Programming Assignment 2A

Authors

John Lindblad
guslinjohg@student.gu.se

Olle Lindgren
olleli@student.chalmers.se

Abstract

The aim of this project is to train a classifier that can be used to classify comments as pro or anti brexit. The learning problem is formulated as supervised learning and the training and test data have been collected using crowdsourcing. After training a range of models using different learning algorithms and comparing them based on classification accuracy on the test data, we found some Neural Network models to be the best in terms of performance. However, their advantage in terms of accuracy was not remarkably large and it could be discussed if they are the best choice base given the time required for training.

Introduction

The project described in this report aim to create a text classifier that classifies comments about brexit as either pro-brexit or anti-brexit. The task was to train a classifier using some annotated and crowdsourced data. This means that the problem was given as a supervised learning problem.

Method description

This section will describe the methods used in different steps of the project, from data collection to testing on the final test set.

Data collection and annotation

The data was collected and annotated using a crowdsourcing method. We obtained a training dataset with 13 517 comments about brexit and the corresponding annotations for each comment. Each comment had between 1 and 15 annotations. We chose to exclude all comments where the agreement was not 100% which gives 7703 comments. We initially tried to train the model with all comments where there was not a tie or where -1 (not able to determine) was most common but this actually resulted in lower accuracy and it yielded only 7966 comments, so no big difference.

Preprocessing of data

The data was preprocessed quite extensively. First, all the input data was put into a Pandas DataFrame with the columns *positive*, indicating the opinion of the comment and *text*, the actual comment. After that, a few columns were added in order to lower the number of words, and thus the number of features in the final TF-IDF vectors. The following columns were created:

lowercase: The contents of *text*, converted to all lowercase. *cleaned*: The contents of *lowercase*, with everything but letters and numbers removed. *after_replacement_s1*: The contents of *cleaned*, but with very similar words replaced with the most common. Very similar words were words that either appeared as synonyms in PyDictionary, or that had the same sequence of letters, like for example like *borris*, *boris* and *boriiiss*. *after_replacement_s2*: The contents of *after_replacement_s1*, with all words that occurred only once deleted. This was done since these words were unlikely to be useful in training. *after_replacement_s3*: The contents of *after_replacement_s2*, with the 100 most common words removed. This was done since these words are equally unlikely to be helpful.

The total number of unique words were around 18000 in *lowercase*, and 10000 in *after_replacement_s2*. In order to preserve some of the information lost when computing these columns, four additional binary columns were added:

has_hashtag: whether or not there was a hashtag in the original comment. *is_long*: whether or not the original comment was longer than average. *mentions_scotland*: whether or not the strings *scotl* or *scotti* appear in the comment. *cleanness*: whether or not the stuff removed between *lowercase* and *cleaned* was more than average.

After processing, a TF-IDF vectorization was done to the column *after_replacement_s2* using the TfidfVectorizer from scikit-learn and converted to a 2D numpy.array. The columns *has_hashtag*, *is_long*, *mentions_scotland* and *cleanliness* were then added to this array.

The column *positive* was also converted into a *numpy.array*.

These two arrays were then used as X and y variables.

Model training

A couple of pre-built models from *scikit-learn* were trained and parameter-optimized on the dataset. The ones chosen for further testing based on good performance or interesting comparison was the following:

- LinearSVC
- RandomForestClassifier
- Perceptron
- Dummy Classifier
- LogisticRegression

In addition to this, five neural networks, built using keras with the tensorflow backend, were also trained on the dataset. For the training of these, the training dataset was split into a training and validation dataset, with 20% being used for validation. The number of epochs were initially set to 30 because of the slow training time.

In training, these all tended to quickly overfit, reaching an accuracy of over 99% on the training dataset by the 4th epoch but with the validation accuracy slowly dropping from a peak of around 77-80% in the first epoch. Therefore, an EarlyStop from keras was added, monitoring the *val_loss* for any declines with a patience of 4 epochs. The number of epochs were now increased to 100, but that was never reached.

The networks were later all re-trained with only one epoch. For this re-training, the training/validation split was changed from 8020 to 9010. With this setup, the validation accuracy increased by 3% to 81.73% on our best network, affectionately named *model 2*. The code for *model 2* is provided in the accompanying Jupyter notebook.

The general idea of the model is that the network would be able to generalize step by step. If this is how the model works is impossible to know but it seems to work quite well. The number of layers and the size of the layers were made as small as possible in order to keep the number of parameters low. We did not experiment with any other types of layers than keras.layers.Dense, mostly because of time constraints.

At the last lecture, we were taught that softmax and relu are the most commonly used activation functions, but the initial networks that we tried did not converge with *softmax* so we instead used the *sigmoid* activation function. No internal activation functions other than relu made the model converge.

The neural networks were not tested on the TF-IDF vectorization of the full dataset as it looked before all the word replacements, because the training time was too long.

Model testing

The best performing of the models above (accuracy above 70% on the validation dataset) are the ones presented below. 20% of the training dataset was set aside for validation:

Algorithm	Accuracy
Linear SVC	78.33%
Random Forest	77.87%
Logistic Regression	77.96%
Dummy (most common) classifier	52.15%
Perceptron	73.69%

Table 1: Accuracy on the training data of some pre-built models from scikit-learn

However, the Neural Network models seem to perform slightly better. The following results (Table 2 and Table 3) are obtained from testing them on the final test data, where

- TPR = True Positive Rate
- TNR = True Negative Rate
- PPV = Positive Prediction Value
- NPV = Negative Prediction Value

	Accuracy	TPR	TNR
Model 2	0.790517	0.821429	0.755515
Model 4	0.793966	0.811688	0.773897
Model 5	0.787069	0.855519	0.709559
Model 6	0.793103	0.808442	0.775735
Model 7	0.792241	0.837662	0.740809

Table 2: Performance scores on the test data of some Neural Network models

	PPV	NPV	F1 score
Model 2	0.791862	0.788868	0.806375
Model 4	0.802568	0.783985	0.807103
Model 5	0.769343	0.812632	0.810146
Model 6	0.803226	0.781481	0.805825
Model 7	0.785388	0.801193	0.810683

Table 3: Performance scores on the test data of some Neural Network models

Conclusion and discussion

This section includes a brief discussion and some conclusions.

Preprocessing

We tested some different ways to preprocess the data with varying results. The lengthy one used and described above was chosen because it reduces the number of features a lot, which reduces complexity and makes it practically possible for us to train the neural networks.

Model complexity and training time

The Neural Network models performed better as shown above. However, one could still discuss the choice of model. If we use Occam's razor as the basis for the model selection one could still argue that the simpler models such as the SVC could sometimes be a better choice. The main problems with the Neural Network models are the time required for training and also later on for prediction.