

HIERARCHICAL TEXT CATEGORIZATION USING CODING MATRICES

Janez Brank, Dunja Mladenić, Marko Grobelnik

Department of Knowledge Technologies, Jozef Stefan Institute

Jamova 39, 1000 Ljubljana, Slovenia

Tel: +386 1 477 3778; fax: +386 1 477 3315

e-mail: {janez.brank,dunja.mladenic,marko.grobelnik}@ijs.si

ABSTRACT

We discuss the task of ontology population as a machine learning problem with a large hierarchy of classes. Since many machine learning methods are designed primarily for two-class problems, it is desirable to transform the multiclass classification problem into several two-class problems. Coding matrices are a unifying formalism for describing such transformations. We present an approach for constructing coding matrices in a greedy way, with a focus on achieving good performance with a tractable number of two-class classification models.

1 INTRODUCTION

In modern information systems, information about the problem domain is often organized in an ontology, i.e. a shared conceptualization of the domain of interest, expressed in machine-processable form. An ontology typically consists of concepts, instances of these concepts, and relations (between concepts and/or between instances). Ontologies can often be quite large, and constructing them can be an expensive and time-consuming task. One part of this process that can be automated relatively well on a large scale is the population of an ontology, i.e. the assignment of instances to concepts. Given a hierarchy of concepts and a set of instances, the task of ontology population is to identify, for each instance, which concept or concepts of the ontology this instance belongs to. We approach this as a problem of machine learning, assuming that some training data is already available (a set of instances for which the correct assignment to concepts is known). Unlike in typical machine learning settings where the number of classes is moderate, in the case of ontology population the number of concepts can be fairly large (several thousands and even hundreds of thousands), and the approach must take this into account.

2 RELATED WORK

We look at the ontology population task as a large-scale classification problem, with a large number of hierarchically organized classes (corresponding to concepts of the ontology) and instances. Additionally, the instances are likely to be represented by a large number of attributes. In many topic ontologies, the instances are textual documents, in which each word is treated as an attribute for the purpose of representing the instance for the machine learning algorithms. Therefore, our approach to machine learning for ontology population will draw largely upon techniques from the area of text categorization [1].

Support vector machines or SVM [2, 3] is one state-of-the-art method that is commonly used for text categorization. It

is particularly suitable for dealing with text classification problems, as it can handle a large number of attributes [4] and avoiding overfitting. Experiments have shown that SVM can lead to good and accurate models in many problem domains, including text categorization as one of the state-of-the-art methods.

Multi-class categorization. One of the disadvantages of the SVM is that, in its original formulation, it is targeted as binary (i.e. two-class) classification problems only. Various approaches have been considered for extending the SVM into the domain of multi-class problems, often at a considerable additional cost of the training. For example, [5] proposed an extension of the original optimization problem in which k models are sought simultaneously, where k is the number of classes. This approach does not scale well to problems with a many classes.

Most of the other approaches are based on translating the original k -class classification problem into several two-class problems. These approaches are usually not SVM-specific but could use any learning algorithm to train the models for the individual problems. When classifying a new instance, it is shown to the models for these two-class problems and the predictions of these models are then combined into an assignment of the instance to one of the k classes of the original multiclass problem.

The individual two-class problems can be defined in various ways. A typical example is “one vs. rest”, in which there is one two-class problem for each of the k classes of the original problem; in the two-class problem corresponding to class i , instances from class i are treated as positive and those from other classes are treated as negative. Thus each model tries to predict whether a given instance belongs to that particular class or not. An alternative is the “one vs. one” approach, in which there is one two-class problem for each pair of classes. Thus for a pair of classes (i, j) , where $1 \leq i < j \leq k$, the corresponding problem treats members of class i as positive, those of class j as negative, and the rest of the training instances is not used at all for this particular two-class problem. The individual problems are simpler than in the one-vs-rest approach, but the number of models is much larger, $k(k-1)/2$ rather than just k . For a large number of classes, this approach is infeasible. We can speed up at the classification by avoiding having each instance classified by all the models e.g., [6].

Coding matrices provide a conceptual unification of one-vs.-one, one-vs.-rest and other approaches translating the original k -class problem into several two-class

problems. Consider a k -class problem that has been translated into m two-class problems. The corresponding coding matrix M has k rows and m columns; the entry M_{cj} indicates how the instances from class c are used in the j -th two-class problem. Thus $M_{cj} = 1$ if instances from class c are treated as positive in the j -th two-class problem, $M_{cj} = -1$ if they are treated as negative, and $M_{cj} = 0$ if they are not used in the definition of the j -th two-class problem.

Thus, each column of the matrix corresponds to one of the two-class problems into which the original k -class problem has been translated. After training, each column now also corresponds to a binary classification model. Ideally, the model for the j -th problem should, when shown an instance from class c , predict +1 if $M_{cj} = 1$ and -1 if $M_{cj} = -1$. (If $M_{cj} = 0$, we cannot form any concrete expectations about the predictions of model j on instances from class c because no such instances were used in training that model.) Thus we can say that class c has been coded into the row of binary predictions $M^c = (M_{c1}, M_{c2}, \dots, M_{cm})$; *coding matrix*.

Once a row of actual predictions of the m binary models on some new instances are available, e.g. $y = (y_1, y_2, \dots, y_m)$, the final assignment of this instance to one of the m classes is obtained by comparing the row of predictions y to each row M^c of the matrix. The c for which the similarity between y and M^c is maximized is then selected as the final prediction of our multiclass model. If the sign of y_j expresses the prediction of the j -th model, and the absolute value of y_j expresses its confidence in the prediction, a formula for comparing the similarity can be a simple dot product of the form $\sum_{j=1..m} y_j M_{cj}$.

For example, coding matrices corresponding to the one-vs.-one and one-vs.-rest approaches:

$$\begin{bmatrix} 1 & -1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & -1 & -1 \end{bmatrix}$$

Various desirable properties of the coding matrix can be considered. For example, the rows should not be too similar to one another; if two rows are very similar, then even a small number of incorrect predictions may cause the row of predictions y to become more similar to the wrong row of the coding matrix. Similarly, the columns should not be too similar either, because this means that the corresponding models are effectively solving very similar two-class classification problems; their errors are therefore also likely to be more correlated. In this case many errors may be made simultaneously, which hampers decoding.

Another desirable characteristic of the matrix is sparsity, i.e. the proportion of entries that are set to 0. If a column is sparse, this means that training the corresponding binary classification model involves fewer training examples (and is therefore faster), and the resulting model is likely to be more accurate. However, if the matrix is very sparse, this also means that the rows are very sparse and it is therefore more difficult to ensure that the rows are different enough from each other. Sparsity also reduces the memory requirements if only the nonzero elements of the matrix are stored. Various families of coding

matrices have been considered in the literature [8, 9, 10]. In addition to the one-vs.-one and one-vs.-rest matrices, these include:

Matrices based on error-correcting output codes [7]. These codes are used e.g. in information theory to provide error-correcting bits during the transmission of data. They have mathematical guarantees on minimal row separation, i.e. ensuring that there is a certain minimum Hamming distance between any two rows. Their downside is that the resulting matrix is usually quite dense, i.e. all its entries are nonzero, and that m is typically $\geq k$. This may make it unfeasible to store the matrix for a large number of classes.

Random coding matrices. In this approach, entries of the matrix are randomly set to +1 or -1. Depending on whether we want a sparse matrix or not, some (or most) entries can be kept at 0. The performance of this family of coding matrices has been found by several authors to be as good as that of more carefully designed matrices (e.g. based on one-vs.-one, one-vs.-all, or error-correcting codes).

Matrices of theoretical interest. The smallest matrix that can theoretically distinguish k classes requires $\lceil \log_2 k \rceil$ columns. However, the corresponding binary classification problems would be extremely difficult, leading to poor performance of the ensemble as a whole. On the other extreme, a matrix can have at most $(3^k - 2^{k+1} + 1)/2$ different columns if we require that each column has at least one +1 and at least one -1 entry, and that no column is equal to or a negation of any other. However, such a matrix is intractably large for any but extremely small values of k .

3 CODING MATRICES FOR LARGE HIERARCHIES

The techniques for the construction of coding matrices presented in the previous section have several drawbacks when applied to problems with a large number of categories and when furthermore these categories are organized into a hierarchy. Most of these methods require at least $O(k)$ models to deal with a k -class classification problem, which could be problematic if the number of classes k is large and the individual models are relatively expensive to train (as is the case for the SVM). Thus, it would be desirable to have a method that focuses on constructing a matrix with a sublinear number of models. Another drawback is that the methods described so far are not aware of the hierarchical relationships between the classes. This will be addressed by our approach, which we present in this section.

3.1. Constraints - hierarchical organization of classes

The structure of the matrix must take the hierarchical organization of the original k classes into account. In particular, if class c is the ancestor of class c' in the hierarchy, and a particular model j uses one of them as positive and the other one as negative (e.g. $M_{cj} = 1$, $M_{c'j} = -1$), this would imply that the instances from class c' must be simultaneously negative and (since any instance of c' is also an instance of c) positive. Thus, it follows that whenever c is an ancestor of c' , the condition $M_{cj} M_{c'j} \geq 0$ should hold for all columns j of the coding matrix. This constraint is

relatively straightforward to incorporate in the random matrix generation algorithm.

Algorithm A: To construct the j -th column of the matrix:

```

1  set  $M_{cj} = 0$  for all  $c = 1, \dots, k$ ;
2  set  $Anc_1 = \{\}, Anc_{-1} = \{\}$ ;
3  while there are sufficiently few nonzero entries in the  $j$ th column,
   and not all pairs of classes  $(c, c')$  have been tried:
4     select two random classes,  $c$  and  $c'$ , such that neither is
       an ancestor of the other, and such that the pair  $(c, c')$  has
       not yet been considered in some previous iteration of this loop;
5     if  $M_{cj} \neq 0$  then  $A_c = \{M_{cj}\}$ 
       else if  $c \in Anc_1$  then  $A_c = \{1\}$ 
       else if  $c \in Anc_{-1}$  then  $A_c = \{-1\}$ 
       else  $A_c = \{-1, 1\}$ ;
6     initialize  $A_{c'}$  based on  $M_{c'j}$  analogously to step 5;
7      $A = \{(a, a') : a \in A_c, a' \in A_{c'}, a < a'\}$ 
8     if  $A = \{\}$ , go back to step 3;
9     let  $(a, a')$  be a random element of  $A$ ;
       set  $M_{c''j} = a$  for all  $c''$  that are descendants of  $c$  (incl.  $c'' = c$ );
       set  $M_{c''j} = a$  for all  $c''$  that are descendants of  $c'$  (incl.  $c'' = c'$ );
10    include  $c$  and all its ancestors in  $Anc_a$ ,
       include  $c'$  and all its ancestors in  $Anc_{a'}$ ;
11  end while;
```

In the sets Anc_1 and Anc_{-1} , we keep track of the ancestors of classes that have already been used as either positive or negative in the current binary problem. Membership of a class c in either of these two sets limits our choice of acceptable nonzero values for M_{cj} , represented by the set A_c that we compute in step 5. If the sets A_c and $A_{c'}$ indicate that c and c' cannot be given opposite labels, we try some other random pair of classes c and c' .

3.2. Greedy construction of the coding matrix

In this section we propose an approach for greedy construction of the coding matrix one column at a time. The families of coding matrices described in section 2 are all based on either constructing the whole matrix at once (e.g. one-vs.-one, one-vs.-rest, error-correcting codes), or constructing it one column at a time but with each column independent of the others (e.g. random coding matrices, where the only thing that connects different columns can be some general constraint e.g. regarding the density of the matrix).

It may be desirable to construct the matrix gradually, one column at a time (or at least a few columns at a time), while taking into account the part of the matrix that has already been constructed. In particular, if our goal is to maintain or improve the classification performance of the matrix as a whole (i.e. of the ensemble of binary classifiers implied by the matrix) while avoiding the need for an intractably large number of models, it makes sense to try constructing each new column of the matrix in such a way that the model for the binary problem defined by this new column will contribute as much as possible to the performance of the current ensemble of binary models. Therefore, we propose the following greedy approach for constructing the coding matrix:

Algorithm B: greedy construction of the coding matrix.

```

1  begin by initializing the first few columns of the matrix randomly,
   as described by Algorithm A in the previous section;
2  for each subsequent column  $j$ :
3     set  $M_{cj} = 0$  for all  $c = 1, \dots, k$ ;
```

```

4     set  $Anc_1 = \{\}, Anc_{-1} = \{\}$ ;
5     evaluate the current assembly of models, corresponding to
       columns 1, 2, ...,  $j-1$  of the matrix, on a validation set
6     let  $E = (E_{cc'})$  be the confusion matrix, i.e.  $E_{cc'}$  is the number of
       instances that belong to class  $c$  but were incorrectly
       predicted as belonging to class  $c'$ ;
7     while there are sufficiently few nonzero entries in the  $j$ th column,
       and not all pairs of classes  $(c, c')$  have been tried:
8         take next pair of classes  $(c, c')$  in decreasing order of  $E_{cc'} + E_{c'c}$ ;
9         for this pair  $(c, c')$ , perform steps 5–10 of Algorithm A;
10    end while;
```

This algorithm uses the confusion matrix to identify difficult parts of the learning problem, and constructs the next column of the matrix so that its model will focus on those problematic parts, hopefully correcting the mistakes made by the previous models. The measure used to determine which classes to focus on is simply the number of confusions. The principle is similar to the one known in boosting, but boosting works on the level of individual instances, modifying the training set but keeping the number of classes intact. By contrast, the approach presented here could be thought of as boosting on the level of classes.

If the original classification problem has k classes, the confusion matrix is in principle a $k \times k$ matrix, but it's very sparse as each instance from the validation set can contribute only a limited number of confusions. Storing the matrix in memory becomes tractable if only the nonzero entries are stored explicitly. When computing the confusion matrix, the hierarchical structure of classes must also be taken into account. Thus an instance that belongs to (or is predicted as belonging to) a class c also belongs to (or is predicted as belonging to) any ancestor of that class. Currently, we use the following approach to compute the confusion matrix:

```

1  set  $E_{cc'} = 0$  for all  $c$  from 1 to  $k$  and all  $c'$  from 1 to  $k$ ;
2  for each instance  $x$  from the validation set:
3     let  $c$  be the correct class label of  $x$ ,
       and  $A_c$  be the set of all ancestors of  $c$ ;
4     let  $c'$  be the label predicted for  $x$  by the current ensemble, and
       let  $A_{c'}$  be the set of all ancestors of  $c'$ ;
5     for each  $c_1 \in A_c - A_{c'}$  and each  $c_2 \in A_{c'} - A_c$ ,
6         increment  $E_{c_1 c_2}$  by 1;
```

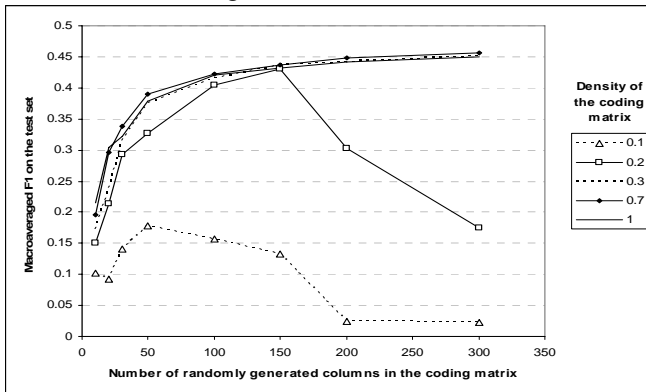
A problem with Algorithm B as described above is that the confusion matrix may not change much when a single new model is added to the ensemble. Therefore the next few models will be constructed similar to the current one, and will make similar prediction mistakes, causing poor performance of the ensemble as a whole. To avoid this problem, our current approach is to use a tabu list – a list of pairs of classes that have been used in the construction of the last few models and should therefore be ignored (skipped in step 8 of Algorithm B) when constructing the next model.

4 EXPERIMENTAL EVALUATION

In this section we present some steps towards an experimental evaluation of our proposed approach. We use a subset of the *dmoz* ontology, consisting of the Top/Sci-

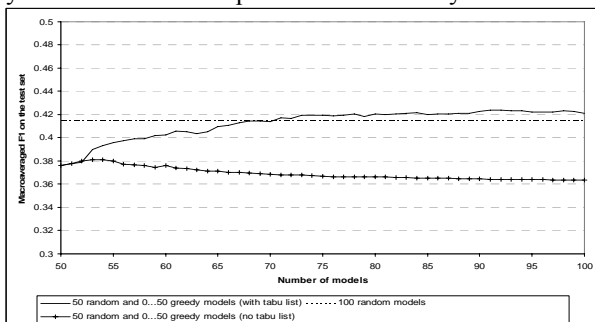
ence category, seven of its subcategories, and 72 of its subsubcategories. From each of the resulting 80 categories, we included 100 documents for training and an additional 100 documents for testing.

The following chart explores the effect of two parameters used in the construction of a random coding matrix: the number of columns in the matrix and the average number of nonzero elements in each row of the matrix (and thus the density or sparsity of the matrix). The several line series on the chart correspond to different levels of matrix sparsity (i.e. 0.1 represents matrices with approx. 10% nonzero entries, etc.). All performance are averages over ten random matrices.



This experiment indicates that excessively sparse matrices should be avoided (the two lowest lines in the graph), especially if they consist of a large number of models, because decoding can be problematic under these conditions and the performance of the ensemble as a whole can degrade. Note that the macro-averaged F_1 measure that was used to evaluate these ensembles is somewhat problematic in this setting since it completely disregards the hierarchical relationship between the classes.

The following experiment illustrates the benefits of the greedy matrix construction algorithm over the purely random approach. Starting with 50 random columns, then generating up to 50 additional columns using our greedy approach. For comparison, we show the performance of a purely random matrix with 100 columns. As this example shows, extending the initial ensemble of 50 random columns by just 20 additional greedily-constructed columns led to the same performance as adding 50 additional random columns. Note that the use of tabu lists proved crucial in this case, as without it there were too many similar models and performance actually decreased.



5 CONCLUSIONS AND FUTURE WORK

As the experiments in sec. 4 show, the current approach is an improvement over random coding matrices in the sense that it requires fewer models are required to achieve comparable or better performance. At the same time, it is not significantly more expensive than the approach based on random matrices; the main additional cost is more evaluation of the ensemble (after each addition of a new column).

For future work, there are many changes and refinements that may be considered. The current approach is a greedy technique for exploring the space of possible coding matrices. The greedy constraints could be relaxed, or other optimization techniques could be used instead, such as local optimization or even genetic algorithms. However, this may make the training process too time-consuming, especially for large hierarchies of classes. Another direction for further work is a more thorough experimental evaluation of the proposed approach and a comparison with other approaches, especially other families of coding matrices, with a focus on large topic ontologies. It would also be interesting to extend the approach proposed here and make it able to deal with changing ontologies (i.e. addition or removal of concepts, etc.).

Acknowledgement

This work was supported by the Slovenian Research Agency SEKT (IST-1-506826-IP), and PASCAL (IST-2002-506778).

References

- [1] Sebastiani, F., Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, March 2002.
- [2] Boser, B. E., Guyon, I. M., Vapnik, V. N., A training algorithm for optimal margin classifiers. *COLT*, 1992.
- [3] Cortes, C., Vapnik, V. N., Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995.
- [4] Joachims, T., Text categorization with support vector machines: Learning with many relevant features. *ECML*, 1998.
- [5] Weston, J., Watkins, C., Support vector machines for multi-class pattern recognition. *Proc. ESANN*, 1999.
- [6] Platt, J. C., Christianini, N., Shawe-Taylor, J., Large margin DAGs for multiclass classification. *NIPS 12*, The MIT Press, 2000.
- [7] Dietterich, T. G., Bakiri, G., Solving multiclass learning problems via error-correcting output codes. *JAIR*, 2:263–286, January 1995.
- [8] Berger, A., Error-correcting output coding for text classification. *Workshop on Machine Learning for Inf. Filtering*, IJCAI 1999.
- [9] Rennie, J., Rifkin, R., Improving multiclass text classification with the support vector machine. *MIT, AI Memo AIM-2001-026*, 2001.
- [10] Ghani, R., Using error-correcting codes for efficient text classification with a large number of categories. *M.Sc. Thesis, Carnegie Mellon University*, 2001.