Naive Bayes Classification of Swedish Forum Posts

Oliver Glant
Stockholm University
olgl1515@student.su.se

Abstract

In this project, a set of tools was developed for classification of Swedish forum posts on unknown subjects. Forum posts were retrieved from the Swedish forum Flashback. The posts were pre-processed and feature vectors were extracted and used to train Naive Bayes classifier.

The accuracy of the classifier was able to correctly classify between 37 and 73 percent of all posts, depending on which categories were included.

Taking into account that forum posts are often short and sometimes badly written, and that the tools developed are incapable of recognizing misspelled or compound words, the achieved accuracy is a clear indication that Naive Bayes is a viable method for this task.

1 Introduction

Automatic text classification is an important part in many of the applications of natural language processing, such as email spam filters and data mining, as it allows filtering on data based on user-defined categories.

The objective of automatic text classification is to predict which category a given text belongs to, which can be done in a number of different ways. Dalal and Zaveri (2011) outlines a generic strategy that divides the process into five discrete steps: pre-processing, feature extraction, model selection, classifier training and classification.

The aim of this project is to develop a simple but complete tool set to perform all of these steps and automatically classify Swedish forum posts according to topic using a Naive Bayes classifier. The tool set is aand demonstrate it using a limited set of posts from a Swedish web forum.

2 Data

This project uses forum posts retrieved from the Swedish forum Flashback (www.flashback.org). Seven subforums were arbitrarily selected and the text of all posts were retrieved and saved line by line in one .json-file per subforum using a web scraping tool adapted specifically for this project.

Subforum (id)	Posts
Candy, Fruit, Snacks (1)	13,575
Physics, Math, Technology (2)	22,838
National Socialism, Fascism, Nationalism (3)	41,350
Pets (4)	5,163
Roleplaying and Board Games (5)	4,912
Relationship Advice (6)	5,252
Cannabis (7)	33,792

Table 1: Number of posts retrieved per subforum.

2.1 Scraping

To retrieve the data used in the project a simple web scraping tool was developed. The tool uses the open source framework Scrapy and was written specifically to extract all posts of a Flashback subforum and outputting them to a .json-file.

The scraper (or spider) is implemented as a subclass of Scrapy's spider and contains three methods: 'start requests', 'parse threads' and 'parse', making up the simple algorithm shown in 'Algorithm 1'.

2.2 Extraction

After scraping, the text was extracted from the .json-files and initial refinement was performed to make sure the retrieved text matched the expected input format. This refinement consisted of the removal inter-punctuation and numbers, fixing of encoding issues and conversion of all characters to lower case. A combination of regular expressions and string methods proved to be the most re-

Algorithm 1 scraper

```
response ← web page
for thread in response do
    parse(thread)
procedure PARSE(thread)
    for post in page do
        Extract post
    if next page exists then
        parse(next page)
```

liable way. The extracted text was then saved in one .txt-file per subforum, with one post per line. Line numbers were used to keep track of individual posts through the whole project. At this stage, the forum posts in each category were also divided into a training part and a testing part, with a ratio of 9:1.

3 Method

The project is divided into several different modules containing one class each, each class more or less representing one step in the whole process. This means that the user can easily implement own solutions for different parts of the process without breaking the toolchain. The modular design also allows for comparisons of different methods for each step and simplifies improvements to the project.

3.1 Pre-processing

The pre-processing step was implemented as a class (preprocessor) with a primary method for word-by-word handling of input and output, and three methods to do the actual pre-processing: one function to tokenize the words, one to remove all stop words (using a list in a .txt-file, initially created by Diaz (2016)) and one to stem the remaining words. Stemming was done using the Snowball Stemmer module from NLTK (Bird et al., 2009), that provides simple stemming for Swedish words.

Algorithm 2 pre-processing

```
for post in file do
    tokenize(post)
    remove stopwords(post)
    for word in post do
        stem(word)
procedure TOKENIZE(post)
    for word in post do
        tokenized post ← word
procedure REMOVE STOPWORDS(post)
    for word in post do
        if word in stopwords then
            remove word
procedure STEM(word)
    stemmer.stem(word)
```

3.2 Feature Extraction

The first step of feature extraction was to index all the stemmed words to build a dictionary. To do this, the dictionary class was created, consisting of a Python dict-object and an indexing method taking an array of posts as input and indexing all words:

Algorithm 3 index words

```
for post in array do
    for word in post do
        if word not in post then
            dictionary ← word
```

Using the dictionary constructed by the algorithm above, the vectorizer class was implemented. It contains a dictionary and a method to convert a pre-processed post into an N-dimensional feature vector, represented by a one-dimensional compressed sparse row matrix from the SciPy-package (Jones et al., 2001), where each element represents the relative frequency of the word to which the index corresponds.

In the example below, the vector shown (in part) could represent a ten-word post where the word at index 1 (in the dictionary) occurs twice and the word and index 2 occurs once.

$$\begin{bmatrix} 0 & 0.2 & 0.1 & \dots & 0 \end{bmatrix}$$

The feature vectors are generated by the method 'vectorize', which takes a category of posts as input and determines all non-zero elements with the algorithm below:

Algorithm 4 vectorizer

```
1: Rows = []
2: Columns = []
3: Data = []
4: for post in category do
5:     post number + +
6:     length ← length of post
7:     for word in post do
8:         if word in dictionary then
9:             if word already found in post then
10:                frequency + 1/length
11:            else
12:                Columns ← word index
13:                Rows ← post number
14:                Data ← 1 / length
```

Using the row index for each post, column index for each word and the relative frequency (stored in the arrays Rows, Columns and Data) a sparse matrix is then generated, containing the feature vectors of all posts in the input category.

3.3 Model Selection

This project employs a Naive Bayes classifier (Using Multinomial Naive Bayes from the Scikit-Learn package), a technique that has been in use in machine learning for a long time (Lewis, 1998) and that is built on the assumption that all features (words in this case) occur independently of each other. The 'classifier' class was constructed, containing a Multinomial Naive Bayes (MultinomialNB) object from Scikit-learn (Pedregosa et al., 2011) and the methods 'train', 'evaluate' and 'classify' as well as a supporting method create the label array used by 'train' and 'evaluate'.

3.4 Classifier Training

The classifier was trained by inputting the training data from all subforums together with a parallell array containing the known labels of each post (the subforum from which it was retrieved) to the 'fit' method provided by Scikit-learn's MultinomialNB object.

When training the classifier, the 'evaluate' method is used to give an estimate of the hit-ratio of the predictions made by the classifier. The estimate is made by having the classifier predict the labels of the testing data (the 1/10th of the posts from every category that were not used in the training of the classifier) and comparing the predicted labels with the (known) correct labels.

3.5 Classification

With the classifier having been trained and evaluated with the training and testing data, classification is done by calling the 'predict' method of MultinomialNB, using a sparse matrix with vectorized posts of unknown categories as input, which returns an array with the predicted label of each post. The output array can be used in another program or saved to an output file (as done in the demonstration program).

3.6 Demonstration

In order to demonstrate the project, which is essentially a toolchain, a simple demonstration program, Hamlet, with a command line interface was developed. The demonstration program allows the user to test all steps from extracting the text from .json files to classifying unknown posts contained in a .txt-file.

4 Results and Evaluation

Evaluation was done by letting the classifier predict the category of the posts in the testing data (that is, 1/10 th of the forum posts from each category) and comparing the predicted labels with the correct labels. It is worth noting that the training and testing data are from the same source; this means that if posts on Flashback should tend to be written in a specific style, accuracy could be worse than indicated here if using the classifier to classify posts from other forums.

4.1 Accuracy

As the project aims to classify forum posts, the main result is the estimated accuracy, here measured as hit-ratio (number of correct classifications divided by number of classified posts) when evaluating the classifier with the training data. The results vary greatly depending on which categories are included, a few examples are shown in Table 2 (below). The accuracy is affected by several different factors. Presumably, data size and frequency of topic-specific words are two of the most important positive factors in this aspect, while number of categories and frequency of short, non-specific posts and bad spelling are negative factors.

4.2 Challenges

As the forum posts retrieved were often short, misspelled and informal, even manual classification was difficult in many cases. When studying some

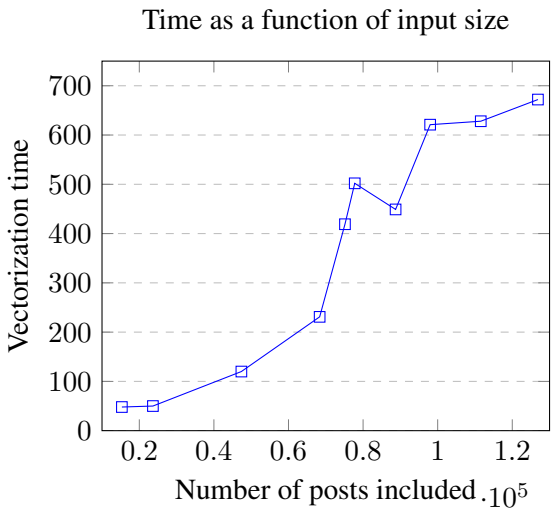
Categories included	Hit-ratio
All (1-7)	0.37
No small categories (1,2,3,7)	0.42
Three largest (2,3,7)	0.47
Three smallest (4,5,6)	0.50
Best hit-ratio (1,7)	0.73

Table 2: Results based on categories included.

of the wrongly classified posts, many of them contained compound words that were thought to be unique by the classifier but could have been clear category indicators if handled correctly. For example, several of the wrongly classified posts were about different kinds of chips, such as ”lökchips” (onion chips) and ”dillchips” (dill chips), and these were all interpreted as different, unrelated words. Likewise, misspelled words were also indexed as unique words. One way to solve these problems could be to check if substrings of new words or very similar strings (in the case of misspelled words) can be found in the dictionary.

4.3 Performance

As the tools in the project are intended for use in mining forum data it is assumed that time is not an important issue and little has been done to optimize the time-complexity of the tools. The most time-consuming part of the process is the vectorization step. The time taken for vectorization appears to increase linearly with data size (see graph below). Since vectorization is performed independently on all forum posts, the entire step could be parallelized and take a fraction of the current time (depending on system used).



As the feature vectors and matrices generated are very large, memory quickly turned out to be an issue. By using compressed sparse row matrices from SciPy, even the largest matrices were reduced to manageable size.

5 Conclusions

This project has implemented all steps of the generic automatic text classification strategy described by Dalal and Zaveri (2011): text preprocessing, feature extraction, model selection, classifier training and evaluation. Due to the limited scope of the project, most of the steps have only been implemented in their most basic form.

Given the challenges mentioned in classification of forum posts, and the relative simplicity in the implementation of many of the steps involved, the accuracy achieved indicates that using Naive Bayes to automatically classify forum posts is feasible.

References

Mita K. Dalal and Mukesh A. Zaveri. 2011. Automatic Text Classification: A Technical Review. *International Journal of Computer Applications*, 28(2):37–40. doi: 10.5120/3358-4633

David D Lewis. 1998. Naive (Bayes) at forty: The independence assumption in information retrieval. *Machine Learning: ECML-1998 - 10th European Conference on Machine Learning, Proceedings.* (Lecture Notes in Computer Science), 1998, 1398:4–15. doi: 10.1007/BFb0026666

Gene Diaz. 2016. Stopwords Swedish (SV). Retrieved from: https://github.com/stopwords-iso/stopwords-sv

Steven Bird, Edwayd Loper and Ewan Klein. 2009. Natural Language Processing with Python. O’Reilly Media Inc.

Eric Jones, Travis Oliphant, Pearu Peterson et al. 2001. SciPy: Open Source Scientific Tools for Python. http://www.scipy.org

F. Pedregosa et al. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 2011, 12:2825–2830.

Scrapy. Open source. https://scrapy.org/