Mini Project 2 – Project Classification

Group 25

Catarina Sousa 93695

Nelson Trindade 93743

Models:

* Our first model is a machine learning algorithm (Naïve Bayes). We start by parsing the input files to have labels and questions separated. We then pre-process the questions by *tokenizing*, *lowing* the words, *stemming*, and *lemmatizing*. At the end of our pre-process, we have a list with all the sentences. After processing, we use *Count Vectorizer* to *fit\_transform* our train set so that our model learns the mean and variance of the features of our set. We then *transform* our test set so that the model uses the same mean and variance that was calculated from the train set to transform the test set. We then use a Multinomial Naïve Bayes distribution since it is suitable for the classification of discrete features.
* Our second model is a Support Vector Machine approach. We parse just as we do in our first model, but then our pre-process is just a tokenizer. We also use the *Count Vectorizer* to *fit\_transform* our train set and *transform* our test set. After this, we use *tfidfTransformer* (it has nothing to do with transformers, just the name) to *fit\_transform* the training matrix and *transform* the testing matrix obtained by the *Count Vectorizer*. These functions transform the count matrix into a normalized *tf-idf* representation. We then use a Support Vector Machine to predict the test labels.

We started by doing small steps each time in order to be sure that what was done was correct. When we created our parser function, we tested it. When we created our processing function, we tested it. We did this technique in order to catch the errors as soon as possible. After having these functions functional, we started testing different approaches.

In the Naïve Bayes approach, we tested different distributions until we concluded that the Multinomial was the best. We tested with different pre-processing techniques until we reached the best one with this model (86.8%).

In the Support Vector Machine approach, we also tested with different pre-processing techniques until we realized that using only the tokenizer was the best. We first didn’t use the *tfidfTransformer* functions and obtained an accuracy of 84%. We started thinking about how we could improve our model and we decided that we wanted to use TF-IDF, since it reflects how important a word is in a corpus. We concluded that with TF-IDF we could get better results and that was what happened, we got 88.4%.

The *tokenizer*, the *stemmer* and the *lemmatizer* that we use are from the *nltk* library, while the *MultinomialNB*, *Count* *Vectorizer* and *tfidfTransformer* are from the *sklearn* library.

To get the accuracy from our models, we compared the predicted labels with the correct labels using the *metrics.accuracy\_score*.

Results: