

Machine Learning 2013: Project 3 - Text Classification Report

anufer@student.ethz.ch
elmerl@student.ethz.ch
nivo@student.ethz.ch

December 23, 2013

Experimental Protocol

Usage:

Download the csv files to /data/3/....csv (... = training, testing, validation)

Run data/fix_csvs.rb (Ruby), compile and run C# project, run code/map.m (Matlab)

Results are in /data/3/....out (... = training, testing_map, validation_map).

1 Tools

We used .NET to do the pre-processing (Create word bag, assign entries to word list, fix input data, etc) and Matlab to do the classification.

- Ruby (code is in /data/3)
- Visual Studio (C#, LINQ) (code is in /code/3_map/preprocessing/external_tools/ML-3 directory)
- Matlab (code is in /code/ directory)
- Git / Github Repository ¹

2 Algorithm

We used

- Some sort of edit distance (not the Levinsthein distance however)
- Maximum a posteriori (MAP) estimation

To classify the data we used the maximum a posteriori probability (MAP) estimate method presented in the tutorial session in week 49. We implemented the algorithm for multinomial distribution for a flexible number of k . The MAP is implemented in Matlab.

¹<https://github.com/lukaselmer/ethz-machine-learning>

3 Features

First, we used pre-pre-processing to clean up the invalid CSV files. For this, we used Ruby².

In our implementation, the pre-processing step consists of the preparation of the input data, the selection of the words to put in the *bag of words* of the Maximum a posteriori (MAP) estimation and the mapping of the input data to a mapping to the selected words.

To prepare the data, we got a list of the top 400 words in the entire training set and we replaced every other word that is similar enough to one of this top word. The similarity is calculated by the edit distance of two words divided by the length of the longer of the two words. Two words are considered similar if the similarity is greater than 0.75.

To group similar words together, preprocessing was used. First, we used the Levenshtein distance³, but then switched to a slightly modified version of the edit distance⁴, which gave slightly better results. Also, instead of using an absolute value threshold for the distance, we used a ratio of about 75% to group similar words together. For this preprocessing, we used C#, because string handling in C# seemed much easier than in Matlab.

Additionally, stemming was implemented in a very simple way: if the word is longer than 10 characters, the last three characters are ignored for the edit distance.

To fill the bag of words we group the training data by classifier and count the occurrence of each word. We take the top 25 words for each classifier if they appear in at least 10% of the training entries of this classifier.

To be usable for the MAP algorithm, each data entry is then mapped to a vector v with $v_i \in \{0, 1\}$. $v_i = 1$ indicates that a certain word i from the bag occurred in the data entry and $v_i = 0$ it doesn't.

4 Parameters

To find the parameters, we used manual testing. The most important feature seemed to be the edit distance ratio, which is 75%. Another important parameter is the amount of top words which are picked at the start of the algorithm. These are the words, which are in a category of it's own before grouping them together.

The MAP algorithm has one parameter α : $\alpha_i \in \{1, k\} = 1.01$.

5 Lessons Learned

Many other tools and algorithms have been tried. However, one we didn't try and might have worked well is SVM.

What we also learned is that string handling is painful in Matlab, especially when compared to the C#'s LINQ queries. It therefore may be a good idea to use different technologies for a project, so each technology's strengths can be used.

²<https://www.ruby-lang.org/>

³https://en.wikipedia.org/wiki/Levenshtein_distance

⁴https://en.wikipedia.org/wiki/Edit_distance