ANALYSIS OF TEXT CLASSIFICATION METHODS

Goran Kolevski; Ana Madevska Bogdanova
(goran.kolevski@gmail.com; ana@ii.edu.mk)

Institute of Informatics
Faculty of Natural Sciences and Mathematics
University Ss Cyril and Methodius
Skopje, Macedonia

ABSTRACT

The goal of this paper is to describe the problem of text document classification as well as to analyze some of the most common methods that are being used for its resolution. There are numerous practical examples where according to some provided document corresponding class of it is required to be assigned. There are a lot of documents that are already classified by some expert from the particular field thus the idea of this paper is to show how such classification can be used to train a system to classify newly provided documents.

I. INTRODUCTION

In this paper I will review three methodologies that can be used to solve this problem with certain success. Methodologies that will be reviewed are Naive Bayes classifiers, method of k nearest neighbors and support vector machines scattered through next topics. For every method I will introduce the mathematical model. Moreover I will analyze how these methods can be applied for text classification. I will also describe an example usage for each of these methods on real data collected from internet. All methods are commonly used for text classification purposes and all of them have some success from different view point so there is hardly to tell which one is the best. The goal of this paper is to describe the problem, the methods that can be used and to show how data can be modeled in order to be processed with these method. I will try to interpret the meanings of these methods and their steps in this process and also point out possible problems why they occur and how to avoid them.

II. NAIVE BAYES CLASSIFIER

Naive Bayes classifier is supervised probabilistic method for machine learning. In order to be used for document classification it should be tailored for the specific problem and most important modeled against appropriate vector space.

According to the Naive Bayes classifiers probability of one document belonging to class c is expressed with next equation (1):

$$P(c|d) = P(c)\prod_{1\le k\le n_d} P(t_k|c). \qquad (1)$$

where $P(t_k|c)$ is the conditional probability of term t_k occurring in a document of class c. One more factor beside the words occurring in the document is the independent probability of the class itself.

For the need of this approach text documents should be modeled as vectors that indicate the term occurrence frequencies for terms of given vocabulary. Let $t_1, t_2, \dots, t_{n_d}$ be the vector that describes arbitrary document d. Elements t_k from the vectors are all words from the vocabulary. As an example the sentence „hello world example“ can be translated to ("hello", "world", "example") where $n_d = 3$.

In the case where we should choose one of multiple offered classes than a multinomial Naive Bayes classifier is being used. The goal of such approach is to choose the class with best score i.e. probability:

$$argmax_{c\in C} P(c|d) = argmax_{c\in C} P(c)\prod_{1\le k\le n_d} P(t_k|c). \qquad (2)$$

However there is a problem with the last formula that is a result of many very small values being multiplied. Having in mind that the probability is expressed with a value in the interval [0..1] and the number of words in a document can be in orders of hundreds or thousands of words than in the last formula we are multiplying many such small values that can cause danger of floating point underflow during our calculation. This problem is being resolved with changing previous formula with sum of logarithms of probabilities as in the next formula:

$$argmax_{c\in C} \left[log(P(c)) + \sum_{1\le k\le n_d} log(P(t_k|c))\right] \qquad (3)$$

Probabilities in the previous formula are measure about how good indicator the term is about some class. More common classes have greater probability of being assigned to the document and that is described with the probability of the class. So the formula is just a sum of all indicators about document membership to the class.

$$P(c) = \frac{N_c}{N} \qquad (4)$$

$$P(t|c) = \frac{T_{ct}}{\sum_{t'\in V} T_{ct'}} \qquad (5)$$

What is left in this model is just to specify how we calculate the independent probability of the classes and the conditional probabilities of the terms for these classes. The class probability calculation is simple and is equal to the

frequency of documents that belong to this class divided by the number of documents. On the other hand the conditional probability of the terms is calculated with dividing the number of occurrences of the term t in the whole set of documents of this class, including multiple occurrences, with the number of occurrences of that term in the whole set of documents. This formula awards the term as good indicator if it happens to be more common and more unique for the current class against those terms that are rare for the class or common in the whole set. There is some loss of information with such approach because we ignore the word positioning what for sure has meaning for semantics purpose but for classification it will be acceptable loss.

A. Example: Classification with Naive Bayes classifier

Subject of mine example will be a set of various magazine articles published globally. I'll use a set of about 120 article sources who have already assigned class by the publisher. Class assignation naturally can be done with multiple classes having hierarchical tree structure and furthermore one document can be assigned multiple classes, however such structure won't be the case in this research and can be considered for further research. I have defined flat structure with five classes that will be unnaturally disjunctive. Such classes will be World, Business, Technology, Sport and Life. Collected articles are subject of preprocessing to clear out unnecessary formatting metadata such as html. Disclaimer: this process is wasting useful already structured semantic data but having in mind it's not subject of this research I will ignore all formatted data and use pure text. These instances will be structured as (text,class) tuples and processed afterwards. In the moment of retrieving data I've fetched 2745 articles what would be enough data for this analysis. Data distribution across classes is dependent upon the number of articles published at the moment and the nature of the source, but however the numbers are from the same range.

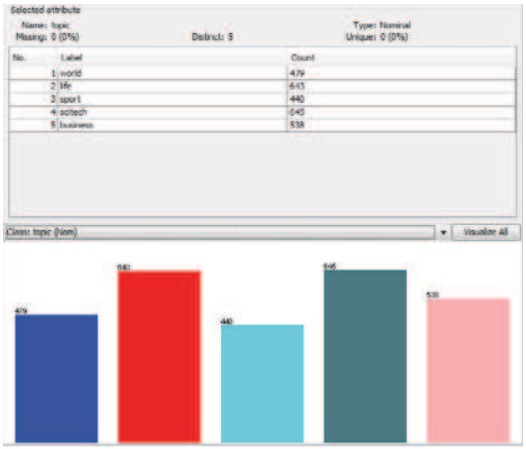

Figure 1: Articles class distribution.

Collected data is cleared out from unnecessary data but it still lacks acceptable formatting for mine research. Naive Bayes method requires term frequencies in given document as well as occurrences in specific classes. With the weka [5] package I'm cleaning the stopwords, filtering out about thousand most common words and stemming them. Important notice is that all this preprocessing steps are losing important grammatical data but anyway the goal of this research is to analyze quantitative classifiers based just on the words. That is why this approach is also called bag of words. After doing Naive Bayes calculations described with formulas (4) and (5) with weka on percentage split 66% to 33% left for validation we can obtain conditional probabilities for retrieved terms for each class. Also independent class probability is calculated as well. For an example

$$P(\text{“world”}) = 479/2745 \approx 0.1745. \qquad (6)$$

There are couple problems with this approach that can be noticed in the list of conditional probabilities. Some of the words are actually numbers, years, prices etc. They might have some value for determination of some class but most probably are useless so they should be considered to be filtered in the preprocessing step next time.

Table 1: Example for conditional probability score for the term "development"

class	word development - score
world	*1.6291951775822744E-4*
life	*4.6253469010175765E-4*
sport	*7.342683016374188E-5*
tech	*0.0011613471627087426*
business	*2.066258006749776E-4*

If we take a look at some particular word as an example "development" we can see that we have very small values for their affiliation to some class. This was one of the problems I've mentioned. However most important fact in this extraction is that technology has highest value for this term what would mean that if the word development shows up next time in the document then most probably it's about technology, but not for sure because it can also belong to some other class.

Table 2: Naive Bayes classifier results

```
...
Time taken to build model: 0.05 seconds

=== Evaluation on test split ===
=== Summary ===

Correctly Classified Instances         715               76.6345 %
Incorrectly Classified Instances       218               23.3655 %
Kappa statistic                          0.7058
```

```
Mean absolute error                0.1036
Root mean squared error            0.2753
Relative absolute error           32.5716 %
Root relative squared error       69.0474 %
Total Number of Instances          933

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.66     0.067      0.669     0.66      0.665      0.903    world
                 0.772    0.084      0.738     0.772     0.754      0.928    life
                 0.911    0.009      0.95      0.911     0.93       0.992    sport
                 0.785    0.056      0.811     0.785     0.798      0.936    scitech
                 0.716    0.079      0.697     0.716     0.706      0.887    business
Weighted Avg.    0.766    0.062      0.768     0.766     0.767      0.927

=== Confusion Matrix ===

   a   b   c   d   e   <-- classified as
 105  18   3   4  29 |   a = world
  14 169   3  19  14 |   b = life
   5   5 133   0   3 |   c = sport
  12  22   0 172  13 |   d = scitech
  21  15   1  17 136 |   e = business
```

This method offers great performances either from speed as well as precision. What I got after this iteration for multiclass Naive Bayes was something above 76% correctly classified instances and time taken to build the model of 0.05seconds. Most interesting entries in the confusion matrix are ones between sport and technology classes saying that none instance from those classes was incorrectly classified in the other class and also world versus business entries having greatest values in the confusion matrix but those two classes are really too similar even for human to classify them. If some of previous comments are taken into consideration for further steps and futher analysis on the data are being made than this precision would probably raise.

III. K Nearest Neighbors

Method of K nearest neighbors is a method with lazy evaluation. The main idea during the training phase is just to save training set instances and be evaluated during decision phase. This is the pure idea of the method, however for performance issues first phase is modified and while training various improvement techniques can be used such as caching and splitting training data across regions for the need of faster calculations. During decision phase class is being determined by calculating a distance from the instance of interest and trained instances. The class is chosen from the majority class of k nearest neighbors.

Question in such defined case is how should we calculate the distance between the documents. Most common metric is Euclidian distance, another one is Hamming distance but however for multidimensional space with many sparse vectors such as document classification problem the metric that is used is cosine similarity of document vectors represented with tf-idf values.

Formally documents can be represented as vectors in high dimensional space such as

$$d_j = (w_{1,j}, w_{2,j}, \dots, w_{t,j}) \tag{7}$$

Every word in the dictionary is determined via one of the vector space dimensions. If the word occurs in the document than corresponding value will be non zero. Simplest way to represent the document is to put one if the word occurs and zero if it doesn't. However better solution is to weight every word according it's frequency in the document with tf-idf representation for each word. Term frequency (TD) value is calculated as

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{4}$$

where $n_{i,j}$ is the number of occurences of the term t_i in the document d_j, and the denominator is the number of all words in the document i.e. $|d_j|$. The other part of tf-idf is the inverse document frequency (IDF) which is calculated:

$$idf_i = \log \frac{|D|}{|\{d : t_i \in d\}|} \tag{8}$$

where |D| is the cardinal number of the D set i.e. the number of training documents and the denominator is the number of documents where the term occurs. TF part values word occurences, what means that most common words for one class will be more valuable and reverse the IDF part will value less the words that are common for many classes which means they are just too common and do not carry enough information for some particular class.

With a vector space prepared like this we can calculate the angle between two documents i.e. the cosine of the angle which would be the same in means of similarity. Since we are interested in the angle distance between these documents its cosine value is just scaled value of the angle. The meaning of this calculation is that the documents that have many similar dimensions (words) will have smaller angles between them. The cosine similarity is calculated with the following formula.

$$cos\ (\alpha) = \frac{d_i\ d_j}{|d_i|\ d_j|} \tag{9}$$

A. *Example: Classification with KNN*

For this example I will use the same set of data as described for Naive Bayes example but I will also have somewhat different approach in its processing. I will introduce tf-idf weights and I will also choose Euclidean distance as a measure instead of cosine similarity I've described. Primary reason is that cosine similarity is not supported by weka at the moment so I will use this to show how bad KNN with Euclidean distance can be compared to other methods. Weka offer multiclass KNN implementation via IBk method thus I will use it for this example. Results

show that this is fast method for training i.e. taking approximately zero seconds but also its worth mentioning that this will be penalized during the phase of making decision since the pure algorithm iterates the dataset, what is actually worse. Enhancement for this issue is available with partitioning the dataset but such approach will take some time during training as an example generating a ball tree required about 28 seconds. What is most important in this analysis is the precision that was not so good in this example with just about 45% correctly classified instances.

Table 3: KNN classifier with Euclidian distance results

```
=== Run information ===

Scheme:       weka.classifiers.lazy.IBk -K 1 -W 0 -A
"weka.core.neighboursearch.LinearNNSearch -A
\"weka.core.EuclideanDistance -R first-last\""
Relation:     tekst-weka.filters.unsupervised.attribute.StringToNominal-
R2-weka.filters.unsupervised.attribute.StringToWordVector-R1-Pzbor--
W1000-prune-rate-1.0-C-T-I-N0-L-S-
stemmerweka.core.stemmers.NullStemmer-M1-
tokenizerweka.core.tokenizers.WordTokenizer -delimiters "
\r\n\t.,;:\'\"()?!"
Instances:    2745
Attributes:   4016
              [list of attributes omitted]
Test mode:    split 66.0% train, remainder test

=== Classifier model (full training set) ===

IB1 instance-based classifier
using 1 nearest neighbour(s) for classification


Time taken to build model: 0 seconds

=== Evaluation on test split ===
=== Summary ===

Correctly Classified Instances         420               45.0161 %
Incorrectly Classified Instances       513               54.9839 %
Kappa statistic                          0.2923
Mean absolute error                      0.2265
Root mean squared error                  0.4153
Relative absolute error                 71.2098 %
Root relative squared error            104.1416 %
Total Number of Instances              933

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area
Class
                 0.27     0.03      0.652     0.27      0.382      0.555    world
                 0.795    0.55      0.307     0.795     0.443      0.691    life
                 0.322    0.004     0.94      0.322     0.48       0.584    sport
                 0.475    0.08      0.646     0.475     0.547      0.742    scitech
                 0.274    0.05      0.584     0.274     0.373      0.545    business
Weighted Avg.    0.45     0.164     0.601     0.45      0.449      0.633

=== Confusion Matrix ===

   a   b   c   d   e   <-- classified as
  43  96   0   8  12 |   a = world
  12 174   0  22  11 |   b = life
   1  86  47   8   4 |   c = sport
   2 101   2 104  10 |   d = scitech
   8 110   1  19  52 |   e = business
```

IV. Support Vector Machines

For two class linear separable set there are many linear separators. The decision border drawn in the middle of the void between the vectors of both classes seems like better classifier candidate than one that is close to border representatives. Some machine learning methods such as the perceptron to name one, are searching for any separator that divides both sets. SVM by definition searches for decision border that maximizes distance from both sets. This distance is called margin. To find the separator only a subset of training instances is required because only boundary vectors are needed to define decision hyper plane. Those vectors are called support vectors thus the name of this method is support vector machines. Margin maximization helps out to make better classification with avoiding to misclassify doubtful instances. Mathematical foundation of SVM can be found in [1][2]

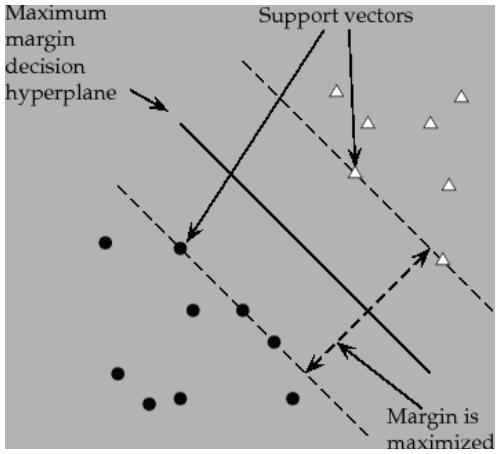

Figure 2: Example showing decision hyperplane, margin and support vectors

However, SVM as described previously requires exactly hyper plane that separates both sets. For text classification purpose especially having in mind high dimensional vector space this is almost impossible. Solution to this problem is to allow some mistakes by introducing slack variables and penalize every misclassification.

Another constraint of described method is multiclass classification as offered solution is just a binary classifier. There are two techniques to override this limitation. First one is with testing membership to each class against all others and choose the class with the best score and the other one is to build one versus one classifiers and test each of them.

A. Example: Classification with multiclass SVM

For this example I will use the same dataset as described for the Naive Bayes case. In the SVM case weka has another

issue. In this case the implementation of the SVM algorithm in weka i.e. SMO is very slow, but there are couple external implementations such as LibSVM. For this example I will use the similar configuration as for the Naive Bayes example, 33% of the dataset left for validation, top 10000 most common words for each class, stopwords filtering etc. From the SVM specific case I'm using the simplest kernel type linear and C-SVC of nu-SVC algorithm since only those are multiclass classification algorithms unlike others which are either binary classification or regression. Results from this experiment are also satisfactory with 71% correctly classified instances what is worse than Naive Bayes but it is in the same range. SVM as a mathematical model is considered as more powerful tool than the rest classifiers so with some calibration can probably be improved. In this experiment the time taken to build the classifier was about four seconds what is slower than Naive Bayes. What can be considered as drawback for SVM classifiers is not the speed but its complexity and can be harder for data model interpretation.

Table 4: LibSVM classifier results

```
=== Run information ===

Scheme:       weka.classifiers.functions.LibSVM -S 0 -K 0 -D 3 -G 0.0 -
R 0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.0010 -P 0.1
Relation:     tekst-weka.filters.unsupervised.attribute.StringToNominal-
Rlast-weka.filters.unsupervised.attribute.StringToWordVector-R1-
Pzbor—W10000-prune-rate-1.0-C-N0-L-S-
stemmerweka.core.stemmers.NullStemmer-M1-
tokenizerweka.core.tokenizers.WordTokenizer -delimiters “
\r\n\t.,;:\'\”()?!”
Instances:    2745
Attributes:   19130
              [list of attributes omitted]
Test mode:    split 66.0% train, remainder test

=== Classifier model (full training set) ===

LibSVM wrapper, original code by Yasser EL-Manzalawy (= WLSVM)

Time taken to build model: 3.98 seconds

=== Evaluation on test split ===
=== Summary ===

Correctly Classified Instances         663               71.0611 %
Incorrectly Classified Instances       270               28.9389 %
Kappa statistic                          0.6349
Mean absolute error                      0.1158
Root mean squared error                  0.3402
Relative absolute error                 36.3871 %
Root relative squared error             85.3235 %
Total Number of Instances              933

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area
Class
                 0.572    0.068      0.632     0.572     0.601      0.752    world
                 0.763    0.136      0.633     0.763     0.692      0.813    life
                 0.829    0.019      0.89      0.829     0.858      0.905    sport
                 0.721    0.07       0.76      0.721     0.74       0.826    scitech
                 0.663    0.074      0.696     0.663     0.679      0.795    business
Weighted Avg.    0.711    0.078      0.715     0.711     0.711      0.816

=== Confusion Matrix ===

   a   b   c   d   e   <-- classified as
  91  33   2   5  28 |   a = world
  13 167   6  25   8 |   b = life
   6  11 121   3   5 |   c = sport
  13  28   6 158  14 |   d = scitech
  21  25   1  17 126 |   e = business
```

V. Summary and Further Research

This text had a goal to describe the problem of text documents classification and analyze some of the methods used for its resolution. Through the document I've made couple oversimplifications on the problem to keep it to the subject that was topic of this paper. First simplification was considering the class structure flat and disjunctive what is obviously not the case and highly hierarchical intersected structure would do the organization of text documents more intuitive. Metadata as well as grammar information was ignored besides it can carry worth information so this should be taken into consideration for research that is more semantically targeted. In depth analysis of provided methods is also available for further research as mine goal was just to model them properly and show how they can be used for text classification. All of the methods are said to be practically used but for different manners. Naive Bayes method is simple, fast and agile enough for incremental system, on the other hand SVM is powerful tool for classification but is somewhat harder to adopt and answer the problems of streaming system. Comparative analysis of these methods is also an interesting topic for research.

References

[1] *Corinna Cortes and V. Vapnik, "Support-Vector Networks", Machine Learning, 20, 1995.*

[2] *Thorsten Joachims, "Transductive Inference for Text Classification using Support Vector Machines", Proceedings of the 1999 International Conference on Machine Learning (ICML 1999), pp. 200-209.*

[3] *Text Classification from Labeled and Unlabeled Documents using EM Kamal Nigam, Andrew Kachites Mccallum, Sebastian Thrun and Tom Mitchell*

[4] *Introduction to Information Retrieval, Christopher D. Manning, Prabhakar Raghavan & Hinrich Schütze Cambridge University Press. 2008.*

[5] *WEKA 3.6 - Machine Learning Toolkit - http://www.cs.waikato.ac.nz/ml/weka/*

[6] *Brin, S. and Page, L. (1998) The Anatomy of a Large-Scale Hypertextual Web Search Engine. In: Seventh International World-Wide Web Conference (WWW 1998), April 14-18, 1998, Brisbane, Australia.*

[7] *Page, Lawrence and Brin, Sergey and Motwani, Rajeev and Winograd, Terry (1999) The PageRank Citation Ranking: Bringing Order to the Web. Technical Report. Stanford InfoLab.*