CS412 Turkish Tweets Sentiment Analysis Project

Ahmet Faruk Yapar, Erdem Bozkurt, Mehmet Ozan Servet, Veli Can Erdem

In this project, we are asked to implement a machine learning algorithm to determine the score of a tweet related to its positiveness and negativeness. The scores are given in the range of [-1,1].

Our first approach was to try various algorithms for classification problems in Weka to get a base success rate. After our tests we found out that ZeroR resulted in %24 rate, which is very similar to the other algorithms tested, such as Random Forest, Naive Bayes etc.

In the given data there are 757 train tweets and 200 test tweets. There are 3671 distinct words and 2110 words when words are reduced to 5 letters and 1052 words when words are reduced to 3 letters. After a brief research we decided to try a few algorithms. We used python 3.6 to try our algorithms, requirements needed to run given source codes are in requirements.txt .

We first tried Naïve Bayesian approach. For any word positive, we added both the sentence and the word as same scores. For example, if a sentence is scored 0.1, this would correspond to 0.1 instances of “Yes”. This resulted in 0.596 error rate with 3 letters, %30 linear error rate and with 5 letters 0.497 error rate which is about %25 linear error rate.

Then we tried averaging over the computed average scores of words in a tweet. Data has 9445 words and about 3671 of them are distinct, resulting in 2.57 words in average. This makes it harder to put the words as inputs to a neural network. There are certain words that are very likely to be negative or positive and averaging works well within this regression problem. This approach resulted in 0.334 linear error with 3 length words when any word that didn't appear in dictionary has 0 score. 0.325 linear error with 5 length words. Smoothing the words themselves resulted in higher error in this approach.

Then we used a multilayer perceptron such that it would take the given features and the Averaging approach's outputs as inputs. We tried various number of hidden layers and various number of hidden nodes. This resulted in 0.35 error at best when hidden layer of (30,20) is used, which is %17.5 error rate.

We tried various size of hidden layers, for example (100, n) which proved too large and (10, n) which proved too small.

We also tried having words as binary inputs to the neural network. We took the first 100 words as additional inputs. This approach resulted in 0.35 error, which is the same as before. There were no improvements when we increased the input dimension. We also expanded the size of the neural network but still the improvement was negligible.

- We used tanh as the activation function. It proved better than the sigmoid function for our network, which had about 0.4 error. (%20)
- We tried various learning rates, for example a = 0.1 and a = 0.01, but results didn’t change much except for the very large changes.
- We tried adding words as features (100 most occuring words, 4000 words etc., but this resulted in decline of performance.
- We tried cross-validation methods, which increased performance by to 0.24 error rate (within %12 of actual value)

Python file main_bayes.py contains the Naïve Bayesian approach, and main_ann.py contains the Multilayer Perceptron approach. Guesses of Neural Network is given in resultNN.py . main_ann_cv.py has the cross-validation.