Final Thesis

Automatic Document classification applied to swedish news

by

Florent Blein

LITH-IDA-EX--05/038--SE
2005-04-05
Titel: Automatic document classification applied to Swedish news

Författare: Florent Blein

Sammanfattning
The first part of this paper presents briefly the ELIN[1] system, an electronic newspaper project. ELIN is a framework that stores news and displays them to the end-user. Such news are formatted using the xml[2] format. The project partner Corren[3] provided ELIN with xml articles, however the format used was not the same. My first task has been to develop a software that converts the news from one xml format (Corren) to another (ELIN).

The second and main part addresses the problem of automatic document classification and tries to find a solution for a specific issue. The goal is to automatically classify news articles from a Swedish newspaper company (Corren) into the IPTC[4] news categories. This work has been carried out by implementing several classification algorithms, testing them and comparing their accuracy with existing software. The training and test documents were 3 weeks of the Corren newspaper that had to be classified into 2 categories. The last tests were run with only one algorithm (Naïve Bayes) over a larger amount of data (7, then 10 weeks) and categories (12) to simulate a more real environment.

The results show that the Naïve Bayes algorithm, although the oldest, was the most accurate in this particular case.

Nyckelord: ELIN, automatic document classification, automatic text classification, Naïve Bayes network, k-Nearest Neighbor, Winnow, Rocchio, Corren
Automatic document classification applied to swedish news

by

Florent Blein

LITH-IDA-EX--05/038--SE

2005-04-05

Supervisor: Ph. D. Jonas Lundberg
Division of Human-Centered Systems, Linköping university

Examiner: Prof. Kjell Ohlsson
Division of Human-Centered Systems, Linköping university
I Abstract

The first part of this paper presents briefly the ELIN\(^1\) system, an electronic newspaper project. ELIN is a framework that stores news and displays them to the end-user. Such news are formatted using the xml\(^2\) format. The project partner Corren\(^3\) provided ELIN with xml articles, however the format used was not the same. My first task has been to develop a software that converts the news from one xml format (Corren) to another (ELIN).

The second and main part addresses the problem of automatic document classification and tries to find a solution for a specific issue. The goal is to automatically classify news articles from a Swedish newspaper company (Corren) into the IPTC\(^4\) news categories. This work has been carried out by implementing several classification algorithms, testing them and comparing their accuracy with existing software. The training and test documents were 3 weeks of the Corren newspaper that had to be classified into 2 categories. The last tests were run with only one algorithm (Naïve Bayes) over a larger amount of data (7, then 10 weeks) and categories (12) to simulate a more real environment.

The results show that the Naïve Bayes algorithm, although the oldest, was the most accurate in this particular case. An issue raised by the results is that feature selection improves speed but can seldom reduce accuracy by removing too many features.

Keywords: ELIN, automatic document classification, automatic text classification, Naïve Bayes network, k-Nearest Neighbor, Winnow, Rocchio, Corren

---

1 Electronic Newspaper Initiative
   http://elin.grupoalamo.com/
2 eXtensible Markup Language
   http://www.w3.org/XML/
3 http://www.corren.se/
4 International Press Telecommunications Council; the IPTC has defined a standard for news categories.
   http://www.iptc.org
II Acknowledgements

I would like to warmly thank my supervisor Jonas Lundberg and my examiner Kjell Ohlsson for their support. They gave me the opportunity to write my thesis in a friendly atmosphere.

I wish to thank the people at IDA who helped me, in particular Håkan Sundblad and Magnus Merkel for sharing their experience on document classification with me; and Mattias Arvola for introducing me to Jonas.

Special thanks to Christian Siefkes from the Berlin-Brandenburg Graduate School on Distributed Information Systems, and to Juan José Garcia Adeva from Sidney School of Electrical and Information Engineering; they gave me some of their time to help me test their software and solve some implementation problems.
III Table of Contents

I Abstract...............................................................................................................................3
II Acknowledgements............................................................................................................5
III Table of Contents.............................................................................................................7
IV Introduction........................................................................................................................11
V XML Converter................................................................................................................13
  V.1 ELIN project.............................................................................................................13
    V.1.1 The ELIN project...............................................................................................13
    V.1.2 The need for news material...............................................................................13
    V.1.3 Corren as a partner.............................................................................................13
  V.2 CELIN, or Corren-ELIN xml converter...................................................................14
    V.2.1 The xml structure...............................................................................................14
    V.2.2 IPTC news IDs...................................................................................................14
    V.2.3 Extractor and Updater........................................................................................14
    V.2.4 Problems encountered........................................................................................17
VI Survey.............................................................................................................................19
  VI.1 What are the different classification algorithms ? ..................................................19
    VI.1.1 Naïve Bayes Network.......................................................................................19
    VI.1.2 kNN...................................................................................................................21
    VI.1.3 Winnow............................................................................................................22
    VI.1.4 Rocchio.............................................................................................................23
    VI.1.5 Trees.................................................................................................................24
    VI.1.6 SVM..................................................................................................................25
  VI.2 Implementations......................................................................................................26
    VI.2.1 TIE Trainable incremental extractor.................................................................26
    VI.2.2 LPU Learning from Positive and Unlabeled examples...................................26
    VI.2.3 WEKA..............................................................................................................26
    VI.2.4 Snow.................................................................................................................27
    VI.2.5 Torch3...............................................................................................................27
    VI.2.6 Awacate............................................................................................................27
    VI.2.7 CRM114...........................................................................................................28
  VI.3 Vocabulary issues....................................................................................................29
    VI.3.1 Feature selection methods................................................................................29
    VI.3.2 Feature weighing methods...............................................................................30
VII Thesis problem..............................................................................................................31
IX.6.2 First test ................................................................. 62
IX.6.3 Second test ............................................................ 62
IX.6.4 Third test ................................................................ 65
IX.6.5 Fourth test ............................................................. 67
X Discussion ..................................................................... 69
  X.1 Choice of algorithms ................................................ 69
  X.2 Choices of tests methods .......................................... 69
  X.3 The limits of my methods ........................................... 70
  X.4 Time issue ............................................................... 70
  X.5 Naïve Bayes problem .............................................. 70
  X.6 Understand torch output and output in general ........... 71
  X.7 Need of text classification ....................................... 72
  X.8 Future work ........................................................... 72
XI Conclusion ................................................................. 73
  XI.1 Classification itself ................................................ 73
  XI.2 Algorithms .......................................................... 73
  XI.3 Use of TC ............................................................. 73
XII References ............................................................... 75
XIII Appendices ............................................................ 77
Linköpings universitet
Institutionen för datavetenskap
Nowadays documents are more and more numerous. With the increasing use of Internet, a lot of documents are created and accessed by more people. This causes a storing problem as well as an indexing one. Although it is emphasized by Internet documents (websites), this is also an important issue among companies and public institutions like universities. There is a need for some tool which would enable users to find the document(s) they are looking for.

On the other side of the issue, there is the problem of actual indexing the documents themselves so they could be retrieved later. Companies such as Reuters[^5] or AFP[^6], which produce a tremendous quantity of documents per day, have a need for software that could index (classify) the newly created documents as efficiently as possible. What does efficiently mean? It means classifying as fast as possible the document in the same category where a human being would have classified this document. For news companies, documents (news articles) need to be classified in one or more news categories. For web search engines like Yahoo!, documents (web pages) need to be classified in one or more Yahoo! category.

The research area that focuses on that problem is called document categorization or document classification. Many algorithms have been developed, and implemented. However, software in general can always be improved; and here specifically because they are somehow dedicated to a certain type of documents, or only accept specific input. It will be shown in the survey section what exactly are the algorithms, how they were implemented in software and later on, how well they perform.

This section describes the ELIN project, why it is needed to convert xml files, and the problems encountered while developing the software.

V.1 ELIN project

V.1.1 The ELIN project

The ELIN\(^7\) project (Electronic newspaper initiative) is a european project whose actors come from Germany, Spain, France and Sweden. Mainly, what ELIN do is provide news to the end-user on different devices. The news can consist of texts, but also pictures, videos and sounds.

The interesting part of this project is the use of the future standard in MPEG\(^8\) content, which means that the videos provided will be reactive to the user input (items in the video are clickable and are links to other news/information). For more information on the videos, see Qiang Liu (2004)\(^1\).

V.1.2 The need for news material

The project is now its final step, currently it is being tested as a whole system (all the modules together). The problem is, ELIN is just a framework, the actual goal of the system is to provide news to the user. Therefore, to test the system, some news are needed to “populate” ELIN.

V.1.3 Corren as a partner

Corren (Östgöta Correspondenten) is one of the various partners of ELIN. It is a Linköping-based newspaper company which has a paper edition as well as a digital one on its website, www.corren.se. Corren joined the ELIN project as a content provider. Corren uses a specific xml structure, which meant that some operations would have to be performed on the xml files in order for them to be used later in ELIN.

My first job was to build a software that would convert Corren xml files into ELIN xml files.

---

7 [http://elin.grupoalamo.com](http://elin.grupoalamo.com)
V.2 CELIN, or Corren-ELIN xml converter

V.2.1 The xml structure

The xml structure used in ELIN is based on the IPTC[9] standard. For each article, there is a main xml news file, and as many xml media files as there were in the original article. What are considered media here are the article text, each article picture (if any) and video (if any).

V.2.2 IPTC news IDs

To differentiate news easily, the IPTC (and Corren) use IDs. There are slightly more than 1200 IPTC categories[10], in that way they can cover any kind of news. Corren does not use the IPTC standard. I had to match by hand Corren categories with the corresponding IPTC categories in order for the news to be classified in ELIN in the same category as they were in the Corren hierarchy.

V.2.3 Extractor and Updater

Basically, whatever it is Corren or ELIN xml files, the documents are still about news. The main idea was to grab information from the Corren file and insert it into a newly created ELIN file (and create as many xml media files as there are media).

To perform this I have used the JAXP[11] API[12] from Sun. This API provide several classes to manipulate xml documents to the “tag level”.

Example for the main title:

```java
public String getMainTitle()
{
    NodeList nodeList = source.getElementsByTagName("label");
    if (nodeList.getLength() == 0)
    {
        return "";
    }
    Node node = nodeList.item(0);
    Node firstChild = node.getFirstChild();
    return getNiceString(firstChild.getNodeValue());
}
```

9 http://www.iptc.org
10 Each category has its own, unique ID
11 Java API for XML Processing http://java.sun.com/xml/jaxp/
12 Application Program Interface, a dedicated set of tools and libraries that makes it easier for programmers to code a specific problem. In this case, the problem is to handle xml files properly
The main title is contained by the <label> tag. If the label tag doesn't exist, the method returns an empty string. Else, it returns the value of the text tag (tags are “nodes” for the API).

```java
String mainTitle = extractor.getMainTitle();
String secondTitle = extractor.getSecondTitle();
String alternativeTitle = extractor.getAlternativeTitle();

// try to fill the titles according to the existing tags
if ( ((secondTitle.equals('""')) || (alternativeTitle.equals('""')) && (!mainTitle.equals('""'))))
{
    if (secondTitle.equals('""'))
        secondTitle = mainTitle;
    if (alternativeTitle.equals('""'))
        alternativeTitle = mainTitle;
}
if ( ((mainTitle.equals('""')) || (alternativeTitle.equals('""')) && (!secondTitle.equals('""'))))
{
    if (mainTitle.equals('""'))
        mainTitle = secondTitle;
    if (alternativeTitle.equals('""'))
        alternativeTitle = secondTitle;
}
if ( ((mainTitle.equals('""')) || (secondTitle.equals('""')) && (!alternativeTitle.equals('""'))))
{
    if (mainTitle.equals('""'))
        mainTitle = alternativeTitle;
    if (secondTitle.equals('""'))
        secondTitle = alternativeTitle;
}

if ((mainTitle.equals('""')) && (secondTitle.equals('""')) && (alternativeTitle.equals('""')))
{
    // all of the tags are missing
    System.out.println("There are no titles associated with this news ", + correnFileName + ",
    exiting...);
    boolean b = new File(correnFileName).delete();
    return;
}
updater.setMainTitle(mainTitle);
updater.setSecondTitle(secondTitle);
updater.setAlternativeTitle(alternativeTitle);
```

This code ensures that none of the 3 titles (main, secondary and alternative) are left blank (like "") but instead that they are filled, at least with the same value.

```java
public void setMainTitle(String title) {
    NodeList n = destination.getElementsByTagName("mpeg7:Title");
    Node mainTitleNode = null;
    for (int i = 0; i < n.getLength(); i++) {
        Node tempNode = n.item(i);
        NamedNodeMap map = tempNode.getAttributes();
        for (int j = 0; j < map.getLength(); j++) {
            Node temp = map.item(j);
            if ((temp.getNodeValue().equals("main")) && (temp.getNodeName().equals("type"))) {
                mainTitleNode = tempNode;
            }
        }
    }
    Node cc = mainTitleNode.getFirstChild();
    if (cc == null) {
        cc = destination.createTextNode(""');
        mainTitleNode.appendChild(cc);
    }
    cc.setNodeValue(title);
}
```

The main title should be inserted in the `<mpeg7:Title>` tag. As all the titles (main, secondary, alternative) go into the same tag, they have to be identified depending on the attributes (type=”main” for main title). When the right node has been found, its content are updated to the value of the string returned by the getMainTitle() method.

This example of **getting/setting the main title** is quite easy to understand, and represents well enough the kind of operations that were necessary.
V.2.4 Problems encountered

V.2.4.a Start of files
The files from Corren were correct xml (according to the Sun's API) except the 13 first lines, which were more like metadata about the document (author, title, time, hash results....). The idea was then to remove these lines before processing the document.

V.2.4.b Media files
Quite often, media files (pictures, videos) were linked to the news articles. The link had to be kept during the conversion.

V.2.4.c Closing of empty xml tags
In html (and xml) tags are closed by rewriting this tag preceded by a /. For example, <title> would be closed by </title>. Empty tags are thus represented as <title/></title>. To avoid this, xml has a simpler method to close empty tags : <title/>. There was doubt on the capability of the ELIN parser to handle such tags, and it was decided to get rid of <.../> and to replace them by <...></...>. The problem was that the JAXP API was automatically using the <.../> closing way. So what has been implemented is that after writing any xml file (news, image, ...) using the standard JAXP method, CELIN open this file again, and check “manually” for tags closed like <.../> and changes it to <...></...>

V.2.4.d No empty important tags
The tags I created were sometimes empty. For example, I get the main title from the “corren” tag <label>. Let's suppose the corren document looks like :
[...]<label></label>[...]
Or even more, that the tag <label> does not exist in the document. Then I would return the empty string “”. And this for titles, authors, ...
In the resulting document this would create such tags as :
<mpeg7:Title type="main"/>
That was not considered a good approach, as the ELIN system performs searches on titles. Having (many) titleless news would be useless for a news system. There was an “empty string” problem for main title, secondary title, alternative title, and keywords. What has been decided was to use whatever main/secondary/alternative title to fill the missing other titles. If all of the 3 titles were empty, then the file was discarded. For keywords, it was decided to use all the words that had their first letter in uppercase, “Like This”. Among those, the words preceded by a “.” were not counted. If there were no keywords then the file was discarded.
V.2.4.e XML-free images

An image problem was encountered when the ELIN system could not display properly the images. It appeared that it was because the images had been post-processed by Corren before they gave them to us. There were several xml lines which were acting as a stamp. The author, time, date and some obscure hash functions results have been written at the top of each .jpg file. These were almost, but not exactly, the same extra lines that were at the beginning of the xml files.
The thing is, ACDSee could display the images properly but it was the only software to do so. Windows's own software, as any browser (Firefox or IE) were unable to open the pictures. The solution that was found was to reduce the image size to 99% of their original size (alter them as little as possible) and that would eventually remove the xml tags.

V.2.4.f Picture renaming

It appeared that some pictures had the same name but were totally different. This could happen because Corren divide their pictures in weeks which may be in different folders. However that would cause a problem in Elin as the files are stored in a single folder. The solution was to have a “hard” link between a picture and the article by naming the picture with the article filename (and some index number). If the article filename was corren_article.xml, after the conversion the picture filenames would be corren_article_1.jpg, corren_article_2.jpg, ...

V.2.4.g Tests of the converter

The converter has been tested as soon as early november. When all the problems mentioned above have been solved (this took a few days) the converter has been used to produce xml files for ELIN. So far it successfully converted more than 3000 files.

V.2.4.h Use of the converter

The converter was used for populating the ELIN final implementation, which was presented during the Net@Home\[^{13}\] show in Nice, France (1st & 2\textsuperscript{nd} December, 2004).

Currently the converter is being used at Corren for importing old news articles into the system.

\[^{13}\] http://www.net-athome.com/
VI Survey

This section lists the main algorithms used in the Text Classification field, explaining some of them in details.

Nowadays a lot of classification algorithms exist. For people interested in the theory lying beneath document classification, this webpage [i] can help define precisely (in mathematical terms) what a category is, with some examples of its property.

VI.1 What are the different classification algorithms?

This webpage [i] sums up these algorithms with their advantages and drawbacks.

VI.1.1 Naïve Bayes Network

The Bayes network approach [ii] is based on probabilities. But actually there are 2 methods, both qualified as “naïve” which relies on the Bayes algorithm. The difference - explained by McCallum & Nigam[iii] (1998) – is that one is based on the multi-variate Bernouilli event model whereas the other is using the multinominal event model.

Mainly, the multi-variate Bernouilli model works better with a fixed vocabulary, as it sees the document as a vector of attributes (words in our case); each value assigned to an attribute is either if it is present in the document (value = 1) or if it is not (value = 0). The document itself is an event, characterized by the presence or absence of the attributes. As well, the classes are viewed as the same vector but with differents values for the attributes, every class containing the same number of attributes. Therefore here the word frequency information is lost; as well as the information of where in the document the word was (word order).

The second version sees the document as a bag of words, the information of where every word appear in the document is lost but the word frequency is kept. For this algorithm every word is an event. Every class contains a different number of attributes. This works with a dynamic vocabulary (i.e., “all” the words) and therefore fits more our purpose of document classification. In the rest of this report I will refer to “Naïve Bayes network” as the second alternative.

---

14 http://www.iro.umontreal.ca/~nie/IFT6255/Projets/Classification.ppt

This webpage contains another definition of the algorithm
The problem can be viewed as:
- There is a set of classes $C_i \ (i \in [0..n])$.
- A document $D$ needs to be classified in one of the classes $C_i$.

The probability that $D$ is in class $C_k$ is given by the formula:

$$P(C_k / D) = \frac{\left( P(D / C_k) \times P(C_k) \right)}{\left( \sum P(D / C_j) \times P(C_j) \right)}$$

which is known as Bayes' Theorem.

In this formula the denominator is always positive (only made of probabilities) and does not depend on the class $C_k$, that means it always has the same value. Therefore the only value to calculate is the numerator, and the class $C_k$ which gets the highest results as a numerator will be the class that fits the most.

$$\left( P(D / C_k) \times P(C_k) \right)$$

$P(C_k)$ is easy to estimate, because the number of items in $C_k$ is known, as well as the total number of items.

Thus $P(C_k) = N_k / N$ where:
- $N_k$ = number of documents in class $k$
- $N$ = number of documents in training set

$P(D / C_k)$ is harder to compute, that is where the “naïve” part comes into play, it is assumed that there is no dependence between the consecutives words in document $D$. Therefore,

$$P(D / C_k) = \prod P(W_t / C_k)$$

where $W_t$ are all the words (or features, or attributes) in $D$.

A (easier) way to calculate the product of all words probabilities is to use the logarithm. The property that is used is: $\log(AB) = \log(A) + \log(B)$

Then:

$$\log(P(D / C_k)) = \log(\prod P(W_t / C_k)) = \sum \log(P(W_t / C_k))$$
VI.1.2 kNN
This algorithm is the k-nearest neighbor \([iv]\) invented by Yiming Yang\[16\]. The algorithm computes the relatedness of the document D with all the documents in the training set (for which the categories are known), then for the k “nearest” documents, if a large majority of them have been classified in the same category C then the document D is also classified in category C.
If k is set to k = size(training set) and if “large majority” is small enough then D will be classified in the category which contains the most documents.

The drawing above shows that :
- if k=3 (smaller circle) the document will be classified in category C
- if k=7 (bigger circle) the document will be classified in category D
What is understood from this is that k should be big enough to consider multiple documents, but not too big not to lose accuracy. The problem of “what does it mean for a document to be a neighbor of another document?” remains to be solved.
It's up to the “implementer” to solve this problem, that means the knn algorithm can classify up to “anything digital” on a computer, provided there are rules for defining resemblance between the different items to classify.
A basic rule for solving this problem is to say that each word in the vocabulary represents a dimension in space. Therefore a document (collection of words) is a n-dimension (n = size of vocabulary) vector in that world, the component “value” of each dimension being the number of times the word appears in the document.

\[16\] http://www-2.cs.cmu.edu/~yiming/
VI.1.3 Winnow

Winnow \[{vi}\][{vii}] uses term-weights to classify documents. In its basic version it is limited to 2 classes, the document D will either go in class C or class C'. There are 2 variants of this algorithm, Positive Winnow and Balanced Winnow.

For Positive Winnow, basically there is a weight \( w_j \) associated to every term \( t_j \) for a given class C. Then the sum of all the weights of each term in document D has to be calculated (sum of all \( w_j * t_j \)) and if that sum is above a threshold \( \theta \), then D is classified into C, else it goes into C'.

If D was pertinent and rejected (i.e. It should have be classified in C and was classified in C') then \( w_j \) is increased by doing \( w_j = w_j * \alpha \), else if D was not pertinent and accepted, then \( w_j \) is decreased as \( w_j = w_j * \beta \). And this, for all the terms that were in D (\( \alpha > 1 \) and \( 0 < \beta < 1 \) are here to promote a term by increasing its weight, or demoting it).

In Balanced Winnow there are 2 weights \( w_{1j} \) & \( w_{2j} \) for each term \( t_j \). The final weight of \( t_j \) is obtained by \( w_j = w_{1j} - w_{2j} \). In case of a misclassification (false positive) \( w_{1j} \) becomes \( w_{1j} * \alpha \) and \( w_{2j} = w_{2j} * \beta \); in the opposite case \( w_{1j} = w_{1j} * \beta \) and \( w_{2j} = w_{2j} * \alpha \).

Now, what if a word \( w \) is in document D but not in any class ? Its weight(s) would be 0 so the word will be discarded. And it is what should happen, as if D contains only news words (according to the classes) then the algorithm has strictly no clue about in which category to classify D.

This approach is limited to 2 classes, however in reality there are often more than 2 classes. An improved, multi-class version of winnow is SNoW, Sparse Network of Winnow. It uses one winnow algorithm per class (i.e. if the summed weight \( W \) of all the terms in D is greater than \( \theta \) then D is marked as “fitting” class C). Then it computes all the positive results (finds the highest of all the \( W \)) and classify D in the correct category.

J.G. Beney — C.H.A. Koster (2003) \[{viii}\] have worked on classifying the patents applications for the European patent agency in Rijswijk, Holland. Patent applications are big documents (more than 5000 words) and are documents of rich vocabulary. Their work was based on Winnow and in their paper they come to the conclusion that Winnow (and SVM) are better algorithms for big texts whereas Rocchio or kNN are preferably used for smaller texts.
VI.1.4 Rocchio

The Rocchio algorithm[^x] has been developed in the 60s. At the beginning it was designed to refine queries in the IR (information retrieval) field. But soon it appeared that it was efficient in document classification as well.

\[
\tilde{Q}_{new} = \alpha \bar{Q}_{orig} + \beta \frac{1}{R} \sum_{d \in Rel} \bar{d} - \gamma \frac{1}{N-R} \sum_{d \not\in Rel} \bar{d}
\]

This equation is taken from[^x].

The Rocchio algorithm was developed long before kNN[^xi], however they look like each other in the way that they both convert features in plans and documents in vectors based on these plans. Let's set D as the document to classify. Where kNN computed a vector for each document in the training set, Rocchio computes the average vector for each category. D will be classified in the category where the vector is the closest to the D vector. The closeness is calculated with the cosine of the two vectors (thus, the closer they are, the greater the cosine).

In the equation above, Qorig is null as there is no original query when the algorithm is used in document classification (however there is, when used in information retrieval). Speaking, I would say that in document classification the algorithm answers the open question “to which category belongs this document?” (economy, sport, ...) and in information retrieval it answers the closed question “does this document fit my request Qorig?” (yes/no).

Therefore, the parameter \( \alpha \) does not play any role here; \( \beta \) and \( \gamma \) are the most important parameters.

The problem with Rocchio is that this algorithm is not very efficient for disjoint categories. For example, if the “sport” category contains n documents where n-2 talk about hockey and 2 talk about tennis; then the average vector of “sport” will tend greatly towards hockey; and the document D talking about tennis could be classified in another category than “sport”. However for joint categories this algorithm should perform well.
VI.1.5 Trees

Trees are famous in computer science programming. They are an analogy of nature's trees, they have as well a single root, branches and leaves. Nodes are places from where 2 or more branches are grown (so, the root is a node).

The idea to use trees in documents classification is to have a deterministic path from where every class could fit (the tree root) to where only one class is left (one of the various tree leaves). Trees can be represented easily on paper, so you can follow a tree “by hand” to understand how the algorithm managed to put document D in class C. Each leave of the tree would then be a category.

In our case, it is not about a specific “algorithm” but more about a set of rules for each node (where 2 or more branches start) to determine to which branch document D belongs. A tree is built by taking the word with most information (regarding to the information-measuring algorithm chosen, see below feature selection/weighing methods) and then is growing downwards to the lesser-information words until it reaches an end via a leaf, i.e. a category.

VI.1.5.a C4.5

C4.5 [17] is a famous instance of a tree. It has been developped by Ross Quinlan in the early 90's to improve the existing ID3 algorithm, by adding this features [18]:

- Avoiding overfitting the data
  - Determining how deeply to grow a decision tree
- Reduced error pruning
- Rule post-pruning
- Handling continuous attributes
- Choosing an appropriate attribute selection measure
- Handling training data with missing attribute values
- Handling attributes with differing costs
- Improving computational efficiency

Trees could be interesting but in our case these algorithms are maybe not the best choice as each node “tests” a document attribute. For simple examples (fixed vocabulary, few different possibilities...) they work quite good, but for entire texts that would mean having at most as many nodes as there are words in the text. Furthermore, many trees used in document classification are binary trees (just work for 2 categories).

VI.1.6 SVM

The Support Vector Machine algorithm has been developed by V. N. Vapnik\textsuperscript{xii} and is one of the most recent and more powerful algorithms used in text classification. It has first been applied by T. Joachims \textsuperscript{xiii} when he developed the software SVM\textit{light}. Basically the idea is to find a “separator” that would divide the positive examples from the negative examples by the largest possible margin. This separator is, from a geometrical perspective, a n-dimensions surface and documents are points in an n-dimensions space (n be the feature number, ...).

For more information on the mathematical expressions, see \textsuperscript{19}.

\textsuperscript{19} \url{http://www.cs.cornell.edu/People/tj/publications/joachims_98c.pdf}
VI.2 Implementations

VI.2.1 TIE Trainable incremental extractor
This software[^20] has been designed for information retrieval in the first place, but it can classify texts using the Winnow algorithm. At the beginning this software learnt all the time; i.e. during the training phase and during the testing phase. After email conversation with its author, this automatic learning behavior was still active but could be disabled; that is a nice improvement for document classification.

VI.2.2 LPU Learning from Positive and Unlabeled examples
LPU[^21] is based on the SVM algorithm. It uses the Thorsten Joachims' implementation of SVM[^xiv], SVMLight[^22]. LPU process is divided into 2 steps:
- training the algorithm (with positive and negative examples)
- finding the best instance of a classification algorithm (multiple iterations, 1 for each instance)
The main theory behind this software is explained in details in B.Liu, Y Dai, X.L. Wee Sun Lee, P.S. Yu (2003)[^xv]. The main drawback of LPU, as for other algorithms, is that it uses a binary class categorization (like Winnow).

VI.2.3 WEKA
Weka[^23][^xvi] stands for Waikato Environment for Knowledge Analysis. It is said that WEKA is one of the best software for machine learning and data mining. The part that is the most related to this thesis is the text classification aspect. WEKA offers a lot of classifiers (more than 30, among them are different BayesNetwork implementations, trees, rule-based classifiers, Winnow, C4.5, etc)
The problem with WEKA is that it requires a specific input, that is, an “.arff” file which regroups all documents and the values of their fixed number of attributes along with the real class they belong to. This could work in a development environment but it is not applicable easily in this thesis as there are texts of different lengths. Therefore the tests of WEKA were not pushed further.

[^20]: http://www.inf.fu-berlin.de/inst/ag-db/software/ties/
Developped by Christian Siefkes of Berlin-Brandenburg Graduate School on Distributed Information Systems
[^21]: The software can be found at: http://www.cs.uic.edu/~liub/LPU/LPU-download.html
[^22]: http://svmlight.joachims.org/
[^23]: http://www.cs.waikato.ac.nz/~ml/
VI.2.4 Snow
Sparse Network of Winnow \(^{24}\)\(^{[xvii]}\) was developed by several researchers at University of Illinois's cognitive computation group. This software solves the problem of binary classes in Winnow. In its basic version, Winnow can either classify a document in a class C, or not classify it as C (i.e., classify it as C'). This does not cause much problem for a 2-classes categorization process, but what if there are more than 2 categories? Snow works as follow: it assigns a unique “instance” of Winnow to a particular class. Then there are as many instances as there are categories. What the program does is assigning a score to each category (given by the algorithm “attached” to that category) and the highest score (fittest category) wins, the document is classified in it. Technically speaking SNoW uses the positive Winnow version (i.e. 1 weight for each feature as opposed to balanced Winnow where there are 2 weights for each feature).
Another interesting fact about SNoW is that its authors have implemented two algorithms other than Winnow: Perceptron and Naïve Bayes.

VI.2.5 Torch3
Torch \(^{25}\)\(^{[xviii]}\) is rather a library than a plain software. It has been developed by three researchers at IDIAP, Switzerland, and is still in a development phase (they are planning to release Torch4). Although it is library-like, Torch comes with already-coded algorithms that are ready to work. Among them are SVM, kNN, and even a speech decoder.
As it is a library, Torch has not been designed to support a lot of user interactions, and produces minimal output.

VI.2.6 Awacate
This software \(^{26}\)\(^{[xix]}\) is developped by the WEG engineering group\(^{26}\) at the university of Sidney. It is a framework that implements various algorithms (kNN, Rocchio, ...) and that is designed for performance and integration. Awacate is still in development phase, thus testing documents require the hard-coding of directories, method,...

\(^{24}\) http://l2r.cs.uiuc.edu/~danr/snow.html
\(^{25}\) http://www.torch.ch/
\(^{26}\) http://www.weg.ee.usyd.edu.au/people/jjga/awacate/
VI.2.7 CRM114

The Controlable Regex Mutilator concept \cite{27} is a project run by W. S. Yerazunis. It is not dedicated towards text classification but instead towards spam filtering. For this reason, this program can not classify news (at least more than 2 categories of news) and has not been tested in this thesis. However I think it is worth mentioning this project for various reasons:

- this software is known for filtering spam very quickly and very accurately (99.984% of correctness)
- it can be set to use some Bayes/Markov theorems which show that these algorithms are widely used

\footnote{http://crm114.sourceforge.net/}
VI.3 Vocabulary issues

In TC, the vocabulary is the set of all the features: the bigger the training set, the bigger the vocabulary. Although having a big vocabulary enables the classifier to recognize more features, it also increases the memory usage and processing time.

In order to have a low memory usage and a short processing time, the vocabulary should be reduced to the features that represent the most their respective categories.

VI.3.1 Feature selection methods

Feature selection methods are ways to keep the vocabulary low by removing:
- words that appear too many times
- words that appear too few times
- ...

Some of these methods are so obvious/often used that it is impossible to trace back the author(s) of such methods.

VI.3.1.a Mutual information

This technique has been developed by [xx] in 1993 by Quinlan. It computes the relatedness between a word and a concept, which - applied to text classification – means finding how well a word is related to a category. This enables any classification algorithm to work on the n “most representative” words for each category, helping reduce its calculation time and improve its performance.

\[
I(T, w) = E(T) - E(T|w) \\
= - \sum_{C \in C} \Pr(T(d) = C) \cdot \log \Pr(T(d) = C) \\
+ \sum_{C \in C} \Pr(T(d) = C, w = 0) \cdot \log \Pr(T(d) = C|w = 0) \\
+ \sum_{C \in C} \Pr(T(d) = C, w = 1) \cdot \log \Pr(T(d) = C|w = 1)
\]

(This equation and its explanation have been taken from “A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization” by Thorsten Joachims, 1996)
Pr(T(d) = C) is the probability than an arbitrary article \(d\) is in category \(C \in \mathbf{C}\). \(Pr(T(d) = C, w = 1)\) and \(Pr(T(d) = C, w = 0)\) are the probabilities that article \(d\) is in category \(C\) and it does or does not contain word \(w\). \(Pr(T(d) = C \mid w = 0)\) and \(Pr(T(d) = C, w = 1)\) are defined similarly.

**VI.3.1.b Term Frequency**

Basically, the TF computes the number of times a feature appears in a document \(d\).

**VI.3.1.c Document Frequency**

For a feature, the DF is the number of documents in which this feature appears.

**VI.3.1.d Inverse Document Frequency**

It is based on the DF. The formula that gives the IDF for a feature is:

\[
IDF(w) = \log \left( \frac{|D|}{DF(w)} \right)
\]

It balances the DF with the number of documents in the training set. The more a word appears, the less information it is supposed to have, unless there are a lot of documents. This is why the number of documents \(|D|\) had to be introduced in the formula.

**VI.3.1.e TF/IDF**

The formula of TF/IDF is the following:

\[
TF/IDF(w, d) = TF(w, d) \times IDF(w)
\]

This equations gives a weight which has to be computed for each feature in each document \(d\) in the training set. Then the final weigh for each feature is given; and the \(n\) most important words are kept.

**VI.3.2 Feature weighing methods**

It can be a good idea to assign a weight to each feature if some features are more important than others. As feature selection techniques grade the features (by giving a weight to each of them) they can also be used as feature weighing.
VII Thesis problem

The goal of this thesis is to provide a software which will be able to classify news articles automatically in categories.

Some research has been carried out before on Reuters news articles ([xix], [xxii] and various others), but these works has been done on documents written in the english language. So far there has not been any research on swedish documents. It is of scientific interest to know how the existing algorithms behave with non-english texts.

The input available for this thesis is itself not common, as it only consists of 2 categories with some (~750) short documents (~350 words/doc). Previous research has concentrated on longer documents (xxiii). A lot of existing classification software work with preprocessed input (numbers instead of words) and it is up to the end-user to convert his document into the proprietary input format of the software. It would be easier for users to just give their documents as input and not have to worry about modifying them. Such easy software are not so common.

Furthermore, trying to classify news articles is hard because 2 articles about the same subject “foreign policy” can talk about really different matters (Irak war, WTO problems, weather catastrophes) so the words are totally different. Does that mean the documents will be harder to classify? As well, newspapers receive articles from different sources and even if these documents have a “category tag”, it may not be the same classification system as the newspaper uses internally. Classifying documents is not a very interesting work and a classification software could help reduce the time needed, if not by classifying 100% of the documents, then by giving clues and guesses to the human classifier.
Last, the archive of a newspaper might not have been classified, there is a need for such software to classify the articles (for example when they want to digitalize their paper archive).

Thus, the goals of this thesis are in the end to:

- propose a software that could help people in Corren to classify *automatically* and *with a high accuracy percentage* a non-classified document that they receive/produce
- propose a solution to people working on ELIN to get unclassified content (i.e. old Corren articles) and automatically classify it
VIII Method

This section describes how the algorithms have performed.

VIII.1 Idea

To achieve the goals, the following steps were:

- implement 4 algorithms described in the survey
- test and compare them with existing implementations (only with 2 categories)
- use the best of the 4 algorithms and try it on all existing categories in Corren

These algorithms are:

- Naïve Bayes network
- k-nearest neighbor
- Winnow
- Rocchio

VIII.2 Content description

As I said, the purpose of this thesis is to classify documents. At the beginning Corren provided us with 3 weeks (week 40-42) which were composed of xml files and media files (jpg pictures, text included in the xml files). The first problem was to extract the texts into separate .txt files. Fortunately what had been done for CELIN could be re-used, for instance the Extractor class (and the getText() method).

On overall there were 4709 xml files. After removing files with the same filters used in CELIN (the files without any IPTC ID, those without keywords, etc...) only 1840 text files remained, still separated into 3 weeks.

VIII.3 Categories description: why just 2 categories?

The categories which are used here are the IPTC news categories. There are a bit more than 1200 IPTC categories in total; and there are 71 Corren categories. Only 34 Corren categories could be matched with 40 IPTC categories. Among these 40 categories, only 19 contained documents.

Among those 19 IPTC categories, several were containing only few documents, so it was decided not to use these categories for the training/testing of the software. With a threshold set to 1024 bytes of data, the documents contained enough texts to be used in a classification process.
However half of the documents were divided into 2 categories (which were containing 763/1516 documents) whereas the other documents were divided into 17 categories. These latter categories did not contain enough documents to be taken into consideration, therefore the tests have been run on the 2 most populated categories. These categories are 15000000 Sport and 04017000 Economy (general).

In other classification researches the training set was twice as big as the test set (training set = 2/3 of all documents). In order to “comply” to this de facto rule, it was decided to divide the training and the testing set as follow :

- Week 40+41 as training set (518 files, 130432 words, 251 words/doc)
  - Economy = 250 files
  - Sport = 268 files
- Week 42 as test set (245 files, 89760 words, 366 words/doc)
  - Economy = 119 files
  - Sport = 126 files

VIII.3.1 Features representation/selection

VIII.3.1.a Stop words

So far it is agreed that the classification process relies only on endogenous knowledge (the document content). In this thesis the work has been done on text document, which means the knowledge is represented by the words themselves.

In many languages some words contain more information than others. For example, in the expression “the raindeer”, raindeer gives the reader an information (it represents an animal) but “the” has little information to give.

Another example :

- Economy: the stockmarket prices are decreasing
- Sport: the hockey final will see Sweden play against Finland

In these sentences, “the”, “are”, “the”, “will” are words that do not add extra knowledge about the category in which the sentences should be classified. Such words are called “stop words”.

It is necessary to build a “stop word list” in order to:

- reduce the number of words to be processed / increase the time performance
- optimize the ratio information/number of words
I have created a stop word list (see appendix I) and used it to filter out any “low-information” word. Due to my lack of knowledge of the swedish language at the beginning of the thesis, this list contains the most common words in swedish. During my work I have been in contact with more stop words but I did not want to add them to the stop word list; if I had done so the tests would have been different for each algorithm.

**VIII.3.1.b Stemming**

Stemming is the process of extracting the root of a word. For example if the sentence “it appears that stemming is important” is stemmed, the result will most likely be “it appear that stem be important”.

I did not apply any stemming techniques to the texts because of my lack of knowledge of the swedish language. That means that for example “appear” and “appears” count as different words for TC.

**VIII.4 Vocabulary**

Before going further it would be good to introduce some specific terms of document classification.

**VIII.4.1 Recall**

\[
\text{Recall} = R = \frac{\left( \text{documents found} \land \text{correct} \right)}{\left( \text{total documents} \in \text{the class} \right)}
\]

This percentage is commonly called recall, but the “recognition” ability of the software would be a better explanation of what it represents (measuring its performance in assigning a category to each document).

**VIII.4.2 Precision**

\[
\text{Precision} = P = \frac{\left( \text{documents found} \land \text{correct} \right)}{\left( \text{total documents found} \right)}
\]

This percentage is easier to understand, it represents how well the software classifier has succeeded matching the input documents into their categories (in the end, the fewer non-sport documents in the sport category, the higher the precision).
VIII.4.3 Correct classification

<table>
<thead>
<tr>
<th>Human expert</th>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine classifier</td>
<td>Aj</td>
<td>Bj</td>
</tr>
<tr>
<td>YES</td>
<td>Cj</td>
<td>Dj</td>
</tr>
</tbody>
</table>

This table represents the 4 cases that can occur during a document classification. It is assumed that the human expert is always right. In that case, the correct classifications are when the machine classifies a document like the human expert would. This leads to:

- Correct states: Aj, Dj, the machine agrees with the human
- Incorrect states: Bj, Cj, the machine disagrees and thus misclassified the document

VIII.4.4 Error percentage

No matter:
- how good an algorithm can perform
- how well chosen a training set can be

There will always a possibility for some misclassification from the machine. It is impossible to achieve 100% of correct classification on random documents. But that does not mean it is impossible to classify documents at all.
What is considered a good performance? An algorithm has to classify correctly at least \( \frac{x}{100} \) of the documents where \( x = \frac{100}{\text{number of categories}} \), or else it would just be better to distribute the documents randomly.

This work is being carried out on 2 categories so the algorithms have to classify at least 50% of the documents correctly.

VIII.4.4.a First test

A document will be represented by a bag of words:
- lose word order
- do not do stemming on words

VIII.4.4.b Second test

This test is the same as the other except that every word is converted to a number by summing the ascii codes of the letters.
VIII.5 My software: TextClassifier

The name of the software I have built is TextClassifier, or TC in short. I have tried to achieve several objectives with this software.

VIII.5.1 Implementing algorithms

I have tried to implement some algorithms in my text classification software. The reason was mainly to have my own software with “working” algorithms. This implementation could be tested with the training & test sets to check if I was implementing these algorithms in the correct way. An implicit reason was also to see if I understood the algorithms well enough to be able to use them in a real project.

VIII.5.2 Compare the classification accuracy with existing classification software

Another reason why I have built a software was to compare its classification skills with existing classification software. For each algorithm that I have implemented, I will compare its results with at least one implementation of the same algorithm.

VIII.5.2.a Naïve Bayes in TC

Naïve Bayes network are either based on the “multinominal” model or on the “multi-variate Bernouilli” model. As the reader recalls, I will use the multi-variate Bernouilli model because of its capacity to perform with a dynamic vocabulary (there can be any word of any language in the documents).

Bayes network provide a probability per category about a specific document. Therefore the category with the highest percentage is the one where the document D will be classified by the algorithm.

This percentage is (see survey for more details):

\[
P(D/Ck) = \prod P(Wt/Ck)
\]

\[
\log(P(D/Ck)) = \sum \log(P(Wt/Ck))
\]

Now the problem is to calculate the probability of each word “happening”, for each category Ck. But what if a word is not in Ck, then should its probability be 0 ? This should not happen, as if in D (the document to classify) there is a unknown word, let's say the first name “Karl”, then if Karl has never been seen before by any of the categories during the training session, then all of the categories will have a probability of 0 for this word. The product of the probabilities would be 0 as well and D would not be classified.
In order to compute \( P(W_t | C_k) \) and the whole \( P(D | C_k) \) I have followed Weka's implementation of the Naïve Bayes algorithm. That means I have looked at its code and used (& changed) some of the methods of the naïveBayes class. I was able to do so as Weka comes with a GPL license.

This had lead me to:

\[
\log(P(W_t / C_k)) = \log\left(\frac{N_{tk} + 1}{N_k}\right) \quad \text{when } W_t \text{ exists in } C_k
\]

where:
- \( N_{tk} \) is the frequency word \( W_t \) appears in category \( C_k \)
- \( N_k \) is the the number of words in \( C_k \)

and

\[
\log(P(W_t / C_k)) = \log\left(\frac{1}{N}\right) \quad \text{when } W_t \text{ does not exist in } C_k
\]

where:
- \( N \) is the number of words in all \( C_k \)

With this simple way of calculating \( P(W_t | C_k) \) it was then easy to calculate the probability for each category for document \( D \).

**VIII.5.2.b Opponent : Awacate**

Awacate is still in development, so what I did was creating my own java class in the framework and use the categoriser (Rocchio, kNN, ...) provided by Awacate over my training/test sets. Nothing more was necessary to take advantage of Awacate features.

**VIII.5.2.c kNN in TC**

The second algorithm I have implemented is the k-nearest neighbor. Its implementation seemed easy enough to achieve it after the Naïve Bayes.

The survey showed that the kNN algorithm needs to be adapted to the nature of the objects to be classified. Here I have taken the classic n-dimension vector \((n = \text{number of words in the document})\) representation, with each word value being the number of times that word appears in the document.

First, I have to calculate the vector \( V_d \) for a document \( D \). Once this vector is created, the algorithm has to calculate the vector \( V_i \) for each document \( D_i \) in each category; and then compare it with \( V_d \).
The comparison is done via the following:
- for each document \( D_i \)
  - the difference between \( D \) and \( D_i \) is 0
  - for each word \( w \) in document \( D \)
    - if \( w \) does not appear in document \( D_i \), increase the difference by 1

In the end the difference is compared to the “already k-nearest” \( \{ D_a, D_b, \ldots, D_k \} \) neighbors, and if the difference is lower than the difference of one of the \( \{ D_a, \ldots, D_k \} \), then \( D_i \) is inserted and the farthest \( \{ D_a, \ldots, D_k \} \) is removed from the list.

Comparing \( V_d \) with each \( V_i \) builds the knowledge of which documents are the k-nearest documents to \( D \). The next step is to know to which categories belong the k-nearest documents. From this information, it is easy to get the categorie(s) which “is/are the most likely to contain” \( D \). If there are 2 documents belonging to category \( C \) and 5 documents belonging to category \( C' \) (\( k = 7 \)), \( D \) will be tagged as belonging (the most likely) to category \( C' \).

It is easily understood that it is a better choice to make \( k \) odd rather than even.

**Exact number of neighbors?**
Suppose that \( k \) has been set to 5. Once the algorithm has found the 5 closest neighbors, should it stop here? What about the other documents \( D_k \) that are “as far” (or “as close”) as the farthest 5\(^{th} \) document \( D_5 \)? Why count \( D_5 \) and not all the \( D_k \)? As they are all on the farthest circle (the center being \( D \)) they should be counted as neighbors as well.

What is done in TextClassifier is that, after having found the k-nearest neighbors, there is a re-check (and adding) of all the documents which were at the same distance to \( V_d \) as the farthest document. In order to accelerate this post-check, the software only checks the files that could have been a nearest neighbor.

Example: if
- \( k = 1 \)
- test set = \{ \( D \) \}
- training set = \{ \( D_a, D_b \) \}

Let's say the distance (\( D, D_a \)) is equal to 10, then \( D_a \) is assigned as a nearest neighbor to \( D \). Then if distance (\( D, D_b \)) is equal to 10 as well, distance (\( D, D_b \)) is not lower than distance (\( D, D_a \)) so \( D_b \) can not be a nearest neighbor. But what if \( D_b \) had been compared before \( D_a \)? My software keeps track of all the \( D_b \)-like documents, and adds them as nearest neighbor after the algorithm has finished. Therefore there are “at least” \( k \), and often more, neighbors.
VIII.5.2.d Opponent: Torch3

Torch uses 1 file for training the algorithm and 1 file to test it. As usual these files have to have a specific syntax:

Number of files to train/test number of features+1

feat.1 feat.2 feat.3 [....] feat.n-1 feat.n category_of_file_1
feat.1 feat.2 feat.3 [....] feat.n-1 feat.n category_of_file_2
feat.1 feat.2 feat.3 [....] feat.n-1 feat.n category_of_file_3

.....

If feat.x is present then I use the ascii code of the word, if not then it is replaced by 0.
The output file consists of just 1 number n, 0 < n < 1. It seems that this number is the total percentage of correctly classified files28.

VIII.5.2.e Winnow in TC

The third algorithm was the Winnow algorithm. As you may recall, Winnow comes in two flavors, Positive Winnow and Balanced Winnow. Since Balanced Winnow seemed to be the most “advanced” I have chosen to implement this one rather than program Positive Winnow.

Winnow needs to be trained only once (whereas for example kNN needs to be trained for each file to classify); that means I had to find a way to “remember” the training. This was achieved by just building one object of class WinnowClassifier and using it for training and classifying every file in week 42.

Winnow keeps track of a vocabulary (all the words encountered in the training files) and to each word is assigned a positive weight and a negative weight. In order to classify a file the weights of all the words appearing in the document are summed. If the sum is above a threshold then the document is classified in the category C “attached” to the algorithm, else it is C' (Winnow is a binary classifier). In my tests I have chosen to set the threshold to 0.

I chose to start with an empty vocabulary and build it up by adding new words as soon as they appear. After getting the sum and classifying the document, the list of the words that were not in the vocabulary previously is built; they are added with a standard positive and negative weight.

28 See discussion section for more details
The last step is to update the weights of the words, depending on the classification. Winnow is an error-driven algorithm so it has to know (afterwards) the “real” category of the document:

- If the document D had been classified in category C (and it belonged to C') then it is a false positive, the positive weights are demoted and the negative weights are promoted
- If the document D had been classified in category C' (and it belonged to C) then it is a false negative, the positive weights are promoted and the negative weights are demoted
- Else nothing has to be done, the algorithm has functioned properly

The vocabulary keeps track of the number of times a word appear in the document by multiplying x times (if the word appears x times in the document) the current weight by the promotion/demotion attribute. This leads to:

\[ \text{newWeight} = \text{oldWeight} \times \text{attribute}^x \]

I have also made some test without taking into account the word frequency (thus, \[ \text{newWeight} = \text{oldWeight} \times \text{attribute} \]) and it seems that this approach improves the results. At the moment I can not explain it (it seems to be against logic) but there is something similar happening with the Naïve Bayes network algorithm. It assumes that there are no relations between the words (word order, etc...). Although there is a relation for the human mind, the Naïve Bayes algorithm performs good and sometimes better at classifying documents than algorithms that take word order into account. Therefore maybe it is not such a silly idea not to take into account the number of times a word appears.

**VIII.5.2.f Opponent : SNoW**

Here is how to interpret SNoW output: “Our test file contains labeled examples of exactly the same format as those used in testing, and we can just use the default output mode and let SNoW score our accuracy. In this mode, each example is given to the system and the resulting prediction output by the classifier is compared to the example's label. A mistake is scored if the two do not match.”[29].

---

29 “SNoW user guide” by Carlson & Cumby & Rosen & Roth, Cognitive computation group, Urbana university, Illinois, 1999
http://l2r.cs.uiuc.edu/~danr/snow.html#snow-papers
**VIII.5.2.g Rocchio in TC**

Because it uses vectors, the Rocchio algorithm needs a fixed vocabulary to be able to classify documents. In order to get a fixed vocabulary, I have used the “feature frequency” for term selection, with the number of features varying from 100 up to the total number of features.

Other parameters that I have tested are the coefficients $\beta$ and $\gamma$.

Then, after computing the average vector $V_{Ci}$ for each category $Ci$, and the vector $V_D$ of document $D$, the software calculates $\cos(V_{Ci}, V_D)$ for each category $Ci$. $D$ is then classified in the category for which the cosinus is the highest.

**VIII.5.2.h Opponent : Awacate**

Awacate can also be used with the Rocchio algorithm.
IX Results

IX.1 NaïveBayesNetwork

IX.1.1 TextClassifier

<table>
<thead>
<tr>
<th>Results</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ascii codes</td>
<td>49</td>
<td>0.41</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>0.64</td>
<td>175</td>
<td>71.43</td>
<td>&lt; 18s</td>
</tr>
<tr>
<td>words</td>
<td>118</td>
<td>0.99</td>
<td>0.99</td>
<td>125</td>
<td>0.99</td>
<td>0.99</td>
<td>243</td>
<td>99.18</td>
<td>&lt; 1mn49s</td>
</tr>
</tbody>
</table>

IX.1.2 Opponent : SNoW

Here are the results for SNoW, using the NB algorithm:

<table>
<thead>
<tr>
<th>Results</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>78.37</td>
<td>&lt; 2s</td>
</tr>
</tbody>
</table>

Unfortunately SNoW does not give much information about the classification process.

IX.1.3 Opponent : Awacate

<table>
<thead>
<tr>
<th>Results</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>words</td>
<td>117</td>
<td>0.98</td>
<td>0.98</td>
<td>124</td>
<td>0.98</td>
<td>0.98</td>
<td>241</td>
<td>98.37</td>
<td>&lt; 10s</td>
</tr>
</tbody>
</table>

IX.1.4 Winner

TC performs definitely better than SNoW, and is quite fast. However on the other hand Awacate is less accurate but very fast. I would tend to think Awacate suits better here.
IX.2 KNN

IX.2.1 TextClassifier

As I said, the size of the training set is 518 files. I started with setting k to 7.

Using ascii code

<table>
<thead>
<tr>
<th>Category</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>K= 7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>80</td>
<td>0.67</td>
<td>0.86</td>
<td>113</td>
<td>0.9</td>
<td>0.74</td>
<td>193</td>
<td>78.78</td>
<td>&lt; 2mn13s</td>
</tr>
<tr>
<td>25</td>
<td>72</td>
<td>0.61</td>
<td>0.92</td>
<td>120</td>
<td>0.95</td>
<td>0.72</td>
<td>192</td>
<td>78.37</td>
<td>&lt; 2mn13s</td>
</tr>
<tr>
<td>51</td>
<td>58</td>
<td>0.49</td>
<td>0.95</td>
<td>123</td>
<td>0.98</td>
<td>0.67</td>
<td>181</td>
<td>73.88</td>
<td>&lt; 2mn13s</td>
</tr>
<tr>
<td>101</td>
<td>47</td>
<td>0.39</td>
<td>0.92</td>
<td>122</td>
<td>0.97</td>
<td>0.63</td>
<td>169</td>
<td>68.98</td>
<td>&lt; 2mn26s</td>
</tr>
</tbody>
</table>

If words are replaced by numbers, then the classification is fair but could be greatly enhanced.

Using words

<table>
<thead>
<tr>
<th>Category</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>K= 7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>106</td>
<td>0.89</td>
<td>0.94</td>
<td>119</td>
<td>0.94</td>
<td>0.9</td>
<td>225</td>
<td>91.84</td>
<td>&lt; 2mn53s</td>
</tr>
<tr>
<td>25</td>
<td>107</td>
<td>0.9</td>
<td>0.96</td>
<td>122</td>
<td>0.97</td>
<td>0.91</td>
<td>229</td>
<td>93.47</td>
<td>&lt; 3mn1s</td>
</tr>
<tr>
<td>51</td>
<td>99</td>
<td>0.83</td>
<td>0.97</td>
<td>123</td>
<td>0.98</td>
<td>0.86</td>
<td>222</td>
<td>90.61</td>
<td>&lt; 3mn4s</td>
</tr>
<tr>
<td>101</td>
<td>85</td>
<td>0.75</td>
<td>0.99</td>
<td>125</td>
<td>0.99</td>
<td>0.79</td>
<td>214</td>
<td>87.35</td>
<td>&lt; 3mn15s</td>
</tr>
</tbody>
</table>

The efficiency of this algorithm does not fluctuate much with the values of k. The average efficiency seems to be around 90%, which makes the algorithm interesting. It is interesting to notice a decrease in accuracy when k is increased (both for ascii and words).
 IX.2.2 Opponent: Torch3

<table>
<thead>
<tr>
<th>Category</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>K=</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>107</td>
<td>43.7</td>
</tr>
<tr>
<td>25</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>115</td>
<td>46.9</td>
</tr>
<tr>
<td>51</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>119</td>
<td>48.6</td>
</tr>
<tr>
<td>101</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>119</td>
<td>48.6</td>
</tr>
</tbody>
</table>

Between k=51 and k=101 there is no improvement in the accuracy. The best percentage achieved is less than 50%, which makes Torch3 very fast but not efficient. Torch low results may be due to a misconfiguration of the software (it is not finished, it does not have any end-user manual). However the results above are almost similar to the ones from TC for ascii (around 50%) so it is most likely related to the distribution of the documents in the space.

 IX.2.3 Opponent: Awacate

<table>
<thead>
<tr>
<th>Category</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>K=</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>109</td>
<td>0.92</td>
<td>0.99</td>
<td>125</td>
<td>0.99</td>
<td>0.93</td>
<td>234</td>
<td>95.51</td>
<td>&lt; 43s</td>
</tr>
<tr>
<td>25</td>
<td>103</td>
<td>0.87</td>
<td>0.97</td>
<td>123</td>
<td>0.98</td>
<td>0.88</td>
<td>226</td>
<td>92.24</td>
<td>&lt; 43s</td>
</tr>
<tr>
<td>51</td>
<td>104</td>
<td>0.87</td>
<td>0.99</td>
<td>125</td>
<td>0.99</td>
<td>0.89</td>
<td>229</td>
<td>93.47</td>
<td>&lt; 44s</td>
</tr>
<tr>
<td>101</td>
<td>99</td>
<td>0.83</td>
<td>0.99</td>
<td>125</td>
<td>0.99</td>
<td>0.86</td>
<td>224</td>
<td>91.43</td>
<td>&lt; 43s</td>
</tr>
</tbody>
</table>

The decrease in accuracy with the increase of k is noticeable here as well.

 IX.2.4 Winner

If the end user focuses more on speed then Torch3 should be the first choice, however the user has to keep in mind its poor results (not even 50% of correct classification). Although TC has a better accuracy than Torch3, Awacate should be the first choice as it is the fastest and most reliable software of this comparison.
IX.3 Winnow

IX.3.1 TextClassifier

The Winnow implementation considered here is the Balanced version. The target category, if not explicitly stated, is economy.

The first test was:
- using ascii
- Don't take feature frequency into account
- Random training
- Learn once

The resulting accuracy was poor as shown below:

<table>
<thead>
<tr>
<th>Category</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>99</td>
<td>83</td>
<td>0.78</td>
<td>98</td>
<td>0.78</td>
<td>0.83</td>
<td>197</td>
<td>80.4</td>
<td>&lt; 32s</td>
</tr>
<tr>
<td>A</td>
<td>76</td>
<td>0.64</td>
<td>0.93</td>
<td>120</td>
<td>0.95</td>
<td>0.74</td>
<td>196</td>
<td>80</td>
<td>&lt; 34s</td>
</tr>
<tr>
<td>A</td>
<td>97</td>
<td>0.82</td>
<td>0.91</td>
<td>116</td>
<td>0.92</td>
<td>0.84</td>
<td>213</td>
<td>87</td>
<td>&lt; 43s</td>
</tr>
<tr>
<td>A</td>
<td>84</td>
<td>0.71</td>
<td>0.95</td>
<td>122</td>
<td>0.97</td>
<td>0.78</td>
<td>206</td>
<td>84.1</td>
<td>&lt; 33s</td>
</tr>
<tr>
<td>A</td>
<td>55</td>
<td>0.46</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>0.66</td>
<td>181</td>
<td>73.9</td>
<td>&lt; 30s</td>
</tr>
</tbody>
</table>

The basic word tests started with:
- Use words
- Take frequency into account
- Deterministic training (first training on every economy files, then sport)
- Learn once
The results show that these settings do not lead to a good classification, only slightly above 50%.

The next set of tests had the following parameters:
- Use words
- Don't take word number into account
- Learn once

1) Deterministic training (first training on every economy files, then sport)
The next 3 tests (present in appendix II) show odd results; from which I assume that Winnow (my Winnow, at least) only “remembers” the last training category. Which means if the last category in the training set is C, all the test files would be classified in C. Generally, what can be concluded is that almost all the files in the test set would be classified in the category which was the last one in the training set.

In order to improve the performance it would be interesting to mix the training files (to train Winnow on files chosen randomly over all the training categories). This is what was chosen for the next test:
- Take word number into account
- Random training
- Use words
- Learn once

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Economy</td>
<td>R</td>
</tr>
<tr>
<td>A 71</td>
<td>0.6 0.92</td>
</tr>
<tr>
<td>A 91</td>
<td>0.76 0.82</td>
</tr>
<tr>
<td>A 101</td>
<td>0.85 0.78</td>
</tr>
<tr>
<td>A 79</td>
<td>0.66 0.81</td>
</tr>
<tr>
<td>A 92</td>
<td>0.77 0.81</td>
</tr>
<tr>
<td>A 113</td>
<td>0.95 0.63</td>
</tr>
</tbody>
</table>

A : positive weight = 3, negative weight = 0.5, promotion attribute = 3, demotion attribute = 0.3

Better results are achieved there! That confirms the theory that Winnow remembers the last category, so if there is no “last category” (like here) then the algorithm works better.
This test was the same as above but without taking the word frequency into account:

<table>
<thead>
<tr>
<th>Category</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>105</td>
<td>0.88</td>
<td>0.97</td>
<td>123</td>
<td>0.98</td>
<td>0.90</td>
<td>228</td>
<td>93.1</td>
<td>&lt; 2mn35s</td>
</tr>
<tr>
<td>A</td>
<td>118</td>
<td>0.99</td>
<td>0.96</td>
<td>121</td>
<td>0.96</td>
<td>0.99</td>
<td>239</td>
<td>97.6</td>
<td>&lt; 2mn29s</td>
</tr>
<tr>
<td>A</td>
<td>111</td>
<td>0.93</td>
<td>0.95</td>
<td>120</td>
<td>0.95</td>
<td>0.94</td>
<td>231</td>
<td>94.3</td>
<td>&lt; 3mn6s</td>
</tr>
<tr>
<td>A</td>
<td>115</td>
<td>0.97</td>
<td>0.93</td>
<td>118</td>
<td>0.94</td>
<td>0.98</td>
<td>233</td>
<td>95.1</td>
<td>&lt; 2mn57s</td>
</tr>
<tr>
<td>A</td>
<td>108</td>
<td>0.91</td>
<td>0.96</td>
<td>122</td>
<td>0.97</td>
<td>0.92</td>
<td>230</td>
<td>93.9</td>
<td>&lt; 3mn9s</td>
</tr>
<tr>
<td>A</td>
<td>106</td>
<td>0.89</td>
<td>0.95</td>
<td>121</td>
<td>0.96</td>
<td>0.9</td>
<td>227</td>
<td>92.7</td>
<td>&lt; 2mn52s</td>
</tr>
</tbody>
</table>

A : positive weight = 3, negative weight = 0.5, promotion attribute = 3, demotion attribute = 0.3

This test produced even better results, in a relatively small time. It then seems that taking the word frequency into account decreases the performance of Winnow, whereas it would be logical to count them.

In order to try to improve again the results, it is usual to train the algorithm multiple times over the same training files. Here are the results for 3 passes:

<table>
<thead>
<tr>
<th>Category</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>111</td>
<td>0.93</td>
<td>0.95</td>
<td>120</td>
<td>0.95</td>
<td>0.94</td>
<td>231</td>
<td>94.3</td>
<td>&lt; 5mn50s</td>
</tr>
<tr>
<td>A</td>
<td>110</td>
<td>0.92</td>
<td>0.96</td>
<td>121</td>
<td>0.96</td>
<td>0.93</td>
<td>231</td>
<td>94.3</td>
<td>&lt; 5mn18s</td>
</tr>
<tr>
<td>A</td>
<td>115</td>
<td>0.97</td>
<td>0.97</td>
<td>122</td>
<td>0.97</td>
<td>0.97</td>
<td>237</td>
<td>96.7</td>
<td>&lt; 5mn51s</td>
</tr>
<tr>
<td>A</td>
<td>118</td>
<td>0.99</td>
<td>0.94</td>
<td>118</td>
<td>0.94</td>
<td>0.99</td>
<td>236</td>
<td>96.3</td>
<td>&lt; 5mn31s</td>
</tr>
<tr>
<td>A</td>
<td>116</td>
<td>0.97</td>
<td>0.97</td>
<td>123</td>
<td>0.98</td>
<td>0.98</td>
<td>239</td>
<td>97.6</td>
<td>&lt; 5mn44s</td>
</tr>
</tbody>
</table>

A : positive weight = 3, negative weight = 0.5, promotion attribute = 3, demotion attribute = 0.3
For 5 times:

<table>
<thead>
<tr>
<th>Category</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>115</td>
<td>0.97</td>
<td>0.96</td>
<td>121</td>
<td>0.96</td>
<td>0.97</td>
<td>236</td>
<td>96.3</td>
<td>&lt; 7mn34s</td>
</tr>
<tr>
<td>A</td>
<td>118</td>
<td>0.99</td>
<td>0.89</td>
<td>112</td>
<td>0.89</td>
<td>0.99</td>
<td>230</td>
<td>93.9</td>
<td>&lt; 7mn35s</td>
</tr>
<tr>
<td>A</td>
<td>110</td>
<td>0.92</td>
<td>0.96</td>
<td>121</td>
<td>0.96</td>
<td>0.93</td>
<td>231</td>
<td>94.3</td>
<td>&lt; 7mn47s</td>
</tr>
</tbody>
</table>

A : positive weight = 3, negative weight = 0.5, promotion attribute = 3, demotion attribute = 0.3

For 10 times:

<table>
<thead>
<tr>
<th>Category</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>110</td>
<td>0.92</td>
<td>0.96</td>
<td>121</td>
<td>0.96</td>
<td>0.93</td>
<td>231</td>
<td>94.3</td>
<td>&lt; 13mn</td>
</tr>
<tr>
<td>A</td>
<td>111</td>
<td>0.93</td>
<td>0.97</td>
<td>123</td>
<td>0.98</td>
<td>0.94</td>
<td>234</td>
<td>95.5</td>
<td>&lt; 13mn42s</td>
</tr>
<tr>
<td>A</td>
<td>108</td>
<td>0.91</td>
<td>0.98</td>
<td>124</td>
<td>0.98</td>
<td>0.92</td>
<td>232</td>
<td>94.7</td>
<td>&lt; 14mn51s</td>
</tr>
</tbody>
</table>

A : positive weight = 3, negative weight = 0.5, promotion attribute = 3, demotion attribute = 0.3

What is interesting to see it that the time increases almost linearly whereas on average the results do not improve at all. Therefore learning 3 times over the training files should be the best settings:

- Don't take word number into account
- Random training
- Use words
- Learn 3 times

With these settings the algorithm has a correct classification percentage of around 95% which is fine.
The first results given by TIE are almost the same as TextClassifier. The TIE classifier also has trouble classifying documents when the category changes. The structure of the output file is as follow:

path_to_document|real_category_of_document|category_in_which_TIE_has_classified_the_document

When TIE classifies correctly a document, which means, when
real_category_of_document = category_in_which_TIE_has_classified_the_document
then it says so by writing a '+' in the output file.

Extract of the output file:

[..]
../TextCategorization/week4041/04017000/1sa3xku98csjssk.txt | economy (general) | +
../TextCategorization/week4041/04017000/1sa3xku98csqjnp.txt | economy (general) | +
../TextCategorization/week4041/15000000/1s9veffj12piep2c.txt | sport | economy (general)
../TextCategorization/week4041/15000000/1s9vfr34ear1yr3.txt | sport | economy (general)
[..]
../TextCategorization/week4041/15000000/1s9vf0ij691f6d3.txt | sport | +
../TextCategorization/week4041/15000000/1s9vdanz6u55xjt.txt | sport | +
../TextCategorization/week4041/15000000/1s9ve47dy64q8p.txt | sport | +
../TextCategorization/week42/04017000/1sc3jcbgfsalam63.txt | economy (general) | sport
../TextCategorization/week42/04017000/1sdaw0dkk4us5pi.txt | economy (general) | sport
../TextCategorization/week42/04017000/1sc5bo3d10jko95.txt | economy (general) | +
../TextCategorization/week42/04017000/1sc5jolcggigi77f.txt | economy (general) | sport
[..]
../TextCategorization/week42/04017000/1sd2gppadu7sj9o.txt | economy (general) | +
../TextCategorization/week42/04017000/1sda1168x86hnmj.txt | economy (general) | +
../TextCategorization/week42/15000000/1sc4511f2kkgkp5.txt | sport | economy (general)
../TextCategorization/week42/15000000/1sc4511f2kktzty.txt | sport | +

Here it is clearly shown that every time there is a category change, then the classifier makes some mistakes. At this time it is rightful to ask the question if this is a natural behavior of the algorithm because although there is no word about it in the Winnow-related papers, at least 2 Winnow implementations behaves similarly. This has been confirmed by C. Siefkes. Winnow should preferably be trained with random input, rather than having a sorted list of categories.
Here are the results for TIE with ascii codes:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Results</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>107</td>
<td>0.9</td>
<td>0.95</td>
<td>120</td>
<td>0.95</td>
<td>0.91</td>
<td>227</td>
<td>92.7</td>
<td>&lt;1mn40s</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>53</td>
<td>0.45</td>
<td>0.45</td>
<td>126</td>
<td>1</td>
<td>1</td>
<td>179</td>
<td>73.1</td>
<td>&lt;1mn7s</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>91</td>
<td>0.76</td>
<td>0.99</td>
<td>125</td>
<td>0.99</td>
<td>0.82</td>
<td>216</td>
<td>88.2</td>
<td>&lt;2mn13s</td>
</tr>
</tbody>
</table>

A: fixed training, 10 training, balanced winnow, ascii files
B: random training, 3 training, balanced winnow, ascii files
C: random training, 10 training, balanced winnow, ascii files

It is a pretty fair accuracy. However it can be improved as TIE is an information retrieval software so it can take plain text as input. That will make the tests closer to what could be done with TextClassifier.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Results</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>87</td>
<td>0.73</td>
<td>0.99</td>
<td>125</td>
<td>0.99</td>
<td>0.8</td>
<td>212</td>
<td>86.5</td>
<td>&lt;1mn40s</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>87</td>
<td>0.73</td>
<td>0.99</td>
<td>125</td>
<td>0.99</td>
<td>0.8</td>
<td>212</td>
<td>86.5</td>
<td>&lt;1mn40s</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>24</td>
<td>0.2</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>0.57</td>
<td>150</td>
<td>61.2</td>
<td>&lt;1mn12s</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>115</td>
<td>0.97</td>
<td>0.88</td>
<td>110</td>
<td>0.87</td>
<td>0.96</td>
<td>225</td>
<td>91.8</td>
<td>&lt;2mn5s</td>
</tr>
<tr>
<td></td>
<td>D'</td>
<td>83</td>
<td>0.7</td>
<td>0.99</td>
<td>125</td>
<td>0.99</td>
<td>0.78</td>
<td>208</td>
<td>84.9</td>
<td>&lt;2mn6s</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>117</td>
<td>0.98</td>
<td>0.81</td>
<td>99</td>
<td>0.79</td>
<td>0.98</td>
<td>216</td>
<td>88</td>
<td>&lt;1mn12s</td>
</tr>
</tbody>
</table>

A: random input, trained 10 times with training set, using iso-8859-1 charset
B: random input, trained 10 times with training set
C: random input, trained 3 times with training set
D: fixed input (economy, sport), trained 10 times with training set
D’: fixed input (sport, economy), trained 10 times with training set
E: fixed input (economy, sport), trained 3 times with training set

The best results are achieved with a fixed input and 10 times training.
IX.3.3 Opponent : SNoW

Over all the 245 files in the test set, SNoW managed to correctly classify 73.1% of them (179 files).

<table>
<thead>
<tr>
<th>Results</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>73.06</td>
<td>-</td>
</tr>
</tbody>
</table>

If the algorithm is set to learn during the testing phase, the percentage increases to almost 87%. But it is not supposed to act this way as for the future use as the human classifier will have no “previous knowledge” on the document to classify, and thus SNoW will not have any clue if its guess is right or wrong (so no possibility to learn).

<table>
<thead>
<tr>
<th>Results</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>86.94</td>
<td>-</td>
</tr>
</tbody>
</table>

The score drops to 76.3% when customized values for the promotion/demotion attribute, standard weight and threshold are specified.

IX.3.4 Winner

As far as I have gone in the tests, TextClassifier has the best overall score (94.7%) and is able to repeat this high classification rate with different parameters. This can be explained with the fact that TIE is not really a classification software but was built more for “feature extraction”; though its classification accuracy is pretty high (almost as high as TC).

About SNoW, its results are quite the same as TextClassifier (with ascii & the same set of parameters), which is between 70% and 80% of correct classification. One could think that these two software are even. However SNoW works only with numbers, and that means already processed data; whereas TextClassifier can take plain texts as input. And using words increases the classification correctness, which makes TextClassifier better than SNoW for this test.
IX.4 Rocchio

IX.4.1 TextClassifier

These results show that Rocchio has fair results compared to newer and classification-dedicated algorithms.

For ASCII features:
- using term frequency

Results for $\alpha = 0$, $\beta = 16$, $\gamma = 4$

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Economy</th>
<th>$R$</th>
<th>$P$</th>
<th>Sport</th>
<th>$R$</th>
<th>$P$</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT=100</td>
<td>81</td>
<td>0.68</td>
<td>0.96</td>
<td>123</td>
<td>0.98</td>
<td>0.76</td>
<td>204</td>
<td>83.27</td>
<td>&lt; 19s</td>
</tr>
<tr>
<td>FT=500</td>
<td>89</td>
<td>0.75</td>
<td>0.98</td>
<td>124</td>
<td>0.98</td>
<td>0.81</td>
<td>213</td>
<td>86.94</td>
<td>&lt; 18s</td>
</tr>
<tr>
<td>FT=1500</td>
<td>89</td>
<td>0.75</td>
<td>0.98</td>
<td>124</td>
<td>0.98</td>
<td>0.81</td>
<td>213</td>
<td>86.94</td>
<td>&lt; 39s</td>
</tr>
<tr>
<td>FT=2135</td>
<td>89</td>
<td>0.75</td>
<td>0.98</td>
<td>124</td>
<td>0.98</td>
<td>0.81</td>
<td>213</td>
<td>86.94</td>
<td>&lt; 1mn6s</td>
</tr>
</tbody>
</table>

Results for $\alpha = 0$, $\beta = 1$, $\gamma = 0$

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Economy</th>
<th>$R$</th>
<th>$P$</th>
<th>Sport</th>
<th>$R$</th>
<th>$P$</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT=100</td>
<td>101</td>
<td>0.85</td>
<td>0.89</td>
<td>114</td>
<td>0.90</td>
<td>0.86</td>
<td>215</td>
<td>87.76</td>
<td>&lt; 17s</td>
</tr>
<tr>
<td>FT=500</td>
<td>106</td>
<td>0.89</td>
<td>0.94</td>
<td>119</td>
<td>0.94</td>
<td>0.90</td>
<td>225</td>
<td>91.84</td>
<td>&lt; 21s</td>
</tr>
<tr>
<td>FT=1500</td>
<td>106</td>
<td>0.89</td>
<td>0.94</td>
<td>119</td>
<td>0.94</td>
<td>0.90</td>
<td>225</td>
<td>91.84</td>
<td>&lt; 46s</td>
</tr>
<tr>
<td>FT=2135</td>
<td>106</td>
<td>0.89</td>
<td>0.94</td>
<td>119</td>
<td>0.94</td>
<td>0.90</td>
<td>225</td>
<td>91.84</td>
<td>&lt; 1mn3s</td>
</tr>
</tbody>
</table>

Results for $\alpha = 0$, $\beta = 2$, $\gamma = 1$

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Economy</th>
<th>$R$</th>
<th>$P$</th>
<th>Sport</th>
<th>$R$</th>
<th>$P$</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT=100</td>
<td>43</td>
<td>0.31</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>0.62</td>
<td>169</td>
<td>68.78</td>
<td>&lt; 14s</td>
</tr>
<tr>
<td>FT=500</td>
<td>42</td>
<td>0.35</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>0.62</td>
<td>168</td>
<td>68.57</td>
<td>&lt; 21s</td>
</tr>
<tr>
<td>FT=2135</td>
<td>42</td>
<td>0.35</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>0.62</td>
<td>168</td>
<td>68.57</td>
<td>&lt; 55s</td>
</tr>
</tbody>
</table>
Results for \(\alpha = 0, \beta = 16, \gamma = 4\)

<table>
<thead>
<tr>
<th>Results</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT=100</td>
<td>81</td>
<td>0.68</td>
<td>0.96</td>
<td>123</td>
<td>0.98</td>
<td>0.76</td>
<td>204</td>
<td>83.27</td>
<td>&lt; 16s</td>
</tr>
<tr>
<td>FT=500</td>
<td>89</td>
<td>0.75</td>
<td>0.98</td>
<td>124</td>
<td>0.98</td>
<td>0.81</td>
<td>213</td>
<td>86.94</td>
<td>&lt; 26s</td>
</tr>
<tr>
<td>FT=2135</td>
<td>89</td>
<td>0.75</td>
<td>0.98</td>
<td>124</td>
<td>0.98</td>
<td>0.81</td>
<td>213</td>
<td>86.94</td>
<td>&lt; 1mn1s</td>
</tr>
</tbody>
</table>

Results for \(\alpha = 0, \beta = 2, \gamma = 1\)

<table>
<thead>
<tr>
<th>Results</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT=500</td>
<td>42</td>
<td>0.35</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>0.62</td>
<td>168</td>
<td>68.57</td>
<td>&lt; 21s</td>
</tr>
<tr>
<td>FT=1500</td>
<td>42</td>
<td>0.35</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>0.62</td>
<td>168</td>
<td>68.57</td>
<td>&lt; 43s</td>
</tr>
<tr>
<td>FT=2135</td>
<td>42</td>
<td>0.35</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>0.62</td>
<td>168</td>
<td>68.57</td>
<td>&lt; 54s</td>
</tr>
</tbody>
</table>

Results for \(\alpha = 0, \beta = 1, \gamma = 0\)

<table>
<thead>
<tr>
<th>Results</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT=100</td>
<td>101</td>
<td>0.85</td>
<td>0.89</td>
<td>114</td>
<td>0.90</td>
<td>0.86</td>
<td>215</td>
<td>87.76</td>
<td>&lt; 18s</td>
</tr>
<tr>
<td>FT=500</td>
<td>106</td>
<td>0.89</td>
<td>0.94</td>
<td>119</td>
<td>0.94</td>
<td>0.90</td>
<td>225</td>
<td>91.84</td>
<td>&lt; 19s</td>
</tr>
<tr>
<td>FT=1500</td>
<td>106</td>
<td>0.89</td>
<td>0.94</td>
<td>119</td>
<td>0.94</td>
<td>0.90</td>
<td>225</td>
<td>91.84</td>
<td>&lt; 44s</td>
</tr>
<tr>
<td>FT=2135</td>
<td>106</td>
<td>0.89</td>
<td>0.94</td>
<td>119</td>
<td>0.94</td>
<td>0.90</td>
<td>225</td>
<td>91.84</td>
<td>&lt; 1mn</td>
</tr>
</tbody>
</table>
For string features:
- using term frequency

Results for $\alpha = 0$, $\beta = 16$, $\gamma = 4$

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Results</th>
<th>Economy</th>
<th>$R$</th>
<th>$P$</th>
<th>Sport</th>
<th>$R$</th>
<th>$P$</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT=100</td>
<td>99</td>
<td>0.83</td>
<td>0.93</td>
<td>119</td>
<td>0.94</td>
<td>0.86</td>
<td>218</td>
<td>88.98</td>
<td>&lt; 54s</td>
<td></td>
</tr>
<tr>
<td>FT=500</td>
<td>103</td>
<td>0.87</td>
<td>0.97</td>
<td>123</td>
<td>0.98</td>
<td>0.88</td>
<td>226</td>
<td>92.24</td>
<td>&lt; 51s</td>
<td></td>
</tr>
<tr>
<td>FT=1000</td>
<td>104</td>
<td>0.87</td>
<td>0.97</td>
<td>123</td>
<td>0.98</td>
<td>0.89</td>
<td>227</td>
<td>92.65</td>
<td>&lt; 1mn24s</td>
<td></td>
</tr>
<tr>
<td>FT=5000</td>
<td>105</td>
<td>0.88</td>
<td>0.97</td>
<td>123</td>
<td>0.98</td>
<td>0.9</td>
<td>228</td>
<td>93.06</td>
<td>&lt; 5mn48s</td>
<td></td>
</tr>
<tr>
<td>FT=10000</td>
<td>105</td>
<td>0.88</td>
<td>0.97</td>
<td>123</td>
<td>0.98</td>
<td>0.9</td>
<td>228</td>
<td>93.06</td>
<td>&lt; 17mn20s</td>
<td></td>
</tr>
</tbody>
</table>

Results for $\alpha = 0$, $\beta = 2$, $\gamma = 1$

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Results</th>
<th>Economy</th>
<th>$R$</th>
<th>$P$</th>
<th>Sport</th>
<th>$R$</th>
<th>$P$</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT=100</td>
<td>73</td>
<td>0.61</td>
<td>0.99</td>
<td>125</td>
<td>0.99</td>
<td>0.73</td>
<td>198</td>
<td>80.82</td>
<td>&lt; 50s</td>
<td></td>
</tr>
<tr>
<td>FT=500</td>
<td>77</td>
<td>0.65</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>0.75</td>
<td>203</td>
<td>82.86</td>
<td>&lt; 1mn2s</td>
<td></td>
</tr>
<tr>
<td>FT=1000</td>
<td>78</td>
<td>0.66</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>0.75</td>
<td>204</td>
<td>83.27</td>
<td>&lt; 1mn19s</td>
<td></td>
</tr>
<tr>
<td>FT=5000</td>
<td>81</td>
<td>0.68</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>0.77</td>
<td>207</td>
<td>84.49</td>
<td>&lt; 5mn10s</td>
<td></td>
</tr>
<tr>
<td>FT=10000</td>
<td>81</td>
<td>0.68</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>0.77</td>
<td>207</td>
<td>84.49</td>
<td>&lt; 18mn30s</td>
<td></td>
</tr>
</tbody>
</table>

Results for $\alpha = 0$, $\beta = 1$, $\gamma = 0$

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Results</th>
<th>Economy</th>
<th>$R$</th>
<th>$P$</th>
<th>Sport</th>
<th>$R$</th>
<th>$P$</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT=100</td>
<td>112</td>
<td>0.94</td>
<td>0.88</td>
<td>110</td>
<td>0.87</td>
<td>0.94</td>
<td>222</td>
<td>90.61</td>
<td>&lt; 39s</td>
<td></td>
</tr>
<tr>
<td>FT=500</td>
<td>111</td>
<td>0.93</td>
<td>0.92</td>
<td>116</td>
<td>0.92</td>
<td>0.94</td>
<td>227</td>
<td>92.65</td>
<td>&lt; 52s</td>
<td></td>
</tr>
<tr>
<td>FT=1000</td>
<td>112</td>
<td>0.94</td>
<td>0.92</td>
<td>116</td>
<td>0.92</td>
<td>0.94</td>
<td>228</td>
<td>93.06</td>
<td>&lt; 1mn3s</td>
<td></td>
</tr>
<tr>
<td>FT=5000</td>
<td>111</td>
<td>0.93</td>
<td>0.92</td>
<td>116</td>
<td>0.92</td>
<td>0.94</td>
<td>227</td>
<td>92.65</td>
<td>&lt; 5mn38s</td>
<td></td>
</tr>
<tr>
<td>FT=10000</td>
<td>111</td>
<td>0.93</td>
<td>0.92</td>
<td>116</td>
<td>0.92</td>
<td>0.94</td>
<td>227</td>
<td>92.65</td>
<td>&lt; 17mn5s</td>
<td></td>
</tr>
</tbody>
</table>
- using TFIDF

Result for $\alpha = 0$, $\beta = 16$, $\gamma = 4$

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Results</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT=100</td>
<td></td>
<td>99</td>
<td>0.83</td>
<td>0.93</td>
<td>119</td>
<td>0.94</td>
<td>0.86</td>
<td>218</td>
<td>88.98</td>
<td>&lt; 51s</td>
</tr>
<tr>
<td>FT=500</td>
<td></td>
<td>103</td>
<td>0.87</td>
<td>0.97</td>
<td>123</td>
<td>0.98</td>
<td>0.88</td>
<td>226</td>
<td>92.24</td>
<td>&lt; 1mn3s</td>
</tr>
<tr>
<td>FT=1000</td>
<td></td>
<td>104</td>
<td>0.87</td>
<td>0.97</td>
<td>123</td>
<td>0.98</td>
<td>0.89</td>
<td>227</td>
<td>92.65</td>
<td>&lt; 1mn14s</td>
</tr>
<tr>
<td>FT=5000</td>
<td></td>
<td>105</td>
<td>0.88</td>
<td>0.97</td>
<td>123</td>
<td>0.98</td>
<td>0.9</td>
<td>228</td>
<td>93.06</td>
<td>&lt; 5mn43s</td>
</tr>
<tr>
<td>FT=10000</td>
<td></td>
<td>105</td>
<td>0.88</td>
<td>0.97</td>
<td>123</td>
<td>0.98</td>
<td>0.9</td>
<td>228</td>
<td>93.06</td>
<td>&lt; 17mn15s</td>
</tr>
</tbody>
</table>

Result for $\alpha = 0$, $\beta = 1$, $\gamma = 0$

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Results</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT=100</td>
<td></td>
<td>112</td>
<td>0.94</td>
<td>0.88</td>
<td>110</td>
<td>0.87</td>
<td>0.94</td>
<td>222</td>
<td>90.61</td>
<td>&lt; 40s</td>
</tr>
<tr>
<td>FT=500</td>
<td></td>
<td>111</td>
<td>0.93</td>
<td>0.92</td>
<td>116</td>
<td>0.92</td>
<td>0.94</td>
<td>227</td>
<td>92.65</td>
<td>&lt; 53s</td>
</tr>
<tr>
<td>FT=1000</td>
<td></td>
<td>112</td>
<td>0.94</td>
<td>0.92</td>
<td>116</td>
<td>0.92</td>
<td>0.94</td>
<td>228</td>
<td>93.06</td>
<td>&lt; 1mn13s</td>
</tr>
<tr>
<td>FT=5000</td>
<td></td>
<td>111</td>
<td>0.93</td>
<td>0.92</td>
<td>116</td>
<td>0.92</td>
<td>0.94</td>
<td>227</td>
<td>92.65</td>
<td>&lt; 6mn21s</td>
</tr>
<tr>
<td>FT=10000</td>
<td></td>
<td>111</td>
<td>0.93</td>
<td>0.92</td>
<td>116</td>
<td>0.92</td>
<td>0.94</td>
<td>227</td>
<td>92.65</td>
<td>&lt; 19mn17s</td>
</tr>
</tbody>
</table>

Results for $\alpha = 0$, $\beta = 2$, $\gamma = 1$

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Results</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT=100</td>
<td></td>
<td>73</td>
<td>0.61</td>
<td>0.99</td>
<td>125</td>
<td>0.99</td>
<td>0.73</td>
<td>198</td>
<td>80.82</td>
<td>&lt; 38s</td>
</tr>
<tr>
<td>FT=500</td>
<td></td>
<td>77</td>
<td>0.65</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>0.75</td>
<td>203</td>
<td>82.86</td>
<td>&lt; 1mn2s</td>
</tr>
<tr>
<td>FT=1000</td>
<td></td>
<td>78</td>
<td>0.66</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>0.75</td>
<td>204</td>
<td>83.27</td>
<td>&lt; 1mn21s</td>
</tr>
<tr>
<td>FT=5000</td>
<td></td>
<td>81</td>
<td>0.68</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>0.77</td>
<td>207</td>
<td>84.49</td>
<td>&lt; 7mn25s</td>
</tr>
<tr>
<td>FT=10000</td>
<td></td>
<td>81</td>
<td>0.68</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>0.77</td>
<td>207</td>
<td>84.49</td>
<td>&lt; 20mn51s</td>
</tr>
</tbody>
</table>
The best settings are with:
- words
- term frequency as the feature selector
- $\beta = 1$ and $\gamma = 0$

It classifies correctly 93.06% of the documents in 1mn03s.

**IX.4.2 Opponent : Awacate**

Awacate is a software built especially for text classification. As it works with plain text and it provides several options for tuning (stemming, feature selection) the results are very high:

<table>
<thead>
<tr>
<th>Results</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>83</td>
<td>0.7</td>
<td>0.98</td>
<td>124</td>
<td>0.98</td>
<td>0.78</td>
<td>207</td>
<td>84.49</td>
<td>&lt; 60s</td>
</tr>
<tr>
<td>B</td>
<td>83</td>
<td>0.7</td>
<td>0.98</td>
<td>124</td>
<td>0.98</td>
<td>0.78</td>
<td>207</td>
<td>84.49</td>
<td>&lt; 1mn10s</td>
</tr>
<tr>
<td>C</td>
<td>83</td>
<td>0.7</td>
<td>0.98</td>
<td>124</td>
<td>0.98</td>
<td>0.78</td>
<td>207</td>
<td>84.49</td>
<td>&lt; 1mn10s</td>
</tr>
<tr>
<td>D</td>
<td>111</td>
<td>0.93</td>
<td>0.99</td>
<td>125</td>
<td>0.99</td>
<td>0.94</td>
<td>236</td>
<td>96.33</td>
<td>&lt; 1mn12s</td>
</tr>
<tr>
<td>E</td>
<td>118</td>
<td>0.98</td>
<td>0.99</td>
<td>124</td>
<td>0.98</td>
<td>0.99</td>
<td>242</td>
<td>98.78</td>
<td>&lt; 1mn02s</td>
</tr>
</tbody>
</table>

**A:**
- English language (no stemming, no stop word removal, not restricted to alphabetical words)
- Feature selection : term frequency (Awacate worked on 2093 features)
- Term weighting : term frequency
- $\beta = 16$ & $\gamma = 4$

**B:** same as A with stop word removal and English stemming

**C:** same as B with alphabetical words only

**D:** same as C with term weighting = term frequency + TFIDF

**E:** same as D with feature selection = Chi² (Awacate worked on 2480 features)
The next tests are all done with the E settings:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Results</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>E, $\beta = 16$, $\gamma = 4$</td>
<td>118</td>
<td>0.98</td>
<td>0.99</td>
<td>124</td>
<td>0.98</td>
<td>0.99</td>
<td>242</td>
<td>98.78</td>
<td>&lt; 1mn15s</td>
<td></td>
</tr>
<tr>
<td>E, $\beta = 1$, $\gamma = 1$</td>
<td>116</td>
<td>0.97</td>
<td>0.97</td>
<td>123</td>
<td>0.98</td>
<td>0.98</td>
<td>239</td>
<td>97.6</td>
<td>&lt; 1mn13s</td>
<td></td>
</tr>
<tr>
<td>E, $\beta = 10$, $\gamma = 2$</td>
<td>118</td>
<td>0.98</td>
<td>0.99</td>
<td>124</td>
<td>0.98</td>
<td>0.99</td>
<td>242</td>
<td>98.78</td>
<td>&lt; 1mn13s</td>
<td></td>
</tr>
<tr>
<td>E, $\beta = 1$, $\gamma = 0$</td>
<td>118</td>
<td>0.98</td>
<td>0.99</td>
<td>124</td>
<td>0.98</td>
<td>0.99</td>
<td>242</td>
<td>98.78</td>
<td>&lt; 1mn21s</td>
<td></td>
</tr>
</tbody>
</table>

These tests show that Awacate performs really well with swedish texts, although the configuration (stemming) was implemented for the english language. This enable the possibility for even higher accuracy if swedish stemming is used.

**IX.4.3 Winner:**

From the results (accuracy as well as time) Awacate seems to outperform TC. It classifies on average 5 points (93% < 98%) better than TC. As far as processing time is concerned, it is on average 3 times faster than TC. Awacate is then not only better but also faster that TC.

**IX.5 Tricks to improve accuracy**

**IX.5.1 Do not classify documents if unsure**

In Naïve Bayes, I have tried to reduce the amount of misclassified file. I implemented that by not classifying the files for which the highest probability was not very high compared to the second highest (i.e. if highest_probability < 10*second_highest_probability)

It was not very interesting as more than half of the non-classified files would have been classified correctly.

**IX.5.2 Multiclassifier**

The idea behind the multiclassifier is that if classifiers work together, maybe the flaws of one could be balanced by the strengths of another, thus improving the overall accuracy?

I have created a 5th algorithm that is composed of Naïve Bayes, kNN and Rocchio. The results were worse that expected, as the classification accuracy is 82%, thus far behind what Naïve Bayes can achieve by itself.
IX.6 Final Tests

My last task was to select the best algorithms of the 4 and try to classify documents not into 2 categories but in all the 40 IPTC categories. The categories are presented on next page.

For these tests I will use Naïve Bayes algorithm with TC as it is the one that has provided the highest accuracy. The time needed to classify could be a problem, that's why I will also use Awacate to have the results of a faster software.

IX.6.1 Solving the small number of texts

I already said in the method section that I restricted the tests to 2 categories because the documents were not numerous enough in the other 38 categories. Now I want to test the best algorithm of the 40 categories again. That means I need to have more documents in each category; this can be achieved thanks to Corren which provided 4 more weeks (weeks 43-44-45-46).
The Corren categories are:

- 1mr1j90bv33g8yx Sport och fritid
- 1n7v30pnm5cfcm LHC
- 5xamzc10l2uzm OS
- 1rlcsro9hga7f1 Fotboll
- 1mr1jb2rnn3keh Ekonomi
gztpq7e7bzx8a1l Företagande
gztpsl23gt3pm7h Arbetsliv
gztpu56ksonlydab Forsknings och teknik
1mr1jdjdw6u6oag Kultur och nöje
1mr86087x2df21u Kultur
1mr860tuugk75al Nöje
1pw9a7l80j15bu7 Vätternfestivalen
haz36d6wan61jkl Student
1mr1jwoou6hwof Recensioner
1naxgzeuh28mvjx Böcker
1naxhebh1t29but Film
1naxhiicgylq08 Skivor
1naxhjxq71qxx9nr Teater
1naxhvqvnvl11 Konsert
1naxi2u3s4q6v3 Spel
1ncyandh48x0hv2 Konst
1mr1l1i3zuhjt90 Ledare
1mr5a5d659qt6u Konsument
1nb54htut8hj20 Matlust
1nb54htut8hj20 Matlust
1nb54htut8hj20 Matlust
1en0uc1lg092hj Spel
1pktql25dfvtt EMU
haz3iz3zehltxyr Motor och trafik
heihlegx62c2y6d Jul
1nbhuuq2g4a43u6 Bostad

The IPTC categories are:

- 15000000 sport
- 10010000 leisure (general)
- 15031000 ice hockey
- 15073001 Summer Olympics
- 15073002 Winter Olympics
- 15054000 soccer
- 04017000 economy (general)
- 04006000 financial and business service
- 04018000 business (general)
- 09012000 work relations
- 09003000 employment
- 13006000 research
- 04016037 research and development
- 01000000 arts, culture and entertainment
- 01022000 culture (general)
- 01021000 entertainment (general)
- 01004000 festive event (including carnival)
- 050100001 students
- 01026004 reviews
- 04010002 book
- 01010000 literature
- 01005000 cinema
- 01011000 music
- 01017000 theatre
- 10001000 game
- 01020000 arts (general)
- 14006000 family
- 14006001 parent and child
- 11000000 politics
- 15050004 Europe
- 08001000 animal
- 04016055 consumers
- 04008002 consumer issue
- 100030000 gastronomy
- 04013002 food
- 04008018 currency values
- 04008010 international economic institution
- 04011002 automotive equipment
- 10009000 lifestyle (house and home)
IX.6.2 First test

The first test was to classify only the (sport + economy) documents of the test set. This to see if the accuracy had increased/decreased from the increase of documents.

**Training set**: weeks 40-41-43-44-45 (1241 documents, 309663 words, 249.5 words/docs)
- 560 economy documents
- 681 sport documents

**Test set**: weeks 42-46 (480 documents, 123486 words, 257.26 words/docs)
- 241 economy documents
- 239 sport documents

The results:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>239</td>
<td>0.99</td>
<td>0.98</td>
<td>235</td>
<td>0.98</td>
<td>0.99</td>
<td>474</td>
<td>98.75</td>
<td>&lt; 23s</td>
</tr>
<tr>
<td>B</td>
<td>240</td>
<td>0.995</td>
<td>0.98</td>
<td>234</td>
<td>0.98</td>
<td>0.995</td>
<td>474</td>
<td>98.75</td>
<td>&lt; 5mn30s</td>
</tr>
</tbody>
</table>

A : Awacate, term selection = Chi²
B : TC & NB with words

This confirms TC as the best document classification software for our purpose and NB as the best algorithm.

IX.6.3 Second test

Sometimes a Corren ID corresponds to more than one IPTC category (see file correnelin.txt in appendix III).

Example:
1mr1j90bv33g8yx (Sport och fritid) correspond to 15000000 (sport) and 10010000 (leisure (general))

For this example, sport and leisure only correspond to Sport och fritid, which means that the categories will contain the very same files. It is an important problem that there is no difference between categories, because the classifier will not be able to choose between one or another when classifying a “Sport och fritid” document. In other words, these 2 categories are totally overlapping.

This problem had not happened before because the work was carried out on only 2 disjoint categories.

The solution is to simplify the category list by removing redundant categories (in this case leisure (general) is to be removed).
Therefore clearing redundant categories was useful, as well as checking that a category in the test set is also in the training set (and vice-versa). This is to avoid training the algorithm for nothing (if the category does not exist in the test set) or classifying a document without having properly trained the algorithm (if the category does not exist in the training set).

That lead to 12 categories in the training set with 742 files, and 12 categories in the test set with 371 files.

**Training set**: weeks 40-41 (742 documents, 192247 words, 259.09 words/docs)

**Test set**: week 42 (371 documents, 102964 words, 277.53 words/docs)

<table>
<thead>
<tr>
<th>Category</th>
<th>Correct</th>
<th>Total</th>
<th>R</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cat. 150000000</td>
<td>122</td>
<td>126</td>
<td>0.97</td>
<td>0.93</td>
</tr>
<tr>
<td>Cat. 150540000</td>
<td>3</td>
<td>7</td>
<td>0.43</td>
<td>0.5</td>
</tr>
<tr>
<td>Cat. 040170000</td>
<td>116</td>
<td>119</td>
<td>0.97</td>
<td>0.87</td>
</tr>
<tr>
<td>Cat. 010220000</td>
<td>18</td>
<td>26</td>
<td>0.69</td>
<td>0.47</td>
</tr>
<tr>
<td>Cat. 010210000</td>
<td>22</td>
<td>27</td>
<td>0.81</td>
<td>0.59</td>
</tr>
<tr>
<td>Cat. 100010000</td>
<td>4</td>
<td>14</td>
<td>0.29</td>
<td>1</td>
</tr>
<tr>
<td>Cat. 140060000</td>
<td>7</td>
<td>12</td>
<td>0.58</td>
<td>0.78</td>
</tr>
<tr>
<td>Cat. 14006001</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Cat. 110000000</td>
<td>6</td>
<td>9</td>
<td>0.67</td>
<td>1</td>
</tr>
<tr>
<td>Cat. 040130002</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Cat. 04011002</td>
<td>1</td>
<td>5</td>
<td>0.2</td>
<td>1</td>
</tr>
<tr>
<td>Cat. 100090000</td>
<td>1</td>
<td>22</td>
<td>0.05</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>304</td>
<td>371</td>
<td>81.94%</td>
<td></td>
</tr>
<tr>
<td><strong>Time</strong></td>
<td>&lt; 4mn33s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Correct</td>
<td>Total</td>
<td>$R$</td>
<td>$P$</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------</td>
<td>-------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>Cat. 15000000</td>
<td>124</td>
<td>126</td>
<td>0.98</td>
<td>0.92</td>
</tr>
<tr>
<td>Cat. 15054000</td>
<td>1</td>
<td>7</td>
<td>0.14</td>
<td>0.5</td>
</tr>
<tr>
<td>Cat. 04017000</td>
<td>114</td>
<td>119</td>
<td>0.84</td>
<td>0.96</td>
</tr>
<tr>
<td>Cat. 01022000</td>
<td>17</td>
<td>26</td>
<td>0.65</td>
<td>0.57</td>
</tr>
<tr>
<td>Cat. 01021000</td>
<td>21</td>
<td>27</td>
<td>0.78</td>
<td>0.41</td>
</tr>
<tr>
<td>Cat. 10001000</td>
<td>2</td>
<td>14</td>
<td>0.14</td>
<td>1</td>
</tr>
<tr>
<td>Cat. 14006000</td>
<td>8</td>
<td>12</td>
<td>0.67</td>
<td>1</td>
</tr>
<tr>
<td>Cat. 14006001</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Cat. 11000000</td>
<td>4</td>
<td>9</td>
<td>0.4</td>
<td>1</td>
</tr>
<tr>
<td>Cat. 04013002</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Cat. 04011002</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Cat. 10009000</td>
<td>0</td>
<td>22</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>295</td>
<td>371</td>
<td></td>
<td>79.51%</td>
</tr>
<tr>
<td><strong>Time</strong></td>
<td>&lt; 16s</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

With few categories having most documents, I think 81.84% is a fairly good result. In the third test the results should be higher because of the higher number of documents per category.
### IX.6.4 Third test

Training set: weeks 40-41-43-44-45 (1795 documents, 461597 words, 257.2 words/docs)  
Test set: weeks 42-46 (701 documents, 188339 words, 268.67 words/docs)

<table>
<thead>
<tr>
<th>Category</th>
<th>Results</th>
<th>Correct</th>
<th>Total</th>
<th>R</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cat. 15000000</td>
<td></td>
<td>231</td>
<td>239</td>
<td>0.97</td>
<td>0.98</td>
</tr>
<tr>
<td>Cat. 15054000</td>
<td></td>
<td>4</td>
<td>7</td>
<td>0.57</td>
<td>0.5</td>
</tr>
<tr>
<td>Cat. 04017000</td>
<td></td>
<td>233</td>
<td>241</td>
<td>0.97</td>
<td>0.92</td>
</tr>
<tr>
<td>Cat. 01022000</td>
<td></td>
<td>33</td>
<td>45</td>
<td>0.73</td>
<td>0.61</td>
</tr>
<tr>
<td>Cat. 01021000</td>
<td></td>
<td>52</td>
<td>61</td>
<td>0.85</td>
<td>0.73</td>
</tr>
<tr>
<td>Cat. 05010001</td>
<td></td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cat. 10001000</td>
<td></td>
<td>20</td>
<td>21</td>
<td>0.95</td>
<td>1</td>
</tr>
<tr>
<td>Cat. 14006000</td>
<td></td>
<td>20</td>
<td>24</td>
<td>0.83</td>
<td>0.8</td>
</tr>
<tr>
<td>Cat. 14006001</td>
<td></td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Cat. 11000000</td>
<td></td>
<td>16</td>
<td>20</td>
<td>0.8</td>
<td>0.94</td>
</tr>
<tr>
<td>Cat. 04013002</td>
<td></td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Cat. 04011002</td>
<td></td>
<td>4</td>
<td>9</td>
<td>0.44</td>
<td>1</td>
</tr>
<tr>
<td>Cat. 10009000</td>
<td></td>
<td>5</td>
<td>26</td>
<td>0.19</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>624</td>
<td>701</td>
<td>89.02%</td>
<td></td>
</tr>
</tbody>
</table>

Time < 13mn53s
### Awacate with NB

<table>
<thead>
<tr>
<th>Category</th>
<th>Correct</th>
<th>Total</th>
<th>R</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cat. 15000000</td>
<td>233</td>
<td>239</td>
<td>0.97</td>
<td>0.97</td>
</tr>
<tr>
<td>Cat. 15054000</td>
<td>1</td>
<td>7</td>
<td>0.14</td>
<td>0.5</td>
</tr>
<tr>
<td>Cat. 04017000</td>
<td>231</td>
<td>241</td>
<td>0.96</td>
<td>0.89</td>
</tr>
<tr>
<td>Cat. 01022000</td>
<td>35</td>
<td>45</td>
<td>0.78</td>
<td>0.51</td>
</tr>
<tr>
<td>Cat. 01021000</td>
<td>49</td>
<td>61</td>
<td>0.8</td>
<td>0.67</td>
</tr>
<tr>
<td>Cat 05010001</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cat. 10001000</td>
<td>19</td>
<td>21</td>
<td>0.9</td>
<td>0.95</td>
</tr>
<tr>
<td>Cat. 14006000</td>
<td>13</td>
<td>24</td>
<td>0.54</td>
<td>0.93</td>
</tr>
<tr>
<td>Cat. 14006001</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Cat. 11000000</td>
<td>12</td>
<td>20</td>
<td>0.6</td>
<td>1</td>
</tr>
<tr>
<td>Cat. 04013002</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Cat. 04011002</td>
<td>3</td>
<td>9</td>
<td>0.3</td>
<td>0.75</td>
</tr>
<tr>
<td>Cat. 10009000</td>
<td>1</td>
<td>26</td>
<td>0.04</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>603</td>
<td>701</td>
<td>86.02%</td>
<td></td>
</tr>
<tr>
<td><strong>Time</strong></td>
<td>&lt;28s</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
IX.6.5 Fourth test

Training set: weeks 40-41-42-43-44-45-46 (2468 docs, 645952 words, 261.7 words/docs)
Test set: weeks 47-48-49-50 (1443 documents, 372907 words, 258.42 words/docs)

<table>
<thead>
<tr>
<th>Category</th>
<th>Results</th>
<th>Correct</th>
<th>Total</th>
<th>R</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cat. 15000000</td>
<td></td>
<td>519</td>
<td>545</td>
<td>0.95</td>
<td>0.98</td>
</tr>
<tr>
<td>Cat. 15031000</td>
<td></td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cat. 04017000</td>
<td></td>
<td>450</td>
<td>469</td>
<td>0.96</td>
<td>0.89</td>
</tr>
<tr>
<td>Cat. 01000000</td>
<td></td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cat. 01022000</td>
<td></td>
<td>76</td>
<td>103</td>
<td>0.74</td>
<td>0.70</td>
</tr>
<tr>
<td>Cat. 01021000</td>
<td></td>
<td>99</td>
<td>119</td>
<td>0.83</td>
<td>0.72</td>
</tr>
<tr>
<td>Cat 05010001</td>
<td></td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cat. 10001000</td>
<td></td>
<td>41</td>
<td>41</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Cat. 14006000</td>
<td></td>
<td>42</td>
<td>54</td>
<td>0.78</td>
<td>0.75</td>
</tr>
<tr>
<td>Cat. 14006001</td>
<td></td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>0.8</td>
</tr>
<tr>
<td>Cat. 11000000</td>
<td></td>
<td>27</td>
<td>38</td>
<td>0.71</td>
<td>0.96</td>
</tr>
<tr>
<td>Cat. 04018002</td>
<td></td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cat. 04013002</td>
<td></td>
<td>11</td>
<td>11</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Cat. 04011002</td>
<td></td>
<td>9</td>
<td>17</td>
<td>0.53</td>
<td>0.9</td>
</tr>
<tr>
<td>Cat. 10009000</td>
<td></td>
<td>9</td>
<td>29</td>
<td>0.31</td>
<td>0.69</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>1287</td>
<td>1443</td>
<td></td>
<td>89.19%</td>
</tr>
</tbody>
</table>

Time < 24mn44s
These four tests show more or less that feature selection greatly improves processing time, Awacate being a lot faster than TC. However this comes with a small loss in accuracy as words are removed.
X Discussion

X.1 Choice of algorithms

The algorithms I have chosen to implement/test/compare are Naïve Bayes, kNN, Winnow and Rocchio. From the documentation found it appeared that these were the most famous in the TC field. This was the first reason why I selected them. Another reason lies in the fact that although Rocchio and Naïve Bayes are “old” or not “dedicated” to TC, they still perform quite good according to studies. I wanted to benchmark them against more recent, more accurate algorithms like kNN or Winnow.

X.2 Choices of tests methods

The goal of this work is to classify news articles, therefore the primary input documents that I wanted to test were plain texts, composed of words. So far my software accepts only text files. On the other hand, classification software in general are designed to classify any digital document so they accept specific input (mainly, a feature list where features are represented as numbers). In order to use these software the text documents had to be converted into an input readable by the software. For various reasons (time, lack of knowledge of stemming methods / of swedish language, ...) I chose to represent words as the sum of the ascii code of the word letters. The word “hockey” for example, would be represented by:

h = 104
o = 111
c = 99
k = 107
e = 101
y = 121
=> hockey = 643

In order to have a fair comparison between the software, my program can convert words in numbers, and it will then treat the number 643 as the string “643”.
X.3 The limits of my methods

Using ascii codes to convert words seemed an easy choice. However this approach has an important drawback which is that different words (anagrams) may be represented by the same number (lose information and reduce accuracy).

A better method would still convert words to numeric values but without losing information. The mathematical function should therefore be bijective (to one specific words correspond one and only one specific number).

In my training set there were 21976 “string” features and only 2135 “ascii” features; which means that on average 10 different words were represented as 1 number.

X.4 Time issue

The time issue has been discussed just a few times in the various papers that have been read. As far as I'm concerned, I think this can be explained mostly by the fact that it depends (as well as the performance in accuracy) on the end-user of the software. If the user needs a document to be classified at once (in the case of a large amount of documents being classified) maybe the accuracy could suffer; whereas if the user is more focused on accuracy maybe the time needed to classify the document would be less important.

In my tests I have not taken into account the time needed to convert documents from the “words” format to the “ascii” format. Depending on the software (SNoW, Torch3) this time could vary between 1 and 2 mn.

X.5 Naïve Bayes problem

I had some problems understanding the Naïve Bayes algorithm and how to transform it into java code. First I managed to solve an issue by becoming aware of the multi-variate Bernouilli and the multinominal versions of the algorithm. Some other problems appeared but were dealt with by testing and tuning the code.

Finally there was a major problem in the code as my software was classifying almost all economy documents in the sport category and vice-versa. I think it can be explained by :
  - let be a category A with 1000 words
  - let be a category B with 10 words
  - let be a document D with 10 words
The classification is done as follow:

for each category C
    for each word W in D
        if C contains W
            then probability = probability + prob(W, C)
        endif
    endfor
endfor

The probability of each of B's words will be high whereas the probability for A's words will be low (p(10) >> p(1000)). Thus, even if D belongs to A, it will most likely be classified in B if there are enough B's words in D (no matter how many A's words are in D).

I have solved this by adding a malus to B-like categories:

for each category C
    for each word W in D
        if C contains W
            then probability = probability + prob(W, C)
        else probability = probability + very_low_prob
        endif
    endfor
endfor

very_low_prob emulates the fact that for word W, D would have a very low probability to be classified in C.

\[
\text{prob}(W, C) = \log\left(\frac{\text{frequency}(W \text{ in } C) + 1}{\text{total number of words in } C}\right);
\]
\[
\text{very_low_prob} = \log\left(\frac{1}{\text{total number of words in all categories}}\right) \text{ which is smaller than any prob}(W, C)
\]

X.6 Understand torch output and output in general

I had some trouble understanding some outputs, mainly from Torch. This was solved by asking questions on the website forum[^30^]. Torch creates two files after training & testing, the_class_err and the_test_class_err. Both files only contained one number which looked like a percentage. It seemed that the_class_err shows the percentage of correctly classified files using the training set itself, and the_test_class_err contains the percentage of correctly classified files using the test set. Some other software, for example SNoW, do not produce much output and only give the final percentage of correctness.

X.7 Need of text classification

I am currently discussing with Corren to see if my software could be useful to them in a near future. Further developments would then be necessary, as stated below.

X.8 Future work

What I think could be an extension to this thesis would be to:
- test more software/algorithms (SVM)
- use more sophisticated pre-processing methods (feature selection/feature representation?)
- compare swedish <-> english texts accuracy
- combine the two “tricks” to improve accuracy, use a multiclassifier that would leave a file unclassified if at least 1 algorithm does not agree with the other(s)

If the product is to be used in a company, I have some ideas on how to improve the overall ergonomy:
- accept different format (.doc, .xml, ...), even some files inside a .zip
- add a graphical user interface
- classify physically instead of virtually (i.e. move files into separate category folders)
- leave some files unclassified if the results between some categories are too tight
- convert the software from java to C++ to try to reduce the processing time
XI Conclusion

XI.1 Classification itself

Throughout this paper it has been shown that document classification is possible with swedish texts. Of course, this had been experimented a long time before. Short documents (between 200 and 300 words) were found to be eligible for classification.

XI.2 Algorithms

Although all tested algorithms provide results which range from good to almost perfect, the reader must keep in mind that this work used very few feature selection methods. Thus the results shown in this report are valid only for the specific input and are not representative of the general accuracy of each algorithm.

Another important issue about feature selection is that, as seen in the fourth test, removing too many features could improve the processing time at the cost of reducing the accuracy.

To conclude this study it could be stated that the Naïve Bayes algorithm still plays a major role in the text classification field nowadays. It is especially used for spam recognition in Thunderbird[31] itself, or as a stand-alone software SpamBayes[32].

XI.3 Use of TC

I hope this study will prove itself valuable in the future, be it for enhancements or for a commercial use.

31 http://www.mozilla.org/products/thunderbird/
32 http://spambayes.sourceforge.net/
References


Parts of the book can be found at http://homepages.strath.ac.uk/~aep01236/3.3.htm


v S.C. BAGUI (University of West Florida), K.L. MEHRA (University of Alberta) & M.S. RAO (Osmania University) (1995). A nearest neighbour classification rule for multiple observations based on a sub-sample approach. USA.


vii N. Littlestone(1988). Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm, Department of Computer and Information Sciences, University of California,USA

This paper can be found at : http://ai.stanford.edu/~pabbeel/depth_qual/littlestone1988.pdf


xiii Thorsten Joachims (1998). Making Large-Scale SVM Learning Practical. Universität Dortmund, Germany

This paper can be found at : http://www.joachims.org/publications/joachims_98c.pdf


xv B.Liu, Y Dai, X.L. Wee Sun Lee & P.S. Yu (2003). Building Text Classified using positive and unlabeled text examples,

University of Illinois, National university of Singapore, IBM T.J. Watson Research Center.

This paper can be found at : http://www.cs.uic.edu/~liub/publications/ICDM-03.pdf

xvi Ian H. Witten & Eibe Frank (2000). Data Mining: Practical machine learning tools with Java
implementations, Morgan Kaufmann, San Francisco, CA.


The paper can be found at: http://www.idiap.ch/~bengio/publications/pdf/rr02-46.pdf

Juan José Garcia Adeva. Awacate: Towards a Framework for Intelligent Text Categorisation in Web Applications, School of Electrical and Information, University of Sydney, Australia.

J.R. Quilan, “C4.5 : programs for machine learning”, 1993


XIII Appendices

I. Stop word list:

common-words.txt

av
en
ett
vid
i
med
för
att
på
så
och
till
bredvid
de
är
om
två
inte
har
som
bara
jag
du
han
hon
vi
ni
det
den
de
II. Winnow tables

- Use words
- Don't take word number into account
- Learn once

2) Deterministic training (sport first, then economy)

<table>
<thead>
<tr>
<th>Category Parameters</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>119</td>
<td>1</td>
<td>0.55</td>
<td>28</td>
<td>0.22</td>
<td>0.22</td>
<td>147</td>
<td>60</td>
<td>&lt; 3mn34s</td>
</tr>
<tr>
<td>B</td>
<td>119</td>
<td>1</td>
<td>0.49</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>119</td>
<td>48.6</td>
<td>&lt; 2mn55s</td>
</tr>
<tr>
<td>C</td>
<td>119</td>
<td>1</td>
<td>0.52</td>
<td>14</td>
<td>0.11</td>
<td>0.11</td>
<td>133</td>
<td>54.3</td>
<td>&lt; 3mn5s</td>
</tr>
<tr>
<td>D</td>
<td>119</td>
<td>1</td>
<td>0.55</td>
<td>30</td>
<td>0.24</td>
<td>0.24</td>
<td>149</td>
<td>60.1</td>
<td>&lt; 2mn50s</td>
</tr>
<tr>
<td>E</td>
<td>119</td>
<td>1</td>
<td>0.56</td>
<td>31</td>
<td>0.25</td>
<td>0.25</td>
<td>160</td>
<td>65.3</td>
<td>&lt; 2mn45s</td>
</tr>
<tr>
<td>F</td>
<td>119</td>
<td>1</td>
<td>0.53</td>
<td>21</td>
<td>0.17</td>
<td>0.17</td>
<td>140</td>
<td>57.1</td>
<td>&lt; 2mn52s</td>
</tr>
</tbody>
</table>

The parameters ABCDEF are the same as 1).

A : positive weight = 2, negative weight = 1, promotion attribute = 3, demotion attribute = 0.3
B : positive weight = 2, negative weight = 0, promotion attribute = 3, demotion attribute = 0.3
C : positive weight = 1, negative weight = 1, promotion attribute = 3, demotion attribute = 0.3
D : positive weight = 3, negative weight = 0.5, promotion attribute = 3, demotion attribute = 0.3
E : positive weight =10, negative weight = 0.5, promotion attribute = 3, demotion attribute = 0.3
F : positive weight = 5, negative weight = 0.5, promotion attribute = 5, demotion attribute = 0.1

3) Category attached = sport
   Deterministic training (sport first, then economy)

<table>
<thead>
<tr>
<th>Category Parameters</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>119</td>
<td>1</td>
<td>0.51</td>
<td>10</td>
<td>0.08</td>
<td>0.08</td>
<td>129</td>
<td>52.7</td>
<td>&lt; 3mn32s</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>126</td>
<td>1</td>
<td>0.51</td>
<td>126</td>
<td>51.4</td>
<td>&lt; 3mn23s</td>
</tr>
<tr>
<td>C</td>
<td>119</td>
<td>1</td>
<td>0.49</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>119</td>
<td>48.6</td>
<td>&lt; 3mn27s</td>
</tr>
<tr>
<td>D</td>
<td>119</td>
<td>1</td>
<td>0.51</td>
<td>12</td>
<td>0.1</td>
<td>1</td>
<td>131</td>
<td>53.5</td>
<td>&lt; 3mn16s</td>
</tr>
<tr>
<td>E</td>
<td>119</td>
<td>1</td>
<td>0.52</td>
<td>18</td>
<td>0.14</td>
<td>1</td>
<td>137</td>
<td>55.9</td>
<td>&lt; 2mn50s</td>
</tr>
<tr>
<td>F</td>
<td>119</td>
<td>1</td>
<td>0.49</td>
<td>2</td>
<td>0.02</td>
<td>1</td>
<td>121</td>
<td>49.4</td>
<td>&lt; 2mn39s</td>
</tr>
</tbody>
</table>

A : positive weight = 2, negative weight = 1, promotion attribute = 3, demotion attribute = 0.3

78
B : positive weight = 2, negative weight = 0, promotion attribute = 3, demotion attribute = 0.3
C : positive weight = 1, negative weight = 1, promotion attribute = 3, demotion attribute = 0.3
D : positive weight = 3, negative weight = 0.5, promotion attribute = 3, demotion attribute = 0.3
E : positive weight = 10, negative weight = 0.5, promotion attribute = 3, demotion attribute = 0.3
F : positive weight = 5, negative weight = 0.5, promotion attribute = 5, demotion attribute = 0.1

4) Deterministic training (economy first, then sport)

<table>
<thead>
<tr>
<th>Category</th>
<th>Economy</th>
<th>R</th>
<th>P</th>
<th>Sport</th>
<th>R</th>
<th>P</th>
<th>Total</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>0.02</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>0.52</td>
<td>128</td>
<td>52.2</td>
<td>&lt; 3mn31s</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>126</td>
<td>1</td>
<td>0.51</td>
<td>126</td>
<td>51.4</td>
<td>&lt; 3mn</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>126</td>
<td>1</td>
<td>0.51</td>
<td>126</td>
<td>51.4</td>
<td>&lt; 3mn1s</td>
</tr>
<tr>
<td>D</td>
<td>10</td>
<td>0.08</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>0.54</td>
<td>136</td>
<td>55.5</td>
<td>&lt; 3mn6s</td>
</tr>
<tr>
<td>E</td>
<td>5</td>
<td>0.04</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>0.53</td>
<td>131</td>
<td>53.5</td>
<td>&lt; 3mn7s</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
<td>0.02</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>0.52</td>
<td>128</td>
<td>52.2</td>
<td>&lt; 2mn56s</td>
</tr>
</tbody>
</table>

A : positive weight = 2, negative weight = 1, promotion attribute = 3, demotion attribute = 0.3
B : positive weight = 2, negative weight = 0, promotion attribute = 3, demotion attribute = 0.3
C : positive weight = 1, negative weight = 1, promotion attribute = 3, demotion attribute = 0.3
D : positive weight = 3, negative weight = 0.5, promotion attribute = 3, demotion attribute = 0.3
E : positive weight = 10, negative weight = 0.5, promotion attribute = 3, demotion attribute = 0.3
F : positive weight = 5, negative weight = 0.5, promotion attribute = 5, demotion attribute = 0.1
III. Corren-elin.txt

This file show the relation between Corren category system and ELIN category system. It is organised as follow:

... 
corren_id corren_id_title 
elin_id elin_id_title 
...

If a Corren id does match a IPTC id, then the elin_id is set to 00000000 and there is no elin_id_title. The categories preceded by // are not taken into account, it is to avoid totally overlapping categories.

1mr1fqkfxe6w82 Tidningen 00000000
1mr1glp2015n39x Nyheter 00000000
1mr1h194a31562d Lokala nyheter 00000000
1mr1h88sjm3g8so Boxholm 00000000
1mr1hcd04yw7r7 Kinda 00000000
1mr1hejnjl1ebhm Linköping 00000000
1mr1hg04dz7d2j Mjölby 00000000
1mr1hnz8b6p8sn Motala 00000000
1mr1hoe4hadwimw Norrköping 00000000
1mr1hqd0o1233p7 Åtvidaberg 00000000
1mr1ht7tc7bgy4x Ödeshög 00000000
1mr1hvrz6kdvicz Östergötland 00000000
1mr8wl0439zf51t Vadstena 00000000
1mr1h4sq7tfj3p Inrikes 00000000
1mr1j6qsuj80twq Utrikes 00000000
1mr1j90bv33g8yx Sport och fritid 15000000 sport
//1mr1j90bv33g8yx Sport och fritid
Hundliv
animal
Konsument
consumers
Konsument
consumer issue
Matlust
gastronomy
food
Spel
game
Bokrea
currency values
EMU
international economic institution
Fråga forskare
Sommarcorren
Motor och trafik
automotive equipment
Jul
festo event (including carnival)
ynhetsbrev
nyhetsbrev
EU
Bostad
lifestyle (house and home)
In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/