Experiences with the LVQ Algorithm in Multilabel Text Categorization

A. Montejo-Ráez, M. T. Martín-Valdivia and L. A. Ureña-López

Jaén University, Jaén E-23071 Spain - Departamento de Informática

Abstract. Text Categorization is an important information processing task. This paper presents a neural approach to a text classifier based on the Learning Vector Quantization (LVQ) algorithm. We focus on multilabel multiclass text categorization. Experiments were carried out using the High Energy Physics (HEP) text collection. The HEP collection is an highly unbalanced collection. The results obtained are very promising and show that our neural approach based on the LVQ algorithm behaves robustly over different parameters.

1 Introduction

Text Categorization (TC) is an important task for many Natural Language Processing (NLP) applications. Given a set of documents and a set of categories, the goal of a categorization system is to assign to each document a set (possibly empty) of categories that the document belongs to. The simplest case includes only one class and the categorization problem is a decision problem or binary categorization problem (given a document, the goal is to determinate if the document is 'relevant' or 'not relevant').

On the other hand, the single-label categorization problem consists on assigning exactly one category to each document while the multi-label categorization problem assigns from 0 to m categories to the same document. For a deeper and exhaustive discussion on different text categorization problem see [1].

This work studies the multi-label categorization problem using the Learning Vector Quantization (LVQ) algorithm. The LVQ algorithm is a competitive neural learning algorithm that allows assign a category (from a set of categories) to a document (single-label problem). In order to accomplish the multi-label categorization problem we have used the LVQ algorithm as a binary classifier integrated in the TECAT Toolkit [2]. TECAT stands for TExt CATegorization. It is a tool for creating multi-label automatic text classifiers. With TECAT you can experiment with different collections and classifiers in order to build a multi-labeled automatic text classifier and implements the Adaptive Selection of Base Classifiers (ASBC) as approach to the problem.

The paper is organized as follows. First, the HEP collection is introduced. Then, we describe briefly the TECAT Toolkit as a tool to solve the multi-label automatic text classification problem. Then, we describe the LVQ algorithm and the integration on the TECAT in order to accomplish our experiments. After this, we show our evaluation environment and results obtained. Finally, we discuss our conclusion and future work.

Montejo-Ráez A., T. Martín-Valdivia M. and A. Ureña-López L. (2007).
Experiences with the LVQ Algorithm in Multilabel Text Categorization.
In *Proceedings of the 4th International Workshop on Natural Language Processing and Cognitive Science*, pages 213-221
DOI: 10.5220/0002419302130221

2 The HEP Collection

The HEP corpus[1] is a collection of papers related to *High Energy Physics*, and manually indexed with DESY labels. These documents have been compiled by Montejo-Ráez and Jens Vigen from the CERN[2] Document Server[3] and have motivated intensive study of text categorization systems in recent years [3], [4], [5].

Table 1. The ten most frequent main key words in the *hep-ex* partition.

No. docs.	Keyword
1898 (67%)	electron positron
1739 (62%)	experimental results
1478 (52%)	magnetic detector
1190 (42%)	quark
1113 (39%)	talk
715 (25%)	Z0
676 (24%)	anti-p p
551 (19%)	neutrino
463 (16%)	W
458 (16%)	jet

In our experiments we have used the meta-data from the *hep-ex* partition of the HEP collection, composed by 2,839 abstracts related to experimental High-Energy Physics that are indexed with 1,093 main keywords, with an average number of classes per document slightly above 11. This partition is highly unbalanced: only 84 classes are represented by more than 100 samples and only five classes by more than 1,000. The uneven use is particularly noticeable for the ten most frequent key words: In Table 1 the left column shows the number of positive samples of a key word and the percentage over the total of samples in the collection.

2.1 Evaluation Measures

The effectiveness of a classifier can be evaluated with several known measures [1]. The classical "Precision" and "Recall" for Information Retrieval are adapted to the case of Automatic Text Categorization. To that end, we must first determine the basis the system will be evaluated in: categories or documents. Traditionally contingency table for each category should be generated (Table 2), and then the precision and recall for each category are calculated following equations 1 and 2. But in a multi-label scenario also an equivalent contingency table can be computed, where *i* in this case represents a document instead of a class. For this evaluation to be useful, the average number of classes per document should be relevant. This is the case for the *hep-ex* partition, where an average number of 15 classes are assigned to a single document. Also, this last

[1] The collection is freely available for academic purposes from:
`http://sinai.ujaen.es/wiki/index.php/HepCorpus`
[2] CERN is the European Laboratory for Particle Physics, located in Geneva (Switzerland)
[3] `http://cds.cern.ch`

base of evaluation serves better for the purpose of really test the effectiveness of a label system in real environments, where an user expects a list of classes to a given item. This difference is only for macro-averaged values, because micro-averaged measurements remain the same for both evaluation strategies.

Table 2. Contingency Table for *i* Category.

	YES is correct	NO is correct
YES is assigned	a_i	b_i
NO is assigned	c_i	d_i

$$P_i = \frac{a_i}{a_i + b_i} \tag{1}$$

$$R_i = \frac{a_i}{a_i + c_i} \tag{2}$$

On the other hand, the precision and recall can be combined using the F_1 measure:

$$F_1(R,P) = \frac{2PR}{P+R} \tag{3}$$

As we have pointed out, in order to measure globally the average performance of a classifier, three measures can be used: micro-averaged precision P_μ, macro-averaged precision in a document basis $P_{macro-d}$ and macro-averaged precision in a category basis $P_{macro-c}$.

$$P_\mu = \frac{\sum_{i=1}^{K} a_i}{\sum_{i=1}^{K} (a_i + c_i)} \tag{4}$$

$$P_{macro} = \frac{\sum_{i=1}^{K} P_i}{K} \tag{5}$$

where K is the number of categories or the number of documents depending on the basis used.

Recall and F1 measures are computed in a similar way. In our experiments we have used these measures in order to prove the effectiveness of the studied system.

3 The Adaptive Selection of Base Classifiers in TECAT

We developed the TECAT tool for automatic text classification of HEP documents stored at the CERN digital library [6]. It implements the *Adaptive Selection of Base Classifiers* (ASBC) algorithm as classification strategy (shown in Figure 1), which trains and selects binary classifiers for each class independently. In this algorithm any type of binary classifier can be used, even a set of heterogeneous binary classifiers, like SVM [7], Rocchio [8] and others at the same time, but *(a)* it allows adjusting the binary

Input:
  a set of training documents D_t
  a set of validation documents D_v
  a threshold α on the evaluation measure
  a set of possible label (classes) L
  a set of candidate binary classifiers C
Output :
  a set $C' = \{c_1, ..., c_k, ..., c_{|L|}\}$ of trained
  binary classifiers
Pseudo code:
  $C' \leftarrow \emptyset$
  for-each l_i in L do
    $T \leftarrow \emptyset$
    for-each c_j in C do
      train-classifier(c_j, l_i, D_t)
      $T \leftarrow T \cup \{c_j\}$
    end-for-each
    $c_{best} \leftarrow$ *best-classifier*(T, D_v)
    if *evaluate-classifier*(c_{best}) $> \alpha$
      $C' \leftarrow C' \cup \{c_{best}\}$
    end-if
  end-for-each

Fig. 1. The one-against-all learning algorithm with classifier filtering.

classifier for a given class by a balance factor, and *(b)* it gives the possibility of choosing the best of a given set of binary classifiers.

The algorithm introduces the α parameter, resulting in the algorithm given in Figure 1. This value is a threshold for the minimum performance allowed to a binary classifier during the validation phase in the learning process. If the performance of a certain classifier is below the value α, meaning that the classifier performs badly, classifier and the class are discarded. The effect is similar to that of the *SCutFBR* [9], i.e. we never attempt to return a positive answer for rare classes. This algorithm has been found to outperform well known approaches like Ada-Boost [10] when applied on HEP data [3].

4 LVQ Integration into TECAT

The Learning Vector Quantization (LVQ) algorithm [11] has been successfully used in several applications such as pattern recognition, speech analysis, etc. This work proposes the use of the LVQ algorithm to accomplish the multi-label text categorization problem. However, the LVQ is used only as a binary classifier integrated in the TECAT system.

4.1 The LVQ Algorithm

The LVQ algorithm is a classification method, which allows the definition of a group of categories on the space of input data by reinforced learning, either positive (reward) or negative (punishment). The LVQ uses supervised learning to define class regions on the input data space. Weight vectors associated to each output unit are known as codebook vectors. Each class of input space is represented by its own set of codebook vectors. Codebook vectors are defined according to the specific task.

The basic LVQ algorithm is quite simple. It starts with a set of input vectors x_i and weight vectors w_k which represent the classes to learn. In each iteration, an input vector x_i is selected and the vectors w_k are updated, so that they fit x_i better. The LVQ algorithm works as follows:

In each repetition, the algorithm selects an input vector, x_i, and compares it with every weight vector, w_k, using the Euclidean distance $\|x_i - w_k\|$, so that the winner will be the codebook vectors w_c closest to x_i in the input space for this distance function. The determination of c is achieved by following decision process:

$$\|x_i - w_c\| = \min_k \|x_i - w_k\| \tag{6}$$

i.e.,

$$c = \arg\min_k \|x_i - w_k\| \tag{7}$$

The LVQ algorithm is a competitive network, and thus, for each training vector, output units compete among themselves in order to find the winner according to some metric. The LVQ algorithm uses the Euclidean distance to find the winner unit. Only the winner unit (i.e., the output unit with the smallest Euclidean distance with regard to the input vector) will modify its weights using the LVQ learning rule. Let $x_i(t)$ be an input vector at time t, and $w_k(t)$ represent the weight vector for the class k at time t. The following equations define the basic learning process for the LVQ algorithm:

$$\begin{aligned} w_c(t+1) &= w_c(t) + \beta(t)[x_i(t) - w_c(t)] \\ &\quad \text{if } x_i \text{ and } w_c \text{ belong to the same class} \\ w_c(t+1) &= w_c(t) - \beta(t)[x_i(t) - w_c(t)] \\ &\quad \text{if } x_i \text{ and } w_c \text{ belong to different class} \\ w_k(t+1) &= w_k(t) \text{ if } k \neq c \end{aligned} \tag{8}$$

where $\beta(t)$ is the learning rate, which decreases with the number of iterations of training ($0 < \beta(t) << 1$). It is recommended that $\beta(t)$ be rather small initially, say, smaller than 0.3, and that it decrease to a given threshold, ν, very close to 0 [11]. In our experiments, we have initialized $\beta(t)$ to 0.1.

4.2 Using LVQ with TECAT

The LVQ algorithm has been integrated into TECAT as a binary classifier. For this, LVQ has been used to work with only two possible classes: positive or negative. Thus, we

can pass as parameters to the algorithm the number of codebook vectors that we want to associate to each of this two categories. TECAT tries to train a LVQ classifier for each class, discarding both class and classifier if a minimal value of performance is not reached on a validation subset (the measure used is F1 and the threshold α on it is set to 0.1). Therefore, instead of having a set of codebook vectors for each class, we will have a binary LVQ version of the classifier for each class.

4.3 Adjusting the Codebook Number per Class

The HEP corpus is highly unbalanced, therefore, when learning on a binary space, the number negative samples clearly would surpass that of the positive ones. Following this reasoning, we could foresee a benefit on the classifier adaptability if the number of codebook vectors associated to each of the two sides is set according to the imbalance that the class reflects. For this reason, we have carried out different experiments to adjust the number of codebook vectors to every single class.

In order to study the effects of the number of codebooks assigned to each class we have accomplished several experiments with a varying number of codebook vectors per class. On the one hand, we have used the same number of codebooks per class. In this case we have accomplished three different experiments with $ncb0 = ncb1 = \{1,2,5\}$ where $ncb0$ is the number of codebooks assigned to class 0 and $ncb1$ is the number of codebooks assigned to class 1. These two classes represents the binary decision to take when assigning each class to a document.

On the other hand, we have assigned different number of codebooks per class depending on the number of negative and positive training samples. Thus, we have assigned 1, 2 and 5 codebooks to the class 1, i.e., $ncb1 = \{1,2,5\}$. However, to calculate the number of codebooks assigned to class 0, we have used the following formula:

$$ncb0 = ncb1 + \log_2(\frac{nneg}{npos}) \tag{9}$$

where $npos$ and $nneg$ are the number of training samples positives and negatives respectively. This fraction would be the inverse if the number of positive samples is larger than the number of negative ones (which occurs only for one class in the collection used). For example, if the class had 35 positive samples, and 1671 negative ones, and the value for *ncb1* were 2, then 6 codebook vectors would be trained to cover the negative side of the decision space.

5 Results

Depending on the number of codebook vectors assigned to each class we have accomplished six different experiments: three experiments with the same number of codebooks (ncbv1, ncbv2 and ncbv5) and three with different number of codebook (normcbv1, normcbv2 and normcbv5). Results for the 10 most frequent classes are shown in Table 3 and Figure 2. Tables 4, 5 and 6 show results of macro-averaging and micro-averaging for the different experiments.

Table 3. Results summary for averaged measures on 10 most frequent classes.

Precision	Recall	F1	Experiment
0.744244	0.570907	0.624045	ncbv1
0.757046	0.613313	0.662224	ncbv2
0.771196	**0.617652**	**0.674908**	ncbv5
0.768844	0.563419	0.631547	normcbv1
0.780134	0.611680	0.669962	normcbv2
0.773607	0.614498	0.674848	normcbv5

Table 4. Results summary on macro-averaged values in a document basis.

Precision	Recall	F1	Experiment
0.627956	0.404897	0.455048	ncbv1
0.627246	**0.450413**	0.490741	ncbv2
0.639530	0.447763	**0.495636**	ncbv5
0.652167	0.378170	0.448980	normcbv1
0.655089	0.406707	0.471558	normcbv2
0.645321	0.397321	0.461638	normcbv5

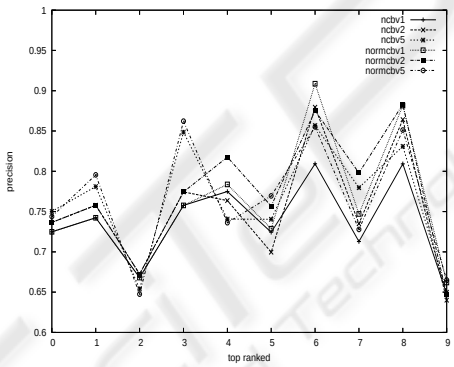

Fig. 2. F1 measurements for different configurations on the 10 most frequent classes.

The main remark is that using an uneven number of codebook vectors on positive and negative sides depending on class imbalance does not yields to better results in general. This is maybe due to a lack of positive samples in most classes. The decision on whether to use a different number of positive and negative codebook vectors should be restricted to scenarios where classes are well represented. This reasoning can be

Table 5. Results summary on macro-averaged values in a class basis.

Precision	Recall	F1	Experiment
0.500383	0.320962	0.362223	ncbv1
0.493239	0.327211	0.369442	ncbv2
0.627357	**0.386476**	**0.458187**	ncbv5
0.514602	0.274930	0.333245	normcbv1
0.517373	0.294911	0.352776	normcbv2
0.487381	0.262504	0.322848	normcbv5

Table 6. Results summary on micro-averaged values.

Precision	Recall	F1	Experiment
0.656294	0.415546	0.508883	ncbv1
0.656267	0.438942	0.526042	ncbv2
0.716130	**0.481063**	**0.575519**	ncbv5
0.710973	0.386372	0.500663	normcbv1
0.701892	0.415265	0.521809	normcbv2
0.710198	0.409406	0.519397	normcbv5

check in Table 3. Here, for most frequent classes, the balancing of codebook vectors improves the obtained results.

Nevertheless, a binary version of the LVQ algorithm stands as a valid solution to this multi-label problem. As can be seen, no major variations in evaluation measures are noticed, despite different configurations, so the algorithm is robust in this sense.

About the number of codebook vectors, it seems that a higher number of them tends to improve the overall classifier performance. Actually, the ncbv5 experiment is the one with the highest measured F1 in all averaging strategies.

6 Conclusions

A neural algorithm for multi-label has been studied. This algorithm is based in the LVQ learning method, integrating it into the ASBC approach. Some experiments have been carried out on a high unbalanced collection: the *hep-ex* partition of the HEP corpus. The results obtained show that, despite the complexity of the collection, the robustness of the algorithm remains against different configuration parameters.

As future work we plan to apply this algorithm on a more comparable collection in the text-categorization domain, specifically Reuters-21578. For this collection, results on applying LVQ with a codebook vector per class are available [12], which reported 0,61 macro-averaged precision and 0,73 micro-averaged precision. Also, since the ASBC algorithm allows to select among a set of possible classifiers, we will test our approach with a wide range of parameterizations of the LVQ algorithm on every class, letting the system decide which parameters are the best of each class.

Acknowledgements

This work has been partially supported by a grant from the Spanish Government, project TIMOM (TIN2006-15265-C06-03), and a grant from the University of Jaén, project RFC/PP2006/Id_514.

References

1. Sebastiani, F.: Machine learning in automated text categorization. ACM Comput. Surv. **34** (2002) 1–47

2. Montejo-Ráez, A.: Automatic Text Categorization of Documents in the High Energy Physics Domain. PhD thesis, University of Granada (2006)
3. Montejo-Ráez, A., Ureña López, L.: Binary classifiers versus adaboost for labeling of digital documents. Sociedad Española para el Procesamiento del Lenguaje Natural (2006) 319–326
4. Montejo-Ráez, A., Ureña López, L.: Selection strategies for multi-label text categorization. Lecture Notes in Artificial Intelligence (2006) 585–592
5. Vassilevskaya, L.A.: An approach to automatic indexing of scientific publications in high energy physics for database spires-hep. Master's thesis, Fachhochsule Potsdam, Institut fr Information und Dokumentation (2002)
6. Montejo-Ráez, A., Steinberger, R., Ureña López, L.A.: Adaptive selection of base classifiers in one-against-all learning for large multi-labeled collections. In et al., V.J.L., ed.: Advances in Natural Language Processing: 4th International Conference, EsTAL 2004. Number 3230 in Lectures notes in artifial intelligence, Springer (2004) 1–12
7. Joachims, T.: Text categorization with support vector machines: learning with many relevant features. In Nédellec, C., Rouveirol, C., eds.: Proceedings of ECML-98, 10th European Conference on Machine Learning. Number 1398, Chemnitz, DE, Springer Verlag, Heidelberg, DE (1998) 137–142
8. Lewis, D.D., Schapire, R.E., Callan, J.P., Papka, R.: Training algorithms for linear text classifiers. In Frei, H.P., Harman, D., Schäuble, P., Wilkinson, R., eds.: Proceedings of SIGIR-96, 19th ACM International Conference on Research and Development in Information Retrieval, Zürich, CH, ACM Press, New York, US (1996) 298–306
9. Yang, Y.: A study on thresholding strategies for text categorization. In Croft, W.B., Harper, D.J., Kraft, D.H., Zobel, J., eds.: Proceedings of SIGIR-01, 24th ACM International Conference on Research and Development in Information Retrieval, New Orleans, US, ACM Press, New York, US (2001) 137–145 Describes RCut, Scut, etc.
10. Schapire, R.E., Singer, Y.: BoosTexter: A boosting-based system for text categorization. Machine Learning **39** (2000) 135–168
11. Kohonen, T.: Self-organization and associative memory. 2 edn. Springer-Verlag (1995)
12. Martín-Valdivia, M., García-Vega, M., García-Cumbreras, M., Ureña López, L.: Text categorization using the learning vector quantization algorithm. In: Proceedings of Intelligent Information Systems. New Trends in Intelligent Information Processing and Web Mining (IIS:IIPWM-04), Zakopane, Poland, Springer-Verlag (2004)