CIE-9-MC Code Classification with Knn and SVM

David Lojo[1,2], David E. Losada[3], and Álvaro Barreiro[1]

[1] IRLab. Dep. de Computación
Universidade da Coruña, Spain
[2] Servicio de Informática, Complexo Hospitalario Universitario de Santiago
Santiago de Compostela, Spain
[3] Grupo de Sistemas Inteligentes, Dep. de Electrónica y Computación
Universidade de Santiago de Compostela, Spain

Abstract. This paper is concerned with automatic classification of texts in a medical domain. The process consists in classifying reports of medical discharges into classes defined by the *CIE-9-MC* codes. We will assign *CIE-9-MC* codes to reports using either a *knn* model or support vector machines. One of the added values of this work is the construction of the collection using the discharge reports of a medical service. This is a difficult collection because of the high number of classes and the uneven balance between classes. In this work we study different representations of the collection, different classication models, and different weighting schemes to assign *CIE-9-MC* codes. Our use of document expansion is particularly novel: the training documents are expanded with the descriptions of the assigned codes taken from *CIE-9-MC*. We also apply SVMs to produce a ranking of classes for each test document. This innovative use of SVM offers good results in such a complicated domain.

1 Introduction

The process of automatic text classification can be defined as follows [5]. Given a static set of classes $C = \{c_1, \cdots, c_n\}$, and a collection of documents to classify (D), the goal is to find a classification function $\Phi = D \times C \rightarrow \{1, 0\}$.

Classification is present in most of the daily tasks. One of these classification tasks is carried out in the Hospitals: the coding of the diagnoses and procedures in medical episodes. When a patient is discharged, a specialized medical doctor writes a report that includes the most relevant data occurred in the clinical episode. These reports are later assigned *CIE-9-MC* codes by a dedicated office (the codification service). There is a team of medical doctors (the coders) who read the discharge report (including the set of diagnoses) and assign *CIE-9-MC* codes. This coding is an international system of numerical categories that are associated to diseases according to some previously established criteria.

The assignment of *CIE-9-MC* codes to a clinic episode has the following main elements:

– The main diagnosis, DxP. It is the disease which is established as the cause of the admission by the doctor who treated the patient.

J. Mira et al. (Eds.): IWINAC 2009, Part II, LNCS 5602, pp. 499–508, 2009.

– The secondary diagnoses, DxS. These are the other diseases that are present at the moment of the admission, or the ones which occurred while the patient was in the Hospital.

This is a supervised multi-class problem. The system learns from a known set of correctly classified cases (assigned by the coders). A document can belong to several classes (a discharge report can have several *CIE-9-MC* codes assigned), and the number of classes varies between documents. The purpose of our research is to build an automatic system that, given a new discharge report to be classified, constructs a ranking of possible codes. In a fully automatic setting, this ranking could be automatically used to assign codes. In a semi-automatic setting, the ranking would be presented to a human who would make the final decision.

We use *knn* and *Support Vector Machines (SVM)* classifiers. Our work with *knn* is similar to the study on knn classifiers in a medical domain reported by Larkey and Croft [4]. However, we introduce here the following variants in:

– The representation of the documents. We use different representations of our collection: the complete texts, the diagnosis part of the texts, and the complete texts expanded with the descriptions of the *CIE-9-MC* codes (document expansion).
– The retrieval techniques. We use different document retrieval models supported by the platforms Lemur and Indri[1].
– The weighting schemes. We use different variants to weight the *CIE-9-MC* codes.

2 Construction of the Collection

To build the collection, we first made an study of the services that produce discharge reports using electronic documents. From this analysis we selected the Internal Medicine service of the Hospital of Conxo, which is one of the hospitals in the *Complexo Hospitalario Universitario de Santiago*, Spain. This selection was based on the high number of documents available, the large size of the reports, the uniform format of the documents, and the complexity of the diagnoses utilized by this service.

The final collection is composed of the discharge reports from jan 2003 to may 2005, with a total of 1823 documents. We randomly split the collection into two parts: 1501 training documents and 322 test documents. There are 1238 different classes in the training set and 544 different classes in the testing set. There are 71 classes that are present in the test set but do not appear in the training set. The 74 documents associated to these classes were not be discarded because these documents have usually other classes assigned and, furthermore, we want the benchmark to reflect a real setting (there are more than 21k *CIE-9-MC* codes and a given training set hardly contains every single code). Table 1 reports the basic statistics of the collection.

[1] www.lemurproject.org

Table 1. Statistics of the collection

	Training	Test
# docs	1501	322
Size	5963Kb	1255Kb
Avg # codes per doc	7.06	7.05
Max # codes per doc	23	19
Avg # terms per doc	519.5	508.1
Min-Max # terms per doc	64-1386	109-1419

3 Text Classification Based on *knn*

Classification methods based on *knn* utilize a similarity or distance measure between documents. The basic idea is that an incoming report, d_{new}, will be classified according to the classes assigned to the training documents that are d_{new}'s k nearest neighbors. This classification method is popular because it is simple, intuitive, and easy to implement. Furthermore, it has shown to perform well in other studies [1,7], particularly when the collection is unbalanced. This is our case here.

The *knn* method retrieves initially k training documents that are similar to the test document, d_{new}. Then, it assigns *CIE-9-MC* codes to d_{new} according to the codes associated to the retrieved documents. In our work, we use *Lemur*, a popular Information Retrieval platform, to support the retrieval phase. An index is built from the training set of documents and the test documents act as queries against the index. Each retrieved document has a similarity score and the list of retrieved documents is sorted in decreasing order of this score. Each code associated to every retrieved document becomes a candidate to be assigned to the test document. Table 2 presents this rank, including the codes associated to the retrieved documents. Every retrieved document has a code associated to the main diagnosis (main code) and several secondary codes associated to other diagnoses reported by the doctors.

Although some studies suggest to use $k = 20$, we did experiments with varying k. Given the ranked documents, the next step is to produce a ranking of codes for the test document. We use the following expression: $Score_c = \sum_{i=1}^{i=k} sim_i \cdot w_{ic}$, where w_{ic} is the weight associated to code c in document i. For every test document, a list of possible codes ranked by decreasing $Score_c$ is produced. Regarding w_{ic} we evaluated several alternatives. The simplest one is the *baseline* weighting method, where $w_{ic} = 1$ when the code c is assigned to the training

Table 2. Ranking of documents in decreasing order of similarity to a test document

Doc	Rank	sim_i	Main Code (DxP)	Secondary Codes (DxS)
51007762	1	-5.60631	787.91	787.01 553.3 ...
41000982	2	-5.63082	507.0	491.21 518.84 ...
...	:	...	...	... ...
...	k	...	...	... ...

document i and $w_{ic} = 0$ otherwise. Other variants of this weighting scheme will be discussed later.

Note that Lemur implements different IR models and some of them (e.g. the one used to produce the ranking shown in Table 2) return negative similarity values. Since the definition of $Score_c$ requires positive similarities, we introduce the following normalization: $Score_{nc} = \sum_{i=1}^{i=k} e^{sim_i} \cdot w_{ic}$.

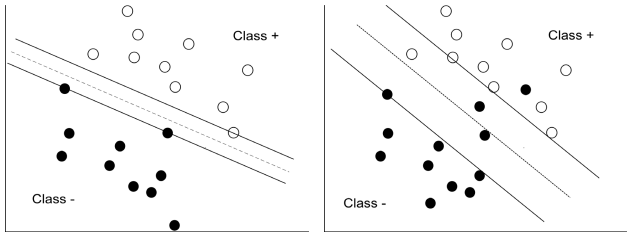
4 Text Classification with SVMs

Support Vector Machines (SVMs) are learning methods proposed by Vapnik [6] that have proved to be very effective in Text Classification [7], and in many other learning problems. SVMs deal naturally with binary (i.e. two-class) classification problems. A SVM model permits to define a linear classifier based on a hyperplane that acts as a border between the two classes. The elements to be classified (documents in our case) are represented using a vector space model. Let us first assume that the documents from each class are separable in this representational space. SVMs look for a hyperplane that separates the classes and, among the alternatives, the hyperplane that is maximally far away from any document is selected. The distance between the hyperplane and the nearest elements is called *margin* and the elements of each class that are the closest points to the hyperplane are referred to as *support vectors*. This is illustrated in Figure 1(a).

Formally, given a training set represented as $\{(x_1, y_1), ..., (x_n, y_n)\}$, where x_i is a vector ($x_i \in R^k$) and $y_i \in \{-1, 1\}$ indicates the membership of x_i to one class or another. The x_i elements can be separated by a hyperplane with the form $w^T \cdot x + b = 0$, where w is a weight vector (perpendicular to the hyperplane) and b is a constant. The classifier is $f(x) = sign(w^T \cdot x + b)$.

It can be proved that finding the maximum margin hyperplane can be expressed through the following minimization problem [6]: Find w and b such that: a) $\frac{1}{2} w^T w$ is minimum, and b) $\forall x_i, y_i : y_i(w^T x_i + b) \geq 1$. There is plenty of studies in the literature on a wide range of optimization techniques to resolve this problem. We skip here any further details about these methods.

In real applications, classification problems are hardly linearly separable. Therefore, it is often necessary to permit that the above conditions do not hold



(a) Linearly separable case (b) No linearly separable case

Fig. 1. SVM in two dimensions

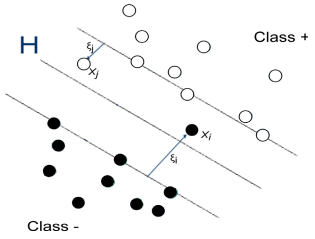


Fig. 2. SVM in two dimensions with slack variables

for all the examples. The usual strategy to deal with these situations is to allow that the hyperplane makes some mistakes (i.e. some points are misplaced), as shown in Figure 1(b). Formally, this means that we introduce *slack* variables into the model. For each x_i, we associate a ξ_i value as follows. A nonzero value for ξ_i allows x_i to not meet the margin requirement at a cost proportional to the value of ξ_i. This situation is depicted in Figure 2. According to this, the slack variables will have a value of zero when the point is correctly situated, and a positive value when the point is misplaced. This new learning problems is formally defined as: Find w, b, $\xi_i \geq 0$ such that: a) $\frac{1}{2}w^T w + C \cdot \sum_i \xi_i$ is minimum, and b) $\forall x_i, y_i : y_i(w^T x_i + b) \geq 1 - \xi_i$

The new minimization problem involves a tradeoff between how large we can make the margin, and the amount of elements that can be wrongly classified. Obviously, we could maximize the margin by simply augmenting the number of wrongly classified elements, but the quality of the classifier would be harmed. The C constant is a way to control this *overfitting* tradeoff. With a high C, the classification will be stricter and we allow less wrongly classified examples (the margin is reduced). A low C means that a more flexible classification is implemented, with larger margin but with more wrongly classified examples. In our empirical study, different C values will be tested in order to understand properly the effect of this tradeoff in the context of our difficult problem.

The approach described above works well with linearly separable datasets that only have a few exceptions or noisy points. However, some problems do not fit this pattern. There are ways to transform a not linearly separable problem into a linearly separable one. A given classification problem is much more likely to be linearly separable if it is transformed into a new classification problem that has a higher dimension. The vectors x_i are mapped into a higher dimensionality space using a non-linear transformation of the input space, $\Phi(x_i)$. Next, the SVMs learn the maximum margin hyperplane in the context of the expanded space. Generally, it is complex to compute the Φ mapping but, for learning purposes, it is sufficient to be able to compute the internal product between points in the new space: $\Phi(x_i)^T \Phi(x_j)$. If the product can be calculated efficiently using the original data (i.e. without having $\Phi(x_i)$ and $\Phi(x_j)$), then the learning problem can be solved in an efficient way. A *kernel* function, $K(x_i, x_j) = \Phi(x_i)^T \Phi(x_j)$, corresponds with this internal product in the expanded space of characteristics.

4.1 Application in the Clinical Domain

Our *CIE-9-MC* code assignment problem is inherently multi-class but SVMs are originally designed to do binary classification. Two main alternatives are discussed in the literature to apply SVMs when the number of classes, c, is greater than two [2]:

- (1-vs-all): it builds c one-vs-rest classifiers and chooses the class whose the hyperplane classifies the test document with the largest margin.
- (1-vs-1): it builds $\frac{c \cdot (c-1)}{2}$ one-vs-one classifiers (one for each possible pair of classes), applies the test document to every classifier and chooses the class that is selected by the most classifiers.

We use here 1-vs-all because it involves the construction of fewer classifiers (one per class). Observe that these methods are designed to assign a single class to each test document. Since we need to assign several codes to every test document (or, more generally, we need to build a ranking of codes for each test document), we adapt 1-vs-all as follows. The margin between the test document and the hyperplane associated to every class is regarded as a fitness measure for the document and the class. Hence, classes are ranked by decreasing order of the margin between the test document and the class' hyperplane.

As argued above, we need a vectorial representation of the documents in a space of characteristics. We opted here to represent documents as vectors of tf/idf weights (each dimension represents a term of the vocabulary). This weighting method has been applied thoroughly in the literature of IR. We used SVM^{light} [3] to implement the SVM learning process.

5 Evaluation Metrics

The evaluation metrics described in this section require the existence of a gold standard. In our case, we know the correct codes for each test document because every training or test document has a list of classes assigned by the coders. Of course, the list of codes associated to the test documents is only used for evaluation purposes. We adopt the following metrics, which have been used in the past for evaluating classifiers of clinical records [4]:

- *Average 11 point precision*: Precision and recall are standard IR evaluation measures. In our case, precision is the proportion of codes suggested by the classifier that are correct. Recall is the proportion of all correct codes that have been suggested by the classifier. Average precision is computed across precision values obtained at 11 evenly spaced recall points.
- *Top candidate*: proportion of cases in which the main code is the top candidate suggested by the classifier.
- *Top 10*: proportion of cases in which the main code is in the top 10 candidates.
- *Recall 15*, *Recall 20*: level of recall in the top 15 or top 20 candidates.

6 Experiments with *knn*

The same preprocessing was applied to test and training documents. We used an stoplist to remove common words, and we did not apply stemming because it does not usually produce benefits in medical domains [4]. For the retrieval step we selected the following retrieval models: *Indri*'s retrieval model, and two variations of the IR vectorial model (referred by *Lemur* as *tf/idf* and *cosine*).

CIE-9-MC codes have the format $CCC.S[X]$, where CCC is the category or section, S is the subcategory and X is a subclassification of the subcategory (it only exists for some subcategories). We evaluate here two different classification problems: category classification (i.e. assign properly the CCCs without regard to subcategories or subclassifications), and code classification (i.e. assign properly the whole code), which is a fine-grained classification and, therefore, it is harder.

6.1 Documents Representation

We created three representations of the collection, namely:

- *Diagnoses*: contains the sections of the discharge report where the medical doctor wrote the diagnoses (i.e. the rest of textual explanations in the report are discarded).
- *Total*: the complete discharge report is considered.
- *Total + CIE-9-MC*: composed by the complete discharge report plus the textual descriptions of the *CIE-9-MC* codes assigned by the coders. The training documents are therefore expanded with code descriptions that are obtained from the *CIE-9-MC* taxonomy.

Observe that the information encoded for the test and training documents is the same with the *Diagnoses* and *Total* representations. In contrast, with *Total + CIE-9-MC*, the representation of the training and test documents is uneven: training documents incorporate additional descriptions from the assigned codes but test documents are not expanded because no information on assigned coded is available at testing time.

Table 3 reports the performance results obtained with $k = 20$, the baseline weighting and Indri's IR model. These results show that *Total* and *Total + CIE-9-MC* are the most reliable representations for both classification problems. We also did some experiments with $k = 10$ and $k = 30$ but concluded that $k = 20$ is the most robust configuration.

Indri's retrieval model is a competitive IR method based on combining statistical language models and inference networks. However, it might be the case that other IR models are better than Indri for this knn problem. So, we compared Indri against two other IR models implemented by Lemur (tf/idf and cosine). This comparison, which is reported in Table 4, was done for the code classification problem using the Total representation. The tf/idf model is clearly inferior to the other models. On the other hand, cosine looks slightly superior to Indri.

Still, the results are not good enough to build and automatic classification system. Some of the metrics (e.g. Top candidate) show poor results. Next, we propose variations that improve the performance of the classifiers.

Table 3. Performance results. knn model (k=20, baseline weighting, Indri's IR model).

Representation	AvgPrec	TopCand.	Top10	Rec15	Rec20
		Code Classification			
Diagnoses	44.0	14.9	58.7	52.6	57
Total	43.1	16.1	64.9	52.5	57.7
Total + *CIE-9-MC*	43.8	17.4	64.3	53.1	58.2
		Category Classification			
Diagnoses	52.0	21.1	67	60.8	67.9
Total	51.2	22.7	74.2	62.4	67.7
Total + *CIE-9-MC*	51.8	24.5	73.9	62.9	68.2

Table 4. Performance results for code classification. knn model (k=20, baseline weighting, Total representation).

Model	AvgPrec	TopCand.	Top10	Rec15	Rec20
Indri	43.1	16.1	64.9	52.5	57.7
tf/idf	40.1	10.5	55.6	50.7	55.6
cosine	45.0	17.1	65.5	54.3	60.0

Effect of the weighting system. The results reported above were obtained with the baseline weighting, which is rather simplistic. Rather than assigning a weight equal to one to every code assigned to the retrieved documents, we will now assign a weight *greater than* one to the main code assigned to every retrieved document and a weight equal to one to the secondary codes. In this way, the main codes receive extra weight in the classification. Table 5 presents the results obtained with varying weights for the main codes.

These results show that Top candidate and Top 10 improve as the weight given to main codes increases. In contrast, Avg. Precision, Recall 15 and Recall 20 tend to decrease slightly with higher weights. However, the improvements in Top candidate and Top 10 are very substantial in comparison with the decrease of the other measures. This shows that the weighting strategy described above works well for these classification problems.

Table 5. knn model (k=20, Total representation, Indri model)

Weight (Main code)	AvgPrec	TopCand.	Top10	Rec15	Rec20
		code classification			
1	43.1	16.1	64.9	52.5	57.7
1.5	42.5	28.9	68.9	52.4	57.5
1.8	41.7	31.9	69.9	52.3	57.5
2.3	40.8	34.5	73.3	51.0	54.3
2.5	40.5	34.5	73.3	50.9	54.3
2.7	40.3	35.4	73.6	50.9	54.3
4.3	37.2	37.3	76.7	45.5	52.7
		category classification			
1	51.2	22.7	74.2	62.4	67.7
1.5	50.6	33.8	77.0	62.4	67.5
1.8	50.3	36.0	78.6	62.4	67.4
2.3	49.3	38.8	80.1	61.2	65.7
2.5	48.9	39.1	80.1	61.2	65.7
2.7	48.5	40.3	80.4	61.2	65.7
4.3	46.0	41.3	82.9	57.0	64.5

Table 6. SVM, Total representation, linear kernel

C	AvgPrec	TopCand.	Top10	Rec15	Rec20
		code classification			
Default	58.1	16.1	74.8	67.3	72.8
0.5	59.4	16.7	73.2	67.3	72.8
1000	59.4	16.7	73.2	67.3	72.8
		category classification			
Default	66.0	22.0	84.1	77.6	82.2
0.5	67.3	22.9	83.2	77.8	82.3
1000	67.3	22.9	83.2	77.8	82.3

Table 7. knn vs SVM

	AvgPrec	TopCand.	Top10	Rec15	Rec20
		code classification			
knn	40.3	35.4	73.6	50.9	54.3
SVM	59.4	16.7	73.2	67.3	72.8
		category classification			
knn	48.9	39.1	80.1	61.2	65.7
SVM	67.3	22.9	83.2	77.8	82.3

6.2 Experiments with SVM

The SVM experiments were done with the *Total* representation, which worked reasonably well for knn. We ran classifications using varying C values, and with the following kernels: linear, polynomial and gaussian. However, we only report here results for linear kernels because these kernels worked better than non-linear kernels. The results are presented in Table 6[2].

6.3 Comparing Knn and SVM

We selected the most robust knn configurations (Table 5, weight=2.7 for codes and weight=2.5 for categories) and compared them against the best SVM configurations. Table 7 presents this comparison. This shows that knn with proper weighting is very effective to achieve good Top Candidate performance. However, SVM beats knn in nearly all the remaining cases. The knn classifier might be useful if we were to select a single class for every test document. However, as argued above, the average number of codes per document is around 7 and, therefore, Avg. Precision, Top 10, Recall 15 and Recall 20 are more important than Top Candidate in this domain. These results indicate that SVM is a better classifier than knn for our classification problem in the medical domain.

7 Conclusions

In this paper we presented preliminary experiments on different classifiers that automatically assign *CIE-9-MC* codes to medical documents. We created a new collection composed of discharge reports from an Internal Medicine service and we experimented with different representations of the collection. Comparing knn

[2] The *Default* setting for C is $n/\sum_{i=1}^{n} x_i \cdot x_i$, where n is the number of training documents.

against SVM we found that knn is better than SVMs to identify the main code associated to a given report. However, SVMs are more adequate than knn for supporting the medical coding process because we need to find automatically as many codes as possible and SVMs show a more consistent behavior in terms of recall. This is a new demonstration of the learning power achieved with SVMs. The performance results obtained here are good enough to build a system that constructs a ranking of candidate codes for every new discharge report. This would be presented to a medical coder who would benefit from the availability of this ranked list.

Acknowledgements

This research was co-funded by FEDER, *Ministerio de Ciencia e Innovación* and *Xunta de Galicia* (*Consellería de Innovación e Industria* and *Consellería de Educación e Ordeación Universitaria*) under projects TIN2008-06566-C04-04, 07SIN005206PR, and 2008/068.

References

1. Aas, K., Eikvil, L.: Text categorisation: A survey. Technical report, Norwegian Computing Center (1999)
2. Hsu, C.W., Lin, C.J.: A comparison of methods for multiclass support vector machines. IEEE Transactions on Neural Networks 13(2) (2002)
3. Joachims, T.: Making large-scale svm learning practical. In: Advances in Kernel Methods - Support Vector Learning. MIT press, Cambridge (1999)
4. Larkey, L., Croft, W.B.: Automatic assignment of cie-9 codes to discharge summaries. Technical report, CIIR (1995)
5. Sebastiani, F.: Machine learning in automated text categorization. ACM Computing Surveys 34(1), 1–47 (2002)
6. Vapnik, V.N.: The Nature of Statistical Learning Theory. Springer, Heidelberg (1995)
7. Yang, Y., Liu, X.: A re-examination of text categorization methods. In: Proc. SIGIR 1999, the 22nd ACM Conference on Research and Development in Information Retrieval, Berkeley, USA, August 1999, pp. 42–49 (1999)