ECOC (ECOC), Spectral Embedding (SPE), and OVR as a function of code length on datasets with 1 000 classes (top) and with 5 000 classes

Table 3. Comparative results of OVR, Random ECOC, Spectral Embedding, and LDR, on datasets with 1000 and 5000 classes with respect to accuracy, tree induced loss, and inference runtime. The runtimes are given as speed-up factors compared to OVR (×2 means twice as fast as OVR). Reported results are the best ones obtained on the datasets whatever the class code length. For LDR, we also provide the performance reached for a minimal l yielding performance at least equal to that of OVR, denoted as LDR (first), to to stress the speed-up.

Classifiers	1000 classes			5000 classes		
	Accuracy	T.I.L	Speed	Accuracy	T.I.L	Speed
One-vs-rest	66.50%	2.63	×1	44.76%	3.98	×1
Random ECOC	65.10%	2.74	×2	44.41%	4.12	×12
SPE	67.73%	2.51	×2	43.75%	4.30	×12
LDR (first)	67.49%	2.54	×5	44.88%	3.98	×17
LDR(best)	**68.40%**	**2.46**	×2	**45.44%**	**3.93**	×12

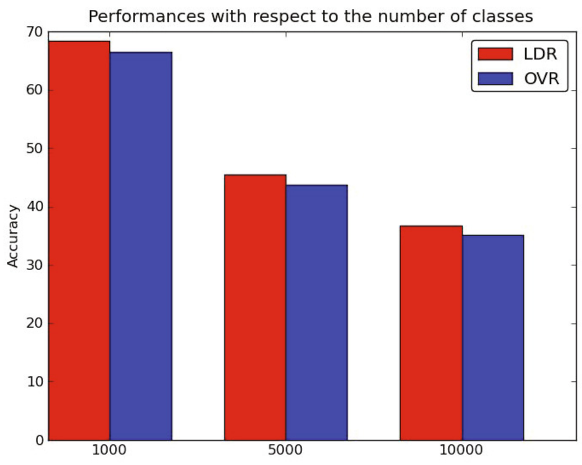

Fig. 3. Accuracy of our method (LDR) and OVR on datasets with 1 000, 5 000 and 10 000 classes. Whatever the dataset LDR exploits class codes of length $l = 500$.

Table 4. Average accuracy (and standard deviation) of LDR ($l = 200$) for zero-shot learning tasks. Results are averaged over 10 runs with removal of different random sets of classes.

# classes removed	10	20	30	40	50
Accuracy (std)	25.64(12.20)	24.45(6.34)	16.76(4.24)	14.31(3.18)	12.76(2.48)

(computed by the autoencoder) are vectors that encode some semantic information on classes.

To explore empirically how our model is able to achieve zero-shot learning, we performed the following experiment on the 1000 classes dataset. We learned the class codes on the 1000 class representations (similarity vectors) computed from the hierarchy, s_i. Then we selected randomly a number of classes (10 to 50) and removed all training samples of these classes from the training set. The dichotomizers were then trained with this reduced training set. At test time, following the approach in [19], we use the learned classifier to discriminate between the classes whose training samples were not present in the training set. Results are given in Table 4 for a class code length equal to 200. One can see that the accuracy achieved by LDR on classes that have not been learned is significantly greater than a random guess although it is naturally lower than the accuracy obtained on classes that were actually represented in the training set as reported in previous section.

Note also that one could go one step further than the zero-shot paradigm and try to recognize samples from a new class which was even not used for learning the class codes, provided one gets its similarity with all classes in the training stage. This would fit with many large multi-class problems where the set of classes is not closed (for instance new classes appear periodically in the DMOZ repository). Preliminary results show a similar performance as above provided the number of new classes remains small. This is a perspective of our work.

5 Conclusion

We have presented a new approach for dealing with hierarchical classification in a large number of classes. It combines the accuracy of flat methods and the fast inference of hierarchical methods. It relies on building distributed compact binary class codes that preserve class similarities. The main features of the method lies in its inference complexity that scales sub-linearly with the number of classes while outperforming the standard OVR and Error Correcting Output Codes techniques on problems up to 10 000 classes. Interestingly our approach also allows, to some extent, considering the addition of new classes in the hierarchy without providing training samples, an instance of the zero-shot learning problem.

References

1. Weinberger, K., Chapelle, O.: Large margin taxonomy embedding for document categorization. In: Koller, D., Schuurmans, D., Bengio, Y., Bottou, L. (eds.) Advances in Neural Information Processing Systems, vol. 21, pp. 1737–1744 (2009)
2. Bennett, P.N., Nguyen, N.: Refined experts: improving classification in large taxonomies. In: SIGIR, pp. 11–18 (2009)
3. Bengio, S., Weston, J., Grangier, D.: Label embedding trees for large multi class tasks. In: Advances in Neural information Processing (2010)
4. Xiao, L., Zhou, D., Wu, M.: Hierarchical classification via orthogonal transfer. In: Getoor, L., Scheffer, T. (eds.) Proceedings of the 28th International Conference on Machine Learning (ICML 2011), pp. 801–808. ACM, New York (2011)
5. Deng, J., Satheesh, S., Berg, A.C., Li, F.F.: Fast and balanced: Efficient label tree learning for large scale object recognition. In: NIPS, pp. 567–575 (2011)
6. Dietterich, T.G., Bakiri, G.: Solving multiclass learning problems via error-correcting output codes. Journal of Artificial Intelligence Research 2, 263–286 (1995)
7. Weinberger, K., Chapelle, O.: Large taxonomy embedding with an application to document categorization. In: Advances in Neural Information Processing (2008)
8. Kosmopoulos, A., Gaussier, E., Paliouras, G., Aseervatham, S.: The ecir 2010 large scale hierarchical classification workshop. SIGIR Forum 44(1), 23–32 (2010)
9. Beygelzimer, A., Langford, J., Lifshits, Y., Sorkin, G., Strehl, A.: Conditional probability tree estimation analysis and algorithms. In: Proceedings of the Twenty-Fifth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI 2009), pp. 51–58. AUAI Press, Corvallis (2009)
10. Cai, L., Hofmann, T.: Hierarchical document categorization with support vector machines. In: Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management, pp. 78–87 (2004)

11. Rifkin, R., Klautau, A.: In defense of one-vs-all classification. J. Mach. Learn. Res. 5, 101–141 (2004)
12. Allwein, E.L., Schapire, R.E., Singer, Y., Kaelbling, P.: Reducing multiclass to binary: A unifying approach for margin classifiers. Journal of Machine Learning Research 1, 113–141 (2000)
13. Gallinari, P., LeCun, Y., Thiria, S., Fogelma-soulie, F.: Mémoires associatives distribuées: une comparaison (distributed associative memories: a comparison). In: Proceedings of COGNITIVA 1987, Paris, La Villette, Cesta-Afcet (May 1987)
14. Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P.A.: Extracting and composing robust features with denoising autoencoders. In: Proceedings of the 25th International Conference on Machine learning, ICML 2008, pp. 1096–1103. ACM, New York (2008)
15. Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., Shah, R.: Signature verification using a siamese time delay neural network. In: NIPS, pp. 737–744 (1993)
16. Pujol, O., Escalera, S., Radeva, P.: An incremental node embedding technique for error correcting output codes. Pattern Recogn. 41(2), 713–725 (2008)
17. Moore, A.: Efficient memory-based learning for robot control (October 1990)
18. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: NIPS, pp. 1753–1760 (2008)
19. Larochelle, H., Erhan, D., Bengio, Y.: Zero-data learning of new tasks. In: AAAI, pp. 646–651 (2008)
20. Palatucci, M., Pomerleau, D., Hinton, G.E., Mitchell, T.M.: Zero-shot learning with semantic output codes. In: NIPS, pp. 1410–1418 (2009)