FOLT Software Project

Luis Dreher

Jan Buchmann

1 Project Documentation

The 2019/20 FOLT software project consisted of two tasks that work with the same data. This data is a set of comments from "Wikipedia talk page edits" (The FOLT 2019/20 Practice Class Organizers, 2020). In these comments, Wikipedia users discuss edits of Wikipedia articles. The first task was a *shared task*, in which the comments were to be classified as *toxic* or *non-toxic* based on the language used. The classification was evaluated with the *accuracy* measure, which gives the proportion of correctly classified comments.

For the classification, one of the classifiers provided in the natural language toolkit (NLTK) (Bird et al., 2009) was to be used. We decided to use the naive Bayes classifier, as this is a well-established tool for this purpose. In the following, all mentions of a "classifier" refer to this specific classifier. The most well-known application of naive Bayes classifiers in text classification is spam e-mail detection (Sahami et al., 1998), which is a somewhat similar task to the one described here.

During the training, the classifier is supposed to "learn" specific properties of the comments that distinguish toxic from non-toxic comments. These properties are often called *features*. Such features can be the presence / absence of specific words or whether a text has fewer or more words than a specific threshold, and many others. During experimentation with the NLTK naive Bayes classifier, we noted that it is not able to handle numerical features adequately. This is illustrated by the following example: An NLTK naive Bayes classifier is trained with four example comments and the word count of each comment as the only feature. Given the following distribution of word counts: (C1, T, WC=50), (C2, NT, WC=70), (C3, T, WC=47), (C4, NT, WC=73), where CX is the comment id, T/NT is toxic/non-toxic and WC=Y is the word count, one would expect that the classifier learns that shorter comments indicate toxicity and vice versa. However, in reality, it only learns that each specific word count indicates a certain class, and views a different word count as a new feature. The main challenge of the first task was to identify those features that are most conclusive for classification (i.e. most specific to a class).

To train the classifier, a part of the data (the train split, consisting of 1800 comments) was provided with a *toxicity* label for each comment. These labels had been obtained by human annotation. Inspection of the train data revealed that 877 comments were labeled as toxic and 923 were labeled as non-toxic. This means that the dataset is fairly balanced in terms of class distribution. A non-balanced class distribution can cause problems, because this might mean that there are not enough examples to learn informative features for some of the classes. The average number of words per comment was also analyzed, which resulted in 61.12 words for toxic comments and 83.13 words for non-toxic comments. From this a feature was created which indicates whether a comment has fewer or more than 72 words. Further, it was observed that toxic comments have a higher proportion of punctuation symbols (3.91%) than non-toxic comments (3.46%). Accordingly, a feature was created indicating whether the proportion of punctuation symbols in a comment is higher or lower than 3.685%. A closer look at the data suggested that main discussion topics are politics, religion and women / minority rights.

The train data was split in a train set of 1200 comments and a development set of 600 comments to be able to estimate the classifier's performance

To find individual word features, two approaches

were tested. In the first, the most common tokens (with and without punctuation) were used as features. In the second, a classifier was trained on the complete train dataset using all tokens lowercased as features. The "most_informative_features"-attribute of the classifier was used to get the most informative tokens. Then, different classifiers were trained on the train set, using different parts of the list of most informative tokens as features. The classifiers were evaluated on the development set. The best performance was achieved with the 9000 most informative tokens.
All combinations of the features described above were used to train and evaluate classifiers. The best accuracy was yielded using the 9000 most informative token features.
To classify the test data, a classifier was trained with the 9000 most informative token features, using all of the labeled data provided as train data (test set + development set). The same features were then extracted for the test data comments and these were then classified.

The second task was to *augment* the comments to mitigate *gender bias* in the data. It is well-established that such bias is easily picked up unintentionally by NLP algorithms, because they learn from real-world data (Lu et al., 2018). Lu et al. (2018) proposed a data augmentation strategy to overcome this problem.
We adopted parts of their approach. In particular, they proposed to duplicate the dataset at hand, and invert gender-related words in the duplicate (Lu et al., 2018). For this, they used a bidirectional dictionary of 124 word pairs, in which replacements for gendered English words can be found. Examples are "waiter" - "waitress" or "monks" - "nuns". We obtained their list of word pairs[1] and applied it in the same way. For the replacement of "him" / "his" and "her", the authors propose a slightly advanced approach, which uses the part-of-speech tag (POS tag) of a token for correct replacement (Lu et al., 2018). We also applied this method.

2 Results Analysis

References

Steven Bird, Edward Loper, and Ewan Klein. 2009. *Natural Language Processing with Python*. O'Reilly Media Inc.

Kaiji Lu, Piotr Mardziel, Fangjing Wu, Preetam Amancharla, and Anupam Datta. 2018. Gender Bias in Neural Natural Language Processing.

M. Sahami, S. Dumais, D. Heckermann, and E. Horvitz. 1998. A bayesian approach to filtering junk e-mail. In *Technical Report WS-98-05*, pages 98–105. The AAAI Press.

The FOLT 2019/20 Practice Class Organizers. 2020. FOLT Software Project Task Description.

[1]The list can be found in the supplement of (Lu et al., 2018)