# Large-scale Hierarchical Text Classification Using SVM and Coding Matrices

Janez Brank[1], Dunja Mladenić[1], Marko Grobelnik[1]

[1] Department of Knowledge Technologies,
Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia
{janez.brank, dunja.mladenic, marko.grobelnik}@ijs.si

**Abstract.** We deal with the problem of classifying textual documents into a topical hierarchy of categories. Multi-class classification problems such as this one are often dealt with by converting them into several two-class classification problems; a binary classifier can be trained for each of these problems and their predictions are then combined to form the final classification of a document into the topic hierarchy. The conversion from the original multi-class problem into a group of two-class problems can be succinctly described by a "coding matrix". In traditional approaches, the coding matrix is either completely random or (more commonly) completely fixed in advance (e.g. 1-vs-1, 1-vs-rest); in both cases, the training data does not affect the design of the coding matrix. Our approach constructs the coding matrix gradually, one column at a time, with each new column being defined in such a way that the new binary classifier attempts to rectify the most common mistakes of the ensemble of binary classifiers built up to that point. The goal is to achieve good performance with a smaller number of binary classifiers. We also present systematic experiments on a small dataset which demonstrate that good coding matrices with a small number of columns exist, but are rare.

**Keywords:** Semantic Web, ontologies, hierarchies, ontology population, ontology generation, algorithms, classification, machine learning, text mining, coding matrices, Support Vector Machines

## 1 Introduction

In modern information systems, information about the problem domain with which the system is dealing is increasingly commonly organized in the form of an ontology, which is a shared conceptualization of the domain of interest, expressed in some well-defined formal language and in a machine-readable 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, both in terms of the number of concepts and the number of instances. Construction of such ontologies can be an expensive and time-consuming task, especially if a lot of manual involvement

by human experts is needed. It is therefore desirable that at least some parts of this process are automated as far as possible.

One part of the ontology construction and maintenance process that lends itself relatively well to automatization 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.

This can be approached as a problem of machine learning. In the scenario considered by this paper, it is assumed that some "training data" is available, i.e. a set of instances for which the correct assignment to concepts is already known, and the problem is to train a model that will predict assignments (as accurately as possible) for new, previously unseen instances. 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.

The remainder of this paper is structured as follows. In Section 2, we present an overview of related work in machine learning and text classification on which the approach presented later draws. In Section 3, we describe our approach based on coding matrices and a greedy optimization strategy. In Section 4, we describe experimental evaluations of our approach in particular and an investigation of the space of coding matrices in general. In Section 5, we present some considerations for future work.

## 2 Related Work

For the purposes of this paper, the ontology population task has been formulated as a large-scale problem of supervised machine learning (i.e. classification), 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].

### 2.1 Support Vector Machines

One state-of-the-art method that is commonly used for text categorization is the support vector machine, or SVM [2, 3]. It has many desirable characteristics that make it particularly suitable for dealing with text classification problems, e.g. its ability to handle a large number of attributes, of which many (indeed most) are not by themselves really relevant or informative as far as predicting the target category is concerned [4]. The SVM is also often able to avoid overfitting the training data. It has a strong mathematical foundation from statistical learning theory, from which various results have been derived, such as guarantees regarding the error rate of an SVM

model, as well as heuristics for model selection (e.g. based on structural risk minimization). Experiments have shown that SVM can lead to good and accurate models in many problem domains, including text categorization where it is now one of the state-of-the-art methods.

### 2.2 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. The SVM approaches learning as an optimization problem in which each training instance is represented by a point in some (possibly very high-dimensional) vector space, and the aim of the learning algorithm is to find a hyperplane that separates the points of one class from those of the other class, while maximizing the margin (the distance of the nearest training points from the plane). 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 $k$ hyperplanes are sought simultaneously, where $k$ is the number of classes; but this optimization problem has $k$ times as many variables as the original problem for a two-class SVM, so this approach does not scale well to problems with a large number of 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 two-class 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 one defines $k$ two-class problems, each of which tries treats one of the original $k$ classes as positive and the other classes as negative. At the end, the instance is assigned to the class whose model expressed the greatest confidence that the instance belongs to its class.

An alternative is the "one vs. one" approach, in which there is one two-class problem for each pair of classes, using instances of one class as positive, those of the other class as negative, and ignoring the rest. Individual two-class problems are thus simpler and easier to train than in the "one vs. rest" approach, but there are now $k(k-1)/2$ of them, rather than just $k$. For the final assignment of an instance to one of the $k$ classes, the predictions of all the models are considered as votes; some authors have proposed schemes that avoid the need to have each new instance classified by all the models [6].

### 2.3 Coding Matrices

Coding matrices provide a conceptual unification of the one-vs.-one, one-vs.-rest and other families of approaches for 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, and

the entry $M_{cj}$ of the matrix indicates how the instances from class $c$ are used in the $j$-th of the two-class problems. 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 at all.

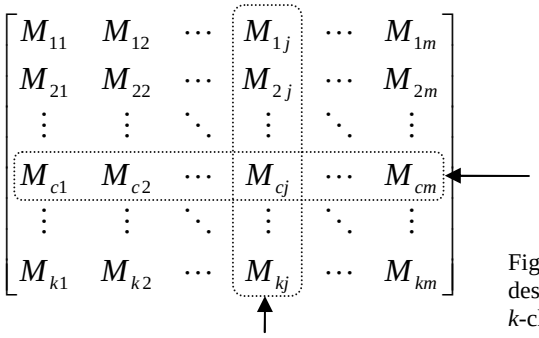



Figure 1. A coding matrix describing the decomposition of a $k$-class problem into $m$ two-class problems.

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 $j$-th model 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 $M^c = (M_{c1}, M_{c2}, \ldots, M_{cm})$; hence the term *coding matrix*.

Once a row of actual predictions of the $m$ binary models on some new instances are avilable, e.g. $\mathbf{y} = (y_1, y_2, \ldots, y_m)$, the final assignment of this instance to one of the $m$ classes is obtained by comparing the row of predictions $\mathbf{y}$ to each row $M^c$ of the matrix, e.g. with the dot product $\Sigma_{j=1..m}\, y_j\, M_{cj}$. The $c$ for which the similarity between $\mathbf{y}$ and $M^c$ is maximized is then selected as the final prediction of our multiclass model.

It is straightforward to express the one-vs.-one and one-vs.-rest approaches in this framework. For one-vs.-rest, we have $m = k$ and $M$ is a square matrix, with all the diagonal elements set to +1 and all the others to –1. For the one-vs.-one approach, we have $m = k(k - 1)/2$ and each column of the matrix has just two nonzero elements, one set to +1 and one to –1.

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 $\mathbf{y}$ 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. If this happens a lot, it may mean that when an error is made, it is made by many models simultaneously, which again makes it more likely that the row of predictions $\mathbf{y}$ will be decoded into the incorrect class label.

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 task involves fewer training examples (and is therefore faster), and the resulting model is likely to be more accurate. This is beneficial because errors of the binary models, if they are too common, can lead to incorrect decoding and thus the assignment of a wrong class label to an instance. 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 (since the entries that are set to 0 do not in general help with the decoding of a row of predictions into a class label). Sparsity of the matrix has another desirable characteristic: only the nonzero elements need to be stored, so the memory requirements may be tractable even if the matrix itself is quite large. (See sec. 4 for an experimental investigation of the effect of sparsity on the performance of the matrix.)

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 described above, these include matrices based on error-correcting output codes [7] (which provide some guarantees on minimal row separation) and random matrices (which have been found by several authors to work comparably well to more carefully designed matrix families). From a purely theoretical point of view, a matrix able to distinguish between $k$ classes will need to have at least $\lceil \log_2 k \rceil$ columns, and may have up to $(3^k - 2^{k+1} + 1)/2$ columns before they start to repeat or be negations of each other.

## 3 Greedy Construction of Coding Matrices for Large Hierarchical Multiclass Categorization Problems

The techniques for the construction of coding matrices that have been 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. One drawback is that 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 due to 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, given that $c'$ is a subclass of $c$, 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 $c = 1, \ldots, k$;
2 set $Anc_1 = \{\}$, $Anc_{-1} = \{\}$;
3 while there are less than a certain number of 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 $A = \{ (a, a') : a \in A_c, a' \in A_{c'}, a\, a' < 0 \}$
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 $c$ (including $c'' = c$);
set $M_{c''j} = a$ for all $c'$ that are descendants of $c'$ (including $c'' = c'$ );
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 $A_c$ for $M_{cj}$, represented by the set $A_c$ that we compute in step 5 of Algorithm A. Given the sets of acceptable nonzero values $A_c$ for $M_{cj}$ and $A_{c'}$ for $M_{c'j}$, we try to select a pair of values ($a$, $a'$) such that both are acceptable and one is +1 while the other is –1. If this is not possible, e.g. because both classes $c$ and $c'$ are already positive or have positive descendants (or are both negative, etc.), we try some other random pair of classes $c$ and $c'$.

Although the time complexity of the algorithm could theoretically be quadratic in the number of classes, $O(k^2)$, this could only happen in pathological cases. For any real-world ontology, the number of iterations of the main loop (lines 3–11) is linear in the desired number of nonzero elements in the column of the matrix, which is at most $k$ but typically less than that because we want the matrix to be sparse.

Note that our approach largely assumes that the hierarchic relationships between the concepts form a tree, i.e. that there are no cyclical relationships and that no concepts has more than one immediate superconcept. However, to handle cyclical relationships, use could be made of the fact that cyclical is-a relationships suggest that the concepts involved are equivalent, and thus each strongly connected component of the is-a graph should be treated as a single concept for the purposes of ontology

population. To handle concepts that have more than one parent, no special processing is strictly required, but the scalability of our approach can suffer.[1]

### 3.2 Approach to 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, rather than the whole matrix at once), 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 $c = 1, \ldots, k$;
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 (i.e. a set of instances that were not used during
    training but for which the correct class membership is known);
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 less than a certain number of nonzero entries in the $j$th column,
    and not all pairs of classes $(c, c')$ have been tried:
8  take the 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;

[1] For example, step 10 of Algorithm A assumes that traversing the set of ancestors of a concept is not too expensive; in a typical tree-shaped hierarchy, the set of all ancestors of any given concept $c$ is fairly small. On the other hand, if a concept may have multiple parents, the ancestor sets can grow quite large. For example, in the dmoz.org topic ontology, using just the `<narrow>` links results in a proper tree of topics, but adding the `<symbolic>` links results in a graph in which almost every concept has thousands of "ancestors".

This algorithm uses the confusion matrix to identify particularly difficult parts of the learning problem, and constructs the next column of the matrix so that the next model will focus on those problematic parts, hopefully correcting the mistakes made by the previous models. In this case, the measure used to determine which classes to focus on is simply the number of confusions, i.e. instances that belong to one class but are mistakenly predicted as belonging to another class. The principle is similar to the one known in boosting, but boosting works on the level of individual instances, modifying the training set for each model but keeping the number of classes intact (i.e. it does not attempt to e.g. define new classes by combining or modifying the classes of the original problem). By contrast, the approach presented here could be thought of as boosting on the level of classes. In traditional boosting, an instance had a greater probability of being chosen (for training the next model) if the existing models tended to misclassify it; in our method, a class has a greater probability of being chosen if the existing models tended to confuse it with some other class.

If the original classification problem has $k$ classes, the confusion matrix is in principle a $k \times k$ matrix, which can be intractably large if the value of $k$ is large. However, note that each instance from the validation set can contribute only a limited number of confusions, therefore the total number of nonzero elements in the matrix is also limited. This means that the confusion matrix can be very sparse and storing it in memory becomes tractable as long as 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;

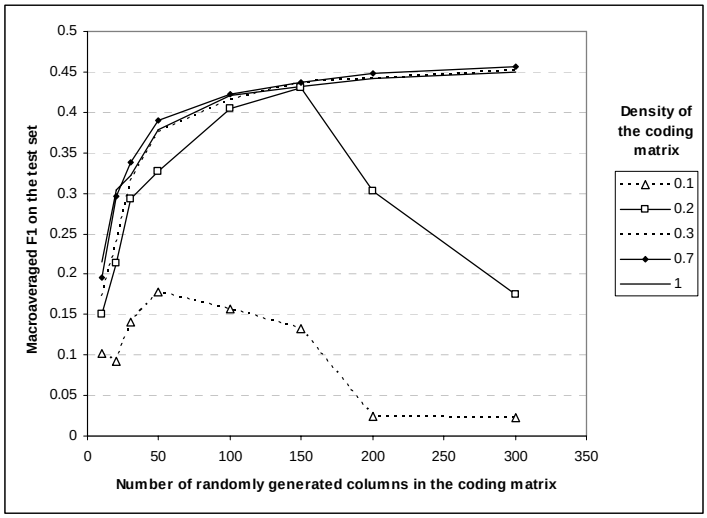
A problem with Algorithm B as described above is that the addition of a single new model might not lead to large enough changes in the predictions of the ensemble as a whole. Therefore the confusion matrix will not change quickly enough and, when constructing the next few columns, the order of pairs ($c$, $c'$) in decreasing number of confusions will remain largely the same. This can lead to many similar columns in the matrix, and the corresponding models are too correlated, making similar prediction mistakes and therefore causing poor performance of the ensemble as a whole. To avoid this problem, our current approach is inspired by tabu-search commonly used in local optimization, so we are using a tabu list – a list of pairs of classes that have been used in the construction of the last few models and should therefore not be used again when constructing the next model. Whenever, during the consideration of some pair ($c$, $c'$), one or both of the values $M_{cj}$ and $M_{c'j}$ change from 0 to a nonzero value, the pair ($c$, $c'$) is added to the tabu list, where it remains until at least a certain number of new models is added to the ensemble. In step 8 of Algorithm B, we skip any pairs that currently appear in the tabu list.

## 4 Experiments

In this section we present some steps towards an experimental evaluation of our proposed approach. For now, we are using a subset of the dmoz ontology, consisting of the Top/Science 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 thus the number of models in the resulting ensemble), 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 measures shown are averages over ten random matrices. The matrices used are purely random, not built with the greedy algorithm of sec. 2.3.

**Fig. 1.** Performance of purely random matrices, as a function of sparsity and no. of columns.

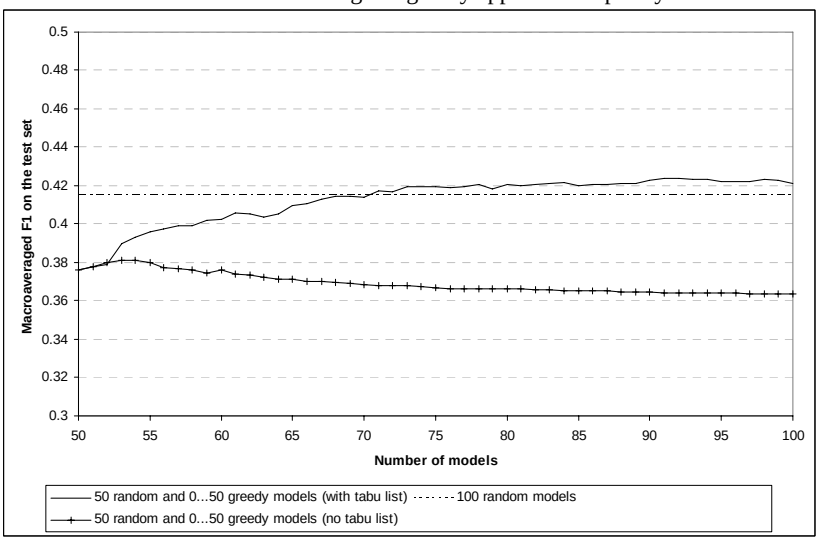



This experiment indicates that excessively sparse matrices should be avoided (the two lowest lines in the graph), especially if they consist of a large number of models, because decoding can be problematic under these conditions and the performance of the ensemble as a whole can degrade. Note that the macro-averaged $F_1$ measure that was used to evaluate these ensembles is less than ideally suited to this setting because it doesn't take the hierarchical relationship between the classes fully into account. In future work, we intend to also use other evaluation measures, e.g. the hierarchical adaptation of the Rand index proposed in [12], or the methodology used in [11].

The following experiment illustrates the benefits of the greedy matrix construction algorithm over the purely random approach. We started by generating 50 random columns, then generated up to 50 additional columns using the greedy approach of section 3. For comparison, we show the performance of a purely random matrix with a hundred columns. As this example shows, extending the initial ensemble of 50

models based on random columns by just 20 additional models based on greedily-constructed columns led to the same improvement in performance as adding 50 additional models based on random columns.

**Fig. 2.** Performance of matrices built using our greedy approach vs. purely random matrices.

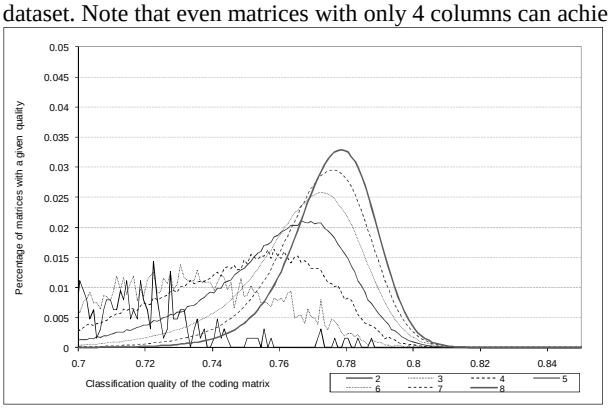



Note that the use of tabu lists proved crucial in this case, as without it no such improvements in performance were achieved; instead, the performance actually degraded as more and more columns constructed by the greedy heuristic were added to the model (because the columns were mostly nearly the same or were negations of one another).

**Systematic investigation of the space of coding matrices.** To get a better picture of what possibilities the coding matrix space offers, we carried out systematic experiments on a tiny dataset where such experiments are feasible. We selected 7 categories from the dmoz Science subtree, forming a 3-level hierarchy (Science, Math, Geometry, Algebra, Physics, Relativity, Quantum Mechanics). Given reasonable constraints on the composition of a column, there are just 36 different possible columns of the coding matrix (up to negation). Thus it is feasible to examine all possible coding matrices up to a certain width (for example, there are about 30 million ways to select 8 columns out of the 36 possible columns). For each matrix of up to 8 columns, we trained the corresponding ensemble of classifiers and evaluated its predictions (based on the average Jaccard coefficient between the correct and predicted category); the resulting score is also a score of the matrix itself (of its suitability for our classification task). From this we can estimate the distribution of matrix scores, e.g. answer the question how many matrices lead to a classification score above a certain threshold. The interesting result here is, as the following table and chart show, that the best matrices of 4 columns perform comparably well as the best matrices of e.g. 7 or 8 columns. This shows that a large number of columns is not strictly necessary; good matrices with a small number of columns do exist, but they are less common than good matrices with more columns. In other experiments, the

results of which are omitted here due to lack of space, we observed that the best performing matrices tend to be those that also have the greatest average inter-row and inter-column Hamming distance.

**Fig. 3.** Distribution of the performance of coding matrices of up to 8 columns on a small dataset. Note that even matrices with only 4 columns can achieve near-best performance.




| No. of columns | No. of matrices with this many columns | Median performance | Best performance |
|---|---|---|---|
| 1 | 36 | 0.5975 | 0.7105 |
| 2 | 630 | 0.6755 | 0.7885 |
| 3 | 7140 | 0.7225 | 0.8025 |
| 4 | 58905 | 0.7485 | 0.8315 |
| 5 | 376992 | 0.7615 | 0.8355 |
| 6 | 1947792 | 0.7685 | 0.8385 |
| 7 | 8347680 | 0.7735 | 0.8395 |
| 8 | 30260340 | 0.7765 | 0.8345 |

## 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), but a careful implementation can save some time by updating the predictions in an incremental fashion after each new model is added to the ensemble.

The current approach for constructing coding matrices is greedy in two aspects. Firstly, once a new column is added to the matrix, it subsequently remains unchanged. Secondly, when constructing a new column, the pairs of classes ($c$, $c'$) are processed strictly in descending order of the number of confusions. Both of these things could in principle be changed; it might sometimes be beneficial to modify or discard an existing column of the matrix, and it might be beneficial to process the classes in a less rigid order when deciding which classes should be used as positive and which as negative for the next column of the matrix. For example, the confusion matrix in effect defines a graph on the classes, with $E_{cc'}$ being the weight of the link from $c$ to $c'$; a max-cut algorithm could be used to divide the classes into positive and negative ones. Alternatively, the problem of constructing the coding matrix could be interpreted directly as what it really is, namely an optimization problem in which we are looking for the best matrix (i.e. the one leading to the ensemble with the smallest error rate) through the space of all possible matrices. Instead of a greedy algorithm, other common optimization strategies could be used, such as local optimization or even genetic algorithms. However, the risk with many such alternatives is that they 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. In doing this we will particularly focus on the population of large topic ontologies, such as dmoz.org (with hundreds of thousands of concepts and several million instances).

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.). For now, a pragmatic solution would be, when deleting a concept, to simply ignore its row when deciding how to decode a vector of predictions into a class label (by comparing it with the rows of the coding matrix); and when adding a new concept, a new model could be added to the ensemble, dealing specifically with separating this concept from the others. After some time, as more changes to the ontology accumulate, the whole ensemble could be re-trained (with a new coding matrix). Alternatively, if the learning algorithm that is used to train the individual binary classifiers supports incremental updates, we could simply add a new row to the matrix (whenever a new class $c$ is added) and update all those models $j$ for which $M_{cj}$ is nonzero (because their training set has gained either some additional positive or some additional negative examples, depending on the sign of $M_{cj}$). Incremental updates of this kind could be done elegantly with e.g. naive Bayes models, but not with SVM models (which is what we use now).

## 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. Proc. of the 5th ACM Workshop on Computational Learning Theory, 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. Proceedings of the 10th European Conference on Machine Learning, 1998.
5. Weston, J., Watkins, C., Support vector machines for multi-class pattern recognition. Proceedings of the 7th European Symposium on Artificial Neural Networks, 1999.
6. Platt, J. C., Christianini, N., Shawe-Taylor, J., Large margin DAGs for multiclass classification. Advances in Neural Inf. Processing Systems 12, The MIT Press, 2000.
7. Dietterich, T. G., Bakiri, G., Solving multiclass learning problems via error-correcting output codes. Journal of Artificial Intelligence Research, 2:263–286, January 1995.
8. Berger, A., Error-correcting output coding for text classification. Workshop on Machine Learning for Information Filtering, IJCAI 1999.
9. Rennie, J., Rifkin, R., Improving multiclass text classification with the support vector machine. Massachusetts Institute of Technology, 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.
11. Mladenić, D., Machine Learning on non-homogeneous, distributed text data. Ph.D. thesis, University of Ljubljana, 1998.
12. Brank, J., Grobelnik, M., Mladenić, D., Automatic evaluation of ontologies. In: Kao, A., Poteet, S. R. (eds.) Natural Language Processing and Text Mining, pp. 193--219. Springer, London (2007)