Learning efficient error correcting output codes for large hierarchical multi-class problems

Moustapha Cissé, Thierry Artières, Patrick Gallinari

Learning efficient error correcting output codes for large hierarchical multi-class problems

M. Cissé, T. Artières, P. Gallinari
Laboratoire d'Informatique de Paris 6
Université Pierre et Marie Curie
Paris, France
Email: firsname.name@lip6.fr

No Institute Given

Abstract. We describe a new approach for dealing with hierarchical classification with a large number of classes. We build on Error Correcting Output Codes and propose two algorithms that learn compact, binary, low dimensional class codes from a similarity information between classes. This allows building classification algorithms that performs similarly or better than the standard and performing one-vs-all approach, with much lower inference complexity.

Large scale classification; Error Correcting Output Codes; Spectral Embedding

1 Introduction

Classification problems with very large number of classes (VLC) do appear in many applications in the web, text, image or video domains. Whereas research on large scale learning algorithms has mainly focused on large sample size or large data dimensionality problems, research on very large class numbers is more recent and is still a challenging problem. As for other scaling challenges, training should be feasible, i.e. should be performed in a reasonable time with available resources. The specificity of VLC lies on the inference complexity problem. Classical approaches are at best linear in the number of classes which is prohibitive when dealing with tenth or hundreds of thousands classes. Besides complexity, VLC problems are often plagued with severe class imbalance.

We propose here an algorithm that scales sub-linearly with the number of classes. The classification problem is cast 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 additional knowledge to improve classification performance and to reduce the inference and training complexities. This information could be provided by existing resources, e.g. a class-taxonomy or inferred from the data itself. Within this framework, the approach we develop relies on learning Error Correcting Codes (ECOC)[8] using a similarity information between classes.

Each class will be represented as a binary code and a binary classifier will be learned for each code bit. A test example will then be categorized according to a simple nearest neighbor rule between the code computed for this example and learned codes. If the code length is order of magnitudes less than the number of classes, the complexity of inference will be reduced accordingly and will be proportional to this code length. The challenge here is to learn compact codes for fast decoding that provide enough discriminating power to reach performance equivalent or superior to classical classification methods with a reduced complexity. Efficient codes will have to satisfy two constraints. Class codes corresponding to similar classes according to the cost-sensitive information and to similar examples should be close so that the bit classifiers may be learned accurately. On the other hand, class codes should be sufficiently different so as to ensure low global error rate, which is the usual requirement for ECOC classification. To reach this goal, we propose to learn class codes that preserve the similarity between classes while constraining a minimal separation of the learned codes.

Our main contributions are two alternative algorithms for learning such compact class codes, with inference complexity scaling with the class code length c. One relies on a spectral embedding of the classes in a latent space, and the other relies on learning an auto-encoder neural network on class similarity representations. We show that this allows building a k class classifier with short length codes ($c << k$) that performs as well as or outperforms the standard one vs all (OVA) classifier, whose inference complexity is in $O(k)$. Finally we provide an experimental investigation of the behavior of our methods on two datasets that were used for the 2010 Large Scale Hierarchical Text Classification challenge [11].

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

2 Related works

Classification in a large number of classes has received much attention in the last few years. Usually the classes are organized into a hierarchy. Among the many approaches explored for this problem, one may distinguish between two types of approaches.

On the one hand *big bang* approaches disregard the eventual hierarchy information and use a single classifier on the whole dataset. One popular representative is the One vs All (OVA) method where one uses a classifier (e.g. a SVM) for distinguishing between one particular class and all other classes. Nearest neighbors have also been used in its standard setting (using all training samples as the model) or using one prototype per class [20]. All these methods may perform well (e.g. [16]) but they have inference complexities that scale at least linearly with the number of classes, which makes them impractical for real-life use as suggested by the 2010 Large Scale Hierarchical Text Classification challenge (LSHTC) final report [11]. Note however that nearest neighbor method

may be made more competitive by learning binary representation of training samples (or of class representatives) as proposed in [1].

On the other hand *top-down* approaches build hierarchical classifiers that match the given class hierarchy, with a classifier at each node of the hierarchy. Such methods may reach inference complexity that scale logarithmically with the number of classes, at the price of a well known error propagation phenomenon, when errors at the root nodes are propagated in the tree and yield misclassification. Most works make use of a simplified hierarchy. For instance a number of studies have proposed to use a simplified hierarchy, keeping only the first levels [17]. [18] proposed a lazy learning scheme where one first uses a flattened version of the hierarchy to design a first classifier whose aim is to select a restricted number of candidate classes. In a second step a classifier is build from this set of classes to classify the test sample. Some other methods use hierarchical classifiers such as hierarchical SVM on the full hierarchy or on a flattened version of the hierarchy [6], [2]. Hierarchical classifiers have come with various optimization criterion [26] [6] [2].

Another line of work consists in designing hierarchical classifiers without relying on an existing hierarchy. [2] proposes to discover a hierarchy from scratch by identifying confusable classes by first learning one to one classifiers. Beygelzimer et al. proposed the Filter tree [5] and the conditional probability tree [5], both are tree-based reductions from multi-class to binary classification, where binary classifiers are learned at each node of the tree.

Finally some authors have exploited simultaneous data and class embedding in a latent space [26],[2]. The idea consists in projecting the data and the labels into a joint low dimensional space, enabling more accurate classification and robustness to sample dimensionality. Such a projection may make use of the class hierarchy information, enabling similar classes to be projected in the same area [26]. In this latter work inference is then performed by nearest neighbor search, in[2] it is done through classification in a tree. Another indirect use of the hierarchy information, slightly different from the above, consists in defining class prototypes (in nearest neighbors systems) according to the tf-idf measure using locally computed statistics inferred from the hierarchy [18].

3 Learning Compact ECOC

We mix here ideas from class embedding and ECOC learning in order to design compact, binary, and discriminative class codes from a similarity information between all classes. This allows us, building efficient and fast classifiers. Many methods have been proposed to determine ECOC, either data independent such as the one-vs-rest reduction [23] or random binary codes as provided by Hadamard matrices [3], or data dependent [22] [10]. Yet the class codes have a length at least k and are not good candidates in our case. We present below two methods for learning low dimensional class codes. One is based on spectral embedding [15], [21], the other builds on autoencoders [12], [25]. Both algorithms take as input a similarity matrix S between pairs of classes, we assume from now

on that class i is represented as the i^{th} column of S, s_i which is a k-dimensional vector, with $s_{i,j}$ the similarity between classes i and j. Our method bares some similarity with ECOC but it differs in that we want class codes to be as short as possible.

Our two methods do not output binary codes but real-valued ones. In order to get binary class codes, we threshold the output of every bit classifier so that it outputs either -1 or 1. This cutt-off is heuristically designed and it is not learned to optimize classification acccuracy, this could be a perspective of our work.

3.1 Spectral class-hashing (SCH)

Finding binary codes is very similar to learning a hash function for fast nearest neighbor search. As stated before, we seek binary c-dimensional class codes, c_i, minimizing the average Hamming distance between similar classes. To maximize the compactness of the codes bits should be independent and have probability 0.5 of being on. Measuring code proximity by the euclidean distance and replacing bit independency by bit non correlation, one can formulate the problem as:

$$\text{minimize: } \sum_{i,j=1}^{k} s_{i,j}||c_i - c_j||^2 \tag{1}$$

$$\text{subject to } c_i \in \{-1,1\}^c \text{ and } \sum_{i=1}^{k} c_i = \vec{0} \text{ and } \sum_{i=1}^{k} c_i c_i^T = I \tag{2}$$

where $s_{i,j}$ is the similarity between class i and class j, and c_i is the learned code for class i. This formalization requires a trade-off between the two objectives discussed in the introduction. The minimized term ensures that similar classes have similar class-codes, while the third constraint requires bits to be uncorrelated, which limits bits redundancy. The second constraint requires each bit to be on half of the time. This optimization problem is equivalent to a balanced graph-partition problem and is known to be NP-hard [15] Relaxing the first constraint and looking for real-valued vector class codes we get the following problem in matrix-form:

$$\text{minimize: trace } C(\Delta - S)C^T \tag{3}$$

$$\text{subject to } C1_{k,k} = 0 \text{ and } CC^T = I \tag{4}$$

where C is the $c \times k$ matrix whose columns are class codes and Δ is the diagonal degree matrix of the similarity matrix, with $\Delta_{i,i} = \sum_j s_{i,j}$. This latter problem is equivalent to a spectral embedding problem whose solution are the top c eigenvectors of the laplacian matrix $L = \Delta - S$. It has been shown that the

normalized laplacian $L_{sym} = \Delta^{-1/2} L \Delta^{-1/2}$ may give better results in terms of clustering quality [21], we used this variant in our implementation since it leads to better results. A related approach has been adopted in [27] where data is projected onto a hamming space for fast document retrieval, this approach is called spectral hashing. The complete procedure is described in algorithm 1, where the last step is the binarization step of the class codes.

Algorithm 1: SCH

1: **Input:** Similarity matrix $\mathbf{S}$, code-size c
2: **Output:** Error correcting output code matrix $\mathbf{C}$

3: Compute the normalized graph laplacian: $L_{sym} = \Delta^{-1/2} L \Delta^{-1/2}$
4: Compute its first c eigenvectors $u_1, u_2, \ldots, u_c$
5: **for all** j such that $1 \leq j \leq c$ **do**
6: Compute the median m_j of u_j's components
7: Threshold each u_j's component at m_j to obtain binary values
8: **end for**
9: **return** Error correcting output code matrix $\mathbf{C}$ whose lines are the thresholded eigenvectors

3.2 Structured auto-hashing (SAH)

Autoencoders have been widely used recently either for building deep neural networks for classification or feature extraction and for dimensionality reduction [12], [13]. An autoencoder is trained to reconstruct the input vector at its output. It is trained to minimize a squared reconstruction error between the input (here a class representation s_i) and its reconstruction at the output of the autoencoder. 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 [25], both for linear and non linear encoders, so that if w is the coding matrix, w^T i the decoding matrix. We also used this constraint here. The vectors of activation of hidden units is the learned encoding function $c_i = f(w \times s_i)$, it is parameterized by a weight matrix w whose transpose is used for reconstruction (decoding) and f is the activation function of hidden units. Using a narrow hidden layer forces to learn non trivial regularities from the inputs, hence interesting and compact codes on the hidden layer. Using an autoencoder to learn low dimensional representations of class-codes writes (omitting bias terms):

$$\text{minimize: } \sum_{i=1}^{k} ||s_i - w^T \times f(w \times s_i)||^2 \tag{5}$$

f may be a linear function, then the autoencoder is proved to learn a representation very similar to the one leaned by a principal component analysis but one can expect to learn more interesting features by using nonlinearities on hidden units, such as the sigmoid function or the hyperbolic tangent.

Again, we want to satisfy two objectives, similar classes should have close codes with respect to hamming distance, but their codes should be distinct and significanty different at the same time. The first objective is naturally reached with autoencoder since it learns codes that preserve distance in the original space. To enforce separation between class codes we propose to look for a solution of the following problem:

$$\begin{aligned} &\text{minimize } \frac{1}{k} \sum_{i=1}^{k} ||s_i - w^T \times f(w \times s_i)||^2 \\ &\text{subject to: } \forall (i,j), i \neq j : ||f(w \times s_i) - f(w \times s_j)|| \geq b \end{aligned} \tag{6}$$

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 propose to solve this optimization problem by stochastic gradient descent using the unconstrained form:

$$\begin{aligned} \text{minimize } &\frac{1}{k} \sum_{i} ||s_i - w^T \times f(w \times s_i)||^2 + \lambda ||W||^2 \\ &+ \frac{\alpha}{k \times (k-1)} \sum_{i,j} \max(0, b - ||f(w \times s_i) - f(w \times s_j)||) \end{aligned} \tag{7}$$

where α and λ weight the respective importance of the regularization and the margin terms (they are tuned by cross-validation). The autoencoder is learned using stochastic gradient descent by iteratively picking two training samples i and j at random and making a gradient step. At the end, the learned distributed representations c_i are, just as in previous section, thresholded to get binary codes. We describe the whole procedure in algorithm 2.

3.3 Training and inference complexity

We focus here on compleixty issues with respect to the number of classes k. Training consists in learning the classcodes of length c, then in learning c classifiers. SCH requires $O(k^2)$ to compute classcodes while SAH is a gradient-based learning whose complexity is more difficult to estimate (it depends on the number

Algorithm 2: SAH

Input: Similarity matrix $\mathbf{S}$, code-size c
Output: Error correcting output code matrix $\mathbf{C}$

Learn the autoencoder to minimize cost in Eq. (7)
Compute the new representation of classes with the learned codes $\forall i, c_i = f(w \times s_i)$
for all j such that $1 \leq j \leq c$ **do**
    Compute the median m_j of c_i's j^th component
    Threshold each c_i's j^th component at m_j to obtain binary values
end for
return Output the error correcting output code matrix $\mathbf{C}$ whose columns are thersholded c_i's

of iterations). Learning the classifiers is in $O(c)$. At the end training complexity is at most $O(k^2 + c)$, but in practice it is closer to $O(k + c)$ for SAH.

Inferring the class of a test sample consists in finding the class code which is most similar (wrt Hamming distance) to the output code computed for this input sample, it is a series of -1 and 1 output by the c binary classifiers. Using fast nearest neighbor search methods such as ball trees or kd-trees this may be done (in practice) in $O(\log k)$ [19]. Overall, the inference complexity is in $O(c+\log k)$.

4 Experiments

We apply the methods described above to a large scale hierarchical text classification (lshtc) problem where the classes are leaves of a tree-structured hierarchy and the similarity matrix is computed from the distances between the classes in the hierarchy according to $s_{i,j} = \exp(-\frac{d_{i,j}^2}{2\sigma^2})$ where $d_{i,j}$ is computed as the number of edges required to go from class i to class j in the tree. This is related to the tree loss which is considered for evaluating the correctness of a classification rule in a tree. Note that we chose $\sigma = 1$ in all our experiments.

4.1 Datasets

We experimented our approach on single-label hierarchical classification tasks. We used two nested datasets (We distinguish between the *small* dataset and the *large* one hereafter) that have been delivered and exploited for the first track of the pascal LSHTC (Large Scale Hierarchical Text Classification) challenge in 2010 [11]. Both datasets are subsets of the DMOZ repository [9], their features are detailed in Table 1.

The classes of the two datasets we use in the experiments are organized in a tree hierarchy. This means that every node in the hierarchy has only one parent. The classes are the leaves of the hierarchy and internal nodes are not instantiated classes. Both hierarchies are of depth 5 [11].

We used same settings as in the challenge for the small dataset. For the large dataset, we kept the original training set and considered the validation set as the test set since the ground truth for the original test set is not available.

For both datasets, samples are content vectors obtained by directly indexing the web pages. We preprocessed samples to get normalized tf-idf feature vectors. Despite the large dimensionality of samples for both datasets, we did not perform feature selection in this study although it is a perspective of the work.

	Large dataset	Small dataset
training samples	93805	6323
test samples	34905	1858
nb of classes	12294	1139

Table 1. Datasets used in the experiments

4.2 Comparison with state of the art methods

Before reporting results we provide some implementation details. All the binary classifiers used in the experiments are binary linear svms, whatever the method used, ours or the OVA reference method. Note that any other classification method could have been used here. We used default values for the bias and the C term ($C = 1$) for learning all these binary classifiers as it is widely admitted that it gives good performances when using normalized tf-idf features. During training and testing, we parallelized all the binary classifiers.

We first investigate the behavior of our methods, SCH and SAH, on the *small* dataset to explore how the performance behave with respect to the class code length. Table 2 reports results of both methods, SCH and SAH, for code length ranging from 64 bits to 1024 bits. We report both accuracy and tree induced loss, which is the average distance in the hierarchy between the correct and the recognized class. These results show that, as it has been noticed before, accuracy and tree induced loss are closely related. There is a systematic superiority of the SAH method over the SCH method. Finally it shows that performance increases significantly with the code length.

Let us compare our methods on the *small* and on the *large* datasets, with a moderate code length of 512 bits, to the standard OVA method (Table 3). SAH systematically outperform OVA on the two datasets whatever the criterion is, while SCH performs slightly less than OVA on the *large* dataset only. Let us recall that the OVA classifier consists in more than 1 100 classifiers on the *small* training set and more than 12 000 classifiers for the *large* dataset.

SAH performs equivalently to OVA when using a code length of about 200 on the *small* dataset and with a code length of about 250 on the *large* dataset. This means that our method may perform as well as OVA with a complexity reduced by factors of 5 and 50 on the *small* and *large* datasets respectively. This

Model	Accuracy (%)	Tree induced loss
64 bits		
SAH	**41.03**	4.47
SCH	40.42	**4.36**
128 bits		
SAH	**43.82**	**4.17**
SCH	42.97	4.32
256 bits		
SAH	**45.32**	**3.99**
SCH	44.98	4.13
512 bits		
SAH	**46.70**	**3.89**
SCH	46.14	4.09
1024 bits		
SAH	**46.79**	**3.83**
SCH	46.21	3.97

Table 2. performance comparison of SCH and SAH for different code lengths

shows that the code length required to peform equivalently to the OVA scales sublinearly with the number of categories.

Model	Accuracy (%)	Tree induced loss
Small dataset		
One-vs-rest	44.72	4.13
SAH	**46.70**	**3.89**
SCH	46.14	3.97
Large dataset		
One-vs-rest	35.68	4.56
SAH	**36.59**	**4.05**
SCH	35.12	4.17

Table 3. Performance comparison on the *small* and *large* datasets. SAH and SCH models use 512 bits length codes.

Finally, Table 4 reports some comparative results with representative methods proposed by participants of the PASCAL LSHTC 2010 challenge. We included representative methods belonging to the two families discussed in section 2, *big bang* approaches and *top-down* approaches [11]. We report results from [17] that use a flattened version of the hierarchy combined with lazy learning, from a prototype based approach (nearest neighbor rule) which exploits the hierarchy to build centroids, using term frequencies and category-dependent IDF at each level of the hierarchy [18]. Finally we report results of [16] whose method belongs to the *big bang* family. One sees that some methods may outperform our

approaches on these datasets but, again, their complexity forbids us using them on larger classification problems.

Model	Accuracy (%)	T.I. Loss	inference complexity
big-bang [16]	46.32	3.28	$O(k)$
flatenning [17]	44.3	3.36	Lazy
prototype [18]	40.42	4.36	$O(k)$
SAH (512)	37.84	3.98	$O(c + \log k)$

Table 4. Performance and complexity comparison of few competitor methods of the 2010 LSHTC challenge on the *large* dataset: k is the number of classes, Lazy means that the method trains a new classifier for every test sample. Note that the SAH result is different from the one reported in Table 3 since the result here has been gained on the test set to be fully comparable with other results in this table.

4.3 Robustness to imbalanced data

Today designing classifiers with imbalanced training sets is a major issue in the machine learning field [14]. This is a usual problem in large scale classification tasks. For instance 20% of the classes represent about 50% of the samples in the *small* dataset, while it represents 80% of the samples in the *large* dataset. Table 5 gives the percentage of classes whose number of training samples ranges from 1 to 3, from 4 to 6 and is greater than 6 (which is the average number of samples per-class in the *small* dataset).

Standard strategies for dealing with imbalanced problems rely on modifying the distribution of the data by either under-sampling the most represented classes or oversampling the less represented ones [7]. Yet there is no clear and established solution today.

Besides, we expect our method to be naturally more robust to imbalanced problems than one-versus-rest classifier since every dichotomizer is learned with all samples from all classes. To investigate the robustness of our approach to imbalanced data we computed detailed statistics to understand the behavior of the methods with respect to the class representativity. We provide in Table 6 experimental results gained with the SAH method (with 512 bits code length) and with OVA on the *small* dataset. It reports performance measures (accuracy and tree induced loss) for the three categories of classes distinguished in Table 5. As may be seen the behavior of our classifier is different from that of the OVA classifier. Our classifier performs significantly better for all under-represented classes while it performs slightly less on well-represented classes. In addition we studied the classification statistics of our classifier and of one-versus-rest on the whole test set and found that our classifier classifies test samples in 511 classes out of the 1139 classes while the OVA classifier classifies the test samples in 459 classes. This confirms the results in Table 6 showing a better robustness of our method with respect to the imbalance problem.

	sample representativity (%)	class-representativity (%)
$[1, 3]$	28.0	60.50
$[4, 6]$	18.88	21.94
> 6	53.12	17.56

Table 5. Percentage of classes (and corresponding percentage of training samples) with very few, few, and more than average number of training samples, on the small dataset.

	Accuracy SAH (%)	Accuracy OVA (%)
$[1, 3]$	**27.99**	25.84
$[4, 6]$	**40.43**	35.76
> 6	65.33	**65.85**

Table 6. Comparative accuracy of SAH and OVA on the categories of classes in Table 5, on the small dataset.

5 Conclusion

We have proposed to deal with hierarchical classification in a large number of classes by building compact binary error correcting output codes that preserve class similarity as given by their distance in a hierarchy. We provided two algorithms that show good performance on two datasets from the Pascal 2010 LSHTC challenge, and whose inference complexity scales sublinearly with the number of classes, up to 50 times faster than one vs all classifier on a 12000 class problem to reach similar performance. One perspective is a more extensive analysis of how inference complexity (propotional to the length of class-codes) increases with respect to the number of classes.

References

1. A. Andoni and P. Indyk *Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions* Communications of the ACM, vol. 51, no. 1, 2008, pp. 117-122.
2. S. Bengio, J. Weston, D. Grangier *Label Embedding Trees for Large Multi-Class Tasks*, In Advances in Neural Information processing Systems, 2010.
3. A. Beygelzimer, J. Langford and B. Zadrozny *Machine Learning TechniquesReductionsBetween Prediction Quality Metrics*
4. A. Beygelzimer, J. Langford, Y. Lifshits, G. Lorkin, A. Strehl *Conditional Probability Tree Estimation Analysis and Algorithms*, UAI, 2009.
5. A. Beygelzimer, J. Langford and P. Ravikumar *Multiclass Classification with Filter Trees*, In International Conference on Algorithmic Learning Theory (ALT), 2009.
6. L. Cai and T. Hofmann *Hierarchical Document Categorization with Support Vector Machines*, in proceedings of International Conference on Information and Knowledge Management, 2004.

7. Nitesh Chawla, Nathalie Japkowicz, and Aleksander Kolcz. *Special issue on learning from imbalanced dataset.* SIGKDD explorations, 2004.
8. T. G. Dietterich and G. Bakiri, *Solving multi-class learning problems via error correcting output codes*, International conference on artificial neural network 1999.
9. *www.dmoz.org*
10. S. Escalera, Oriol Pujol, and Petia Radeva. *Boosted landmarks of contextual descriptors and Forest-ECOC: A novel framework to detect and classify objects in clutter scenes.*, Pattern RecognitionLetters, 28(13) pages 17591768, 2007.
11. A. Kosmopoulos, E. Gaussier, G. Paliouras, S. Aseervatham *The ECIR 2010 Large Scale HierarchicalClassfication Workshop*, ECIR, 2010.
12. Gallinari, P., LeCun, Y., Thiria, S. and F. Fogelman-Soulie *Memoires associatives distribuees.* Proceedings of COGNITIVA 1987.
13. G. E. Hinton and R. R. Salakhutdinov *Reducing the Dimensionality of Data with Neural Networks* in Science 28 : Vol. 313 no. 5786 pp. 504-507.July 2006.
14. N. Japkowicz. *Class imbalances: are we focusing on the right issue?* In International Conference on Machine Learning Workshop on Learning from Imbalanced Data Sets II, 2003.
15. U. Von Luxburg *A tutorial on spectral clustering* in Statistics and Computing, 17 (4), 2007.
16. O. Madani and J. Huang *Large-scale many-class prediction via flat techniques.* InLarge-Scale Hierarchical Classication Workshop of ECIR, 2010.
17. H. Malik. *Improving hierarchical svms by hierarchy flattening and lazy classification.*In Large-Scale Hierarchical Classication Workshop of ECIR, 2010.
18. Y. Miao and X. Qiu. *Hierarchical centroid-based classifier for large scale text classica-tion.* LSHTC Website, 2010
19. A. W. Moore, *Efficient Memory-based Learning for Robot Control*, Ph. D. Thesis, Technical Report, No. 209, Computer Laboratory, University of Cambridge.
20. C. C. Coterillo *An Adaptation of Soucy and Mineau weighting for Large Scale Text Classification.* InLarge-Scale Hierarchical Classication Workshop of ECIR, 2010
21. A.Y. Ng, M.I. Jordan, and Y.Weiss. *On spectral clustering, analysis and an algorithm.* In Advances in Neural Information Processing 14, 2001.
22. O. Pujol, S. Escalera, and P. Radeva. *An incremental node embedding technique for error-correctingoutput codes*, Pattern Recognition, 4 pages 713725, 2008.
23. R. Rifkin and A Klautau , *In defense of one-vs-all classification*, Journal of machine learning research, pages 101-141, 2004.
24. R. Salakhutdinov, G. Hinton *Semantic hashing.* ACM SIGIR, 2007.
25. Vincent, H. Larochelle Y. Bengio and P.A. Manzagol *Extracting and Composing Robust Features with Denoising Autoencoders* ICML, pp 1096 - 1103 2008.
26. K. Weinberger and O. Chapelle. *Large margin taxonomy embedding for document categorization.* InNIPS, pages 17371744, 2009.
27. Y. Weiss, A. Torralba and R. Fergus *Spectral hashing.* in Advances in Neural Information Processing Systems, 2008.