Natural Language 2023/24 - MP2 - Group 6

95629 Maria Campos; 95682 Tomás Pereira

Models

After analysing the given task, we decided on the following pre-processing techniques for the dataset, taking unigrams and bigrams into account: **Lemmatization**, to standardise the reviews due to the high variety of words the dataset's vocabulary has; **Lowercasing**, due to us considering that it is more important to establish similar expressions rather than one-off cases where reviews might have more sentimentality through the use of uppercase words/sentences; **Stop Words**, using a relatively simple list obtained through the NLTK library. We kept the words [“no”, “not”, “as”, “but”] due to their importance as indicators for negation, comparison and adversity, and also added some other stray stop words not in the initial list; **Accent Stripping,** to uniformize the features; And finally, **TF-IDF Vectorization**, to format the features into vector form to be used as input for the classifier models. Later on, we stopped using IDF as it yielded slightly worse results.

Overall, three models were developed: a **Naive-Bayes Classifier** [1], a **Support Vector Machine Classifier** [2] and lastly a custom **Neural Network** [3].

The first two were picked to compare our pre-processing performance to the task's two baselines: 58.5% and 80% accuracy, for a NBC and a SVC, respectively. Our models outperformed the baselines by a lot, as will be seen in the Experimental Results section.

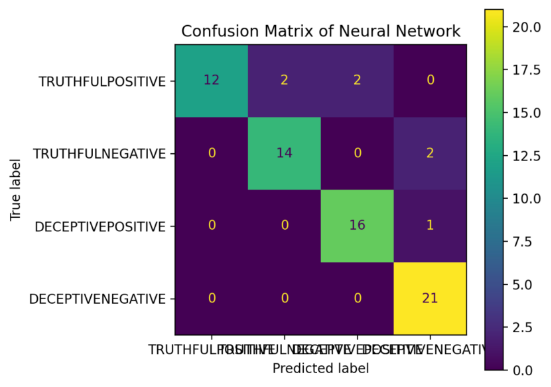
Initially, for the Artificial Neural Network, we attempted the implementation of Recurrent and Transformer Neural Networks , but those failed to converge, possibly due to the lack of enough data to properly train such complex systems as these.

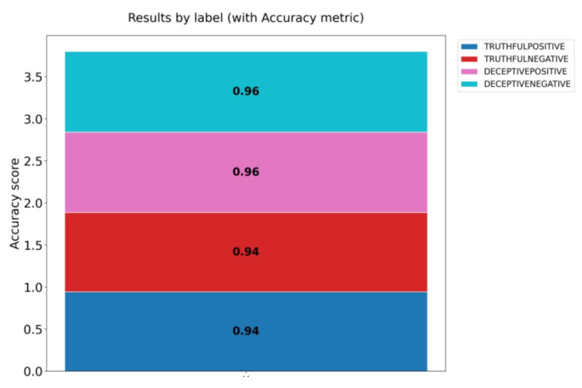
So, instead, we opted to customise and optimise a simpler classifier Neural Network. Taking [3] as a base example, we utilised the TensorFlow Keras library for the development and configuration of a multi-layer perceptron neural network. After many attempts and hyperparameter optimisation, we decided on a NN that ran on 4 layers: a **tanh** input layer, a **sigmoid** and **tanh** hidden layers and finally a **softmax** output layer. The NN was compiled with the default **adam** optimiser configuration alongside a **cosine similarity** loss function.

Interestingly, the NN performed better with stop words, so we kept them in its features. In the end, this model had the best performance, so it will be the one we focus on.

Experimental Setup Results

To train our Neural Network Classifier, we utilised a **batch size of 8** samples per gradient for **12 training epochs.** The input layer and the hidden layers of the NN all had **16 perceptrons.** These values were found to give good results and avoided overfitting. Here's an example of a run with **90% Accuracy,** on a **test set the size of 5%** of the given corpus:





When **model averaged** on **50 differently trained models**, our classifiers scored the following accuracy results: 79.39% for the NBC, 88,57% for the SVC and 94,29% for the NN.

Discussion

Given the problem's context, we noticed that our models performed a lot better with bigger train set sizes, more specifically, 90-95% of the given dataset. We concluded that for the complexity of this problem, the dataset was too small of a size and training greatly benefitted from the additional entries, instead of being led to overfitting.

By analysing some of the examples incorrectly labelled, we can infer that our NN fails in aspects of **context sensitivity**, most prominently, misinterpreting negation as a negative indicator while the real goal of the sentence is positive-ended. Examples are expressions such as "no problem" or "not a big deal" (when referring to something that did not cause bother), which our chosen model interpreted as a negative indicator, even though those expressions typically denote something neutral or even positive. A concrete example of this was with the dataset's 18th entry, where a negative fact is first introduced: "*[the hotel] is bit stuffy and a little outdated for the price*", but later downplayed and neutralised with the statement: "*You are not going to be spending a bunch of time in your room when visiting Chicago, so the room's size is not a big deal*". Our Neural Network in this case weighed the review as more negative than positive.

Other cases of incorrect classification fall under the umbrella of incorrectly labelling TRUTHFUL statements as DECEPTIVE. As this issue doesn't happen for the reverse situation (mislabelling DECEPTIVE statements as TRUTHFUL), we can safely state that this was due to our model's more pessimistic approach in regard to the classification of TRUTHFUL statements. Even as a human, it's very difficult to distinguish cases between these two categories of labels, so we suspect this result is a consequence of our Neural Network being biased towards the DECEPTIVE label in cases where there is a close call between both of the categories. Some of these mislabelling cases were found in entries 26, 40 and 57, which are non-generic reviews with either lots of expressive phrasing combined with idiomatic terms or more of a story-telling structure. Perfect accuracy will always be a near-impossible goal for classification tasks such as this one, but more nuance and attention to detail could probably be added if the dataset had even more examples similar to these.

Future Work

If more time was given to develop this project, we could have attempted combining our classifier with a pre-trained model such as **BERT**. In this case, BERT would act as another step of the pre-processing of the features. This could prove to be useful in one of the bigger issues we found, explicitly, in the interpretation of how negation is placed and utilised in sentences to make the model more context sensitive. This is possible due to BERT's self-attention layers, which enable the encoding of the context of words to a certain degree.

Bibliography

[1]https://www.toptal.com/machine-learning/nlp-tutorial-text-classification
[2]https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/
[3]https://www.analyticsvidhya.com/blog/2021/10/implementing-artificial-neural-networkclassification-in-python-from-scratch/