NLP Assignment 1 Report

Rohit Patnaik 21225

1 Text Sentiment Classification

For this project we have a training dataset with a list of sentences which are tagged as 0 or 1 where 1 means Hate/Humour/Sarcasm and 0 means not Hate/Humour/Sarcasm. I started by reading the csv file as a data frame. Before I could use a classification model such as Naïve Bayes or Logistic Regression I had to process the textual data in usable form. I used a vectorizer which converts text data into numerical feature representations for machine learning models. I tried TfidfVectorizer (which assigns importance to words based on frequency) and Count Vectorizer (which counts word occurrences) and checked which gave higher f1 score for classifying the 1 class.

I tried a few models such as Naïve Bayes and Logistic Regression which gave good results for class 0 but not class 1. I checked for imbalanced data which was indeed the case. So, I used under sampling and over sampling to make the dataset balanced. I then used multiple models such as Logistic Regression, Naïve Bayes, K-Nearest Neighbours and Random Forest. I tried to use Grid Search to tune my hyper parameters. Finally, I got the following results:

- For Hate DataSet we are gettiing highest f1 score for class 1 by undersampling and using Random Foresst Classifier with 50% f1 score.
- For Humour DataSet are getting highest f1 score for class 1 by not doing any sampling and using Random Foresst Classifier with 78% f1 score.
- For Sarcasm DataSet We are gettiing highest f1 score for class 1 by oversampling and using Logistic Regression or Decision Tree Classifier with 81% f1 score.

2 Viterbi Algorithm

For the Viterbi Algorithm, the prerequisite is Starting Probabilities of the Parts of Speech (PoS), the transition probability of transitioning from one PoS to another and the emission probability which gives the probability of the occurrence of a word given a specific PoS. So, I wrote code to calculate this starting, transition and emission probabilities which was straight forward i.e. for:

- Start Probability = Count of tag appeareance at the start of sentence/Total no. of tags which appear at the start.
- Transition Probability = No. of times a PoS tag occurs after another PoS tag/Total no. of possible tags which can occur after the first PoS tag.
- Emission Probability = No. of times the specific word was emitted/Total no. of words that were emitted by the specific pos.

For all of these probabilities I used add-1 smoothing to avoid 0 probabilities. After the prerequisites out of the way, I tried to implement the Viterbi algorithm by implementing one step of the algorithm at a time. I used Counter Dictionaries to store my probabilities and to also store the Delta values and back pointers at each step. After storing the maximum delta values for each step and the back pointers to the PoS which gave this max delta, I took the PoS of the last word which had the max. delta value. From there I back traced to get the final PoS chain. I got an accuracy of 75.11%.

For noisy data, I could not understand what exactly noisy data meant, so I tried to remove extra punctuation marks but that gave a very low accuracy. I tried stripping unnecessary spaces and finally got an accuracy of 68.18% on the noisy data set.