Computational Argumentation

Aim of the work:

The goals of this assignment are:

a.) Work on the argumentative dataset.
b.) Identify the features from text to characterize claims.
c.) Train a Classifier to detect the claims automatically.

Task 1: Feature Engineering

To identify the claims for a given sentence, we need to get the features that helps characterizing the claims using text.

In order to do that, fist we have imported both training and validation datasets and converted to a data frame. The data frame has the columns id, text and labels.

Next step, Data pre-processing like converting the text to a lower case and removing or correcting the contractions.

Features:

1. Sentiment - The Sentiment tells whether the given sentence contains any emotions like whether the statement express a positive emotion, negative emotion and neutral.
   The sentiment has 2 parameters –
   a) Polarity – The polarity assigns the sentence values ranging from -1 to 1. If the statement shows positive emotion value will be between -1 to 0. 0 represents neutral emotion. If the polarity value is between 0 and 1 then it is a positive emotion.
   b) Subjectivity – Subjectivity shows whether the statement is a public opinion and not just an information which conveys some real facts.

   In our approach we are taking the subjectivity of a sentiment which helps finding the claim texts from a set of sentences.

2. Characters and word Count - This feature just take the total count of each and every character and words, including white spaces from a statement. This feature is useful in separating the non-claims from a set of sentences. Most of the claims are lengthy, so by getting length we can identify certain claims without even digging deep into it.

3. Beginning of a sentence – By analysing the dataset, we can find out that the nonclaims starts with various special characters like ‘$ ‘,’&’,’’,’#’ and so many other special characters. Hence, we can characterize claims by using these are features.

4. Parts of Speech tagging – Parts of speech tagging or grammatical tagging is nothing but assigning each word in a sentence part of speech. To identify whether the words are Nouns, Verbs, and Adjectives etc. In our approach we have used mainly nouns to detect claim or not. We are counting the number of nouns to identify.

5. Special characters – After analysing the dataset, we can witness the claims has a special characters which are “”, the claims have been quoted with double quotes and the non-claims contains more unwanted special characters distributed randomly for example: #, $, & and [] etc. In our approach we are considering the special characters like ‘#’, ‘&’, ‘?’, ‘!’ and ‘$’ which are present multiple times in the non-claim sentences.

   Our logic behind this is most of the claims does not end with a’?’ or’!’ and also neither it starts with &, $ or #. Also the **http** links cannot be considered as a claim, hence we are using it as a feature to separate statements mandatorily from claims.

6. TF-IDF – The Term Frequency – Inverse Document Frequency vectorizer transforms tokens in a text to a feature vectors, this is calculated using the word repeated number of times in a sentence to total number of words in a sentence. The resulting score is appended to a feature word. We have used total of 350 words to get the better accuracy depending on the dataset we need to tune the max_features parameter.

Task 2: Classification

After the feature selection and the data frame is ready with the only numerical data or the implicitly convertible Boolean values, it is then fed into a classifier to classify whether a sentence is a claim or not.

Here, we have total 2 classes to classify, claim and non-claim. We need to train the classifier model using the training data set by selecting the necessary features including the target feature that is a **label**. Later the validation dataset or the test set which has the similar features is fed into a classifier without target feature to predict the classes.

Classifier:

To choose a best classifier to train, we have decided to use **Logistic Regression**.

This classifier is mainly used for most of the text classification and especially for binary classification task. Also it is used when number of features are more. This mainly work on finding relationship between features and predicts particular outcome. We choose Logistic Regression because it is used when the target variable is dependent on the categorical. We have more number of features so it fits correctly without under fitting.

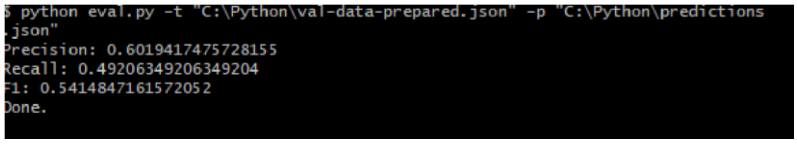
We have also tried other classifiers and compared the scores with our features, and we found the Logistic Regression best getting score of 54% f1 with 75% training accuracy as well as 70% validation data accuracy.

```
Hyperparameter tuned: {'C': 163789.3706954068}
Best score is
              precision    recall  f1-score   support

   Non claim       0.74      0.82      0.78       223
       Claim       0.60      0.49      0.54       126

    accuracy                           0.70       349
   macro avg       0.67      0.65      0.66       349
weighted avg       0.69      0.70      0.69       349
```

We performed the grid search to tune the performance using the hyper parameters and we achieved the best score.

After getting the prediction scores we transformed it into data frame and converted to a given output JSON format.

Finally we did run the eval.py script to verify our F1 score to achieved Baselines and also to know the output format is correct and we got the below positive results.



Steps to execute our code

1.) Download our Group_3_Assignment_2 zip file and unzip it.

2.) Libraries to be downloaded are pandas, numpy, nltk, sklearn, and textblob.

3.) Open the Assignment_2_Group_3.ipynb file on Jupyter Notebook.

4.) Inside the ClaimDataDF json read, give a train-data-prepared.json path and inside the ValDataDF json read the val-data-prepared.json to verify for the validation data set. Also

please change the test data set as well inside ValDataDF read only by giving test data set path instead of validation dataset.

```
claimDataDF = pd.read_json('train-data-prepared.json')
ValDataDF = pd.read_json('val-data-prepared.json')
```

5. Navigate to end of the code and give the path to save the output json inside the field shown in screenshot below, if not it takes the default path where the notebook is saved.

```
with open("predictions.json", "w",) as outfile:
```

6. Run all the fields till end to see the output.

```
Hyperparameter tuned: {'C': 163789.3706954068}
Best score is
              precision    recall  f1-score   support

   Non claim       0.74      0.82      0.78       223
       Claim       0.60      0.49      0.54       126

    accuracy                           0.70       349
   macro avg       0.67      0.65      0.66       349
weighted avg       0.69      0.70      0.69       349
```

7. The predictions.json will be stores inside the python folder where the notbook is saved or where the custom path given.

8. Run the eval.py script using: python val.py –p “path of test set or val set” –t “path of the json output file” and the below output is seen:

```
$ python eval.py -t "C:\Python\val-data-prepared.json" -p "C:\Python\predictions
.json"
Precision: 0.6019417475728155
Recall: 0.49206349206349204
F1: 0.5414847161572052
Done.
```