Team Name: QGD
Members: Qi Shi 20210981; Gabriel Avezum 20210663; Danilo Arfeli number: 20211296

1. Data exploration

After loading the dataset, we found that the dataset has only two columns, 'sentence' and 'emotion'. The 'sentence' column has one sentence in each record and will be used as our predictor variable that will pass through some modification (extraction) later. The 'emotion' is the target of our classification task and contains seven classifications: 1: Anger, 2: Anticipation, 3: Disgust, 4: Fear, 5: Joy, 6: Sadness, 7: Surprise, 8: Trust.

There are no missing values in the dataset. Since it is a multi-classification task, we have counted the record number of different emotion categories (Fig1.). The results show that the record number of different emotion categories is unbalanced, in which the Anger category is significantly higher than other categories. This indicates that for the final evaluation of the prediction model we will have to take the unbalancedness into consideration and in this sense the overall f1 score is a better metric than the accuracy. Although the SMOTE method is a great approach to balance the data, we still have not decided to use it, since it will easily lead to the overfitting of the final model.

In daily life, we can notice that people seem to speak shorter and stronger when they are angry and would say more things when they are happy. Therefore, it is interesting if the sentence's length has some relation to the emotion. We counted the number of words in sentences of different emotions. However, the results turn out to be that there is no significant difference which suggests that The sentence's length provides little valuable information for emotion classification. (Fig2.)

Fig1.Number of records in different emotions

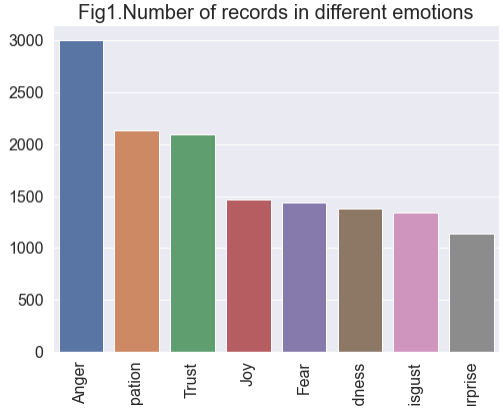
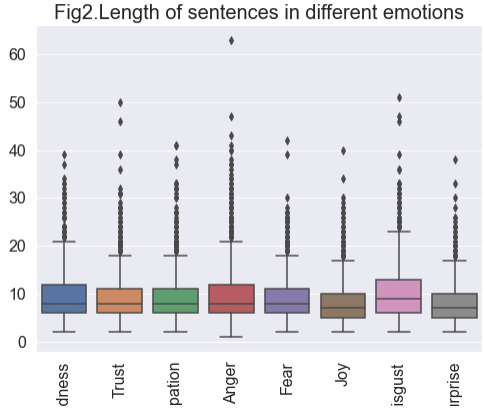


Fig2.Length of sentences in different emotions



2. Preprocessing

Preprocessing is a fundamental step in building a machine learning algorithm for natural language processing because data can be noisy or (and) inconsistent and it can cause unnecessarily high dimensionality vectors, computationally expensive and can potentially get

worse performance.

The dataset was splitted in train(70%) and test(30%) and all the preprocessing was done in the training data and applied in the test. In this project we considered the English language from *SnowballStemmer* and it gave us the possibility to extract a list of stop words and punctuation to test if removing it or not would improve the performance of our model. Besides stop words and punctuation, it was tested on lowercasing, lemmatization and stemmer to improve the performance.

To transform the text into important features we tested CountVectorizer, TfidfVectorizer and FastText. The CountVectorizer converts a collection of text documents to a matrix of token counts and the number of features will be equal to the vocabulary size found by analyzing the data. In TfidfVectorizer that Transform a count matrix to a normalized tf or tf-idf representation, Tf means term-frequency while tf-idf means term-frequency times inverse document-frequency, the goal of using tf-idf instead of the raw frequencies of occurrence of a token in a given document is to scale down the impact of tokens that occur very frequently in a given corpus and that are hence empirically less informative than features that occur in a small fraction of the training corpus. And FastText which is a library created by the Facebook Research Team for efficient learning of word representations and sentence classification and assumes a word to be formed by a n-grams of character.

Word embeddings as Word2Vec, which is a neural network structure to generate word embedding by training the model on a supervised classification problem used to measure semantic and syntactic similarities between words, was also tested.

3. Model

First we tried to find a baseline approach where this model would give a result better than the stronger baseline presented on the project guidelines. To find this first model we still used the SVM model with a linear kernel but we changed the preprocessing of the sentences until we got a better result. Was found that not removing the stopwords and using lemmatize gave the following table:

	precision	recall	f1-score	support
Anger	0.61	0.35	0.44	369
Anticipation	0.51	0.44	0.48	196
Disgust	0.09	0.22	0.13	32
Fear	0.24	0.37	0.29	68
Joy	0.38	0.45	0.41	82
Sadness	0.29	0.37	0.32	67
Surprise	0.18	0.39	0.24	44
Trust	0.36	0.40	0.38	142
accuracy			0.38	1000
macro avg	0.33	0.37	0.34	1000
weighted avg	0.45	0.38	0.40	1000

Figure 3.1 - table with metrics obtained on the development set by our baseline

After this first model we tried using word embedding with word2vec and fasttext. So, comparing the both and trying different models we got a better result with fasttext and SVM model with the rbf kernel and C equals 0.9, the results as shown below:

	precision	recall	f1-score	support
Anger	0.65	0.40	0.49	346
Anticipation	0.49	0.44	0.46	187
Disgust	0.22	0.52	0.31	33
Fear	0.23	0.46	0.31	52
Joy	0.45	0.47	0.46	94
Sadness	0.37	0.41	0.39	79
Surprise	0.24	0.41	0.30	56
Trust	0.44	0.46	0.45	153
accuracy			0.43	1000
macro avg	0.39	0.44	0.40	1000
weighted avg	0.49	0.43	0.44	1000

Figure 3.2 - table with metrics obtained on the development set by our best model

Comparing the both models we can conclude that only the two classes with more observations didn't improve (Anger and Anticipation), but we got a better result in all other classes, also the first model got an overfitting because the train had an accuracy of 0.72 but the best model returned an accuracy of 0.48 on the train. The most common mistake happens when the class has a small sample and when we increase the samples the metrics improve, so it shows that increasing the train/dev set should be a good way to improve the results.

4. Future Work

Several approaches were also applied as a bidirectional LSTM model, and the best accuracy can only reach 35%, based on the basic embedding layer. For the emotion, there would be more potential patterns in the order and combination of words. Currently we only simply add the word vectors of each word in sentences. So for future work, it is suggested to work more on word embedding. An approach under consideration is that in sentences each different word vector will receive a weight and summed together. The weight would be trained by a bidirectional LSTM network. From our knowledge, we know that punctuation is very important to emotion in this way the punctuation could be extracted as a feature and preprocessed independently, then stacked into the final model.