Fake News Detection

Introduction

Given fake news data has six different labels provided for the statements which are considered as either fake or real based on below mapping: (‘true’, ’mostly-true’, ’half-true’): REAL (‘false’, ‘barely-true’, ‘pants-fire’): FAKE. The aim is to train classifier using these labels and get the best accuracy on test data.

- Firstly, the load_data function takes the path of the fake_news.tsv file and reads it by skipping the header as it is not required, neither for training nor for testing purpose. After that the parse_data_line function receives a row-by-row data from the file and returns only the tuple of statement i.e., data_line[2] along with the label as Fake or Real with the help of convert_label which checks for the key provided by data_line[1] and returns the corresponding value. This tuple is then appended to raw_data.
- split_and_preprocess_data: splits data into 80,20 training test splits using raw data & functions:
    - pre_process: tokenizes the string by->remove punctuations, split after space, lowercase.
    - To_feature_vector: receives tokens and does 2 task, first updates global dictionary: if token not present add token as key and assign new length as value, thus creating key with index, secondly creates a local dictionary: if token present add 1 to its value, if not then ass token and value as 1, thus creating Bag of words per sentence, returns this local dictionary.

- cross_validate: parameters- training dataset & number of folds. Training dataset is divided into train and test fold for each fold. Then the below functions are used:
- train_classifier: parameters-training fold which trains the classifier using LinearSVC() model via pipeline from sklearn, returns a classifier.
- Predict_label: parameters- Bag of words of test fold & the classifier, predicts labels using classify_many.
- Predicted label along with labels of test fold labels are used to calculate the accuracy, precision, recall and f1 score for the fold using sklearn metrics. The mean*100 of all the 10 folds is then calculated and stored in an array and returned by cross_validate. Average accuracy for this model is calculated as: 56.86%

- Predicted output, Label and Bag of words of last test fold stored using last fold in cross_validate 1st two are used to generate the heatmap for Error Analysis using confusion_matrix from sklearn.metrics via function confusion_matrix_heatmap which takes 3 parameters: test label, predicted label and list of string labels as “Real” and “Fake”. Values of confusion matrix for the last fold: [309 170] [142 191] where TP=309, TN=191, FP=142, FN=170(same notation will follow in the report). The same is then plotted using a heatmap for better visualization.
- Reading Bag of words from test fold for all the FP and FN to a file using predicted label and actual label of the test fold with help of sys library’s seek, read and write functions.
- Reading the files indicates: certain words appear a greater number of times in both documents respectively. Words like: ‘percent’, ’state’ etc. appear more in FN document which means they add more towards the Real, similarly words like: ‘Obama’, ‘president’ etc. appear more in FP documents which means that they add more towards the Fake. Hence adding weights to certain words can improve the accuracy. Secondly word like “says” and many other stop words which appear too frequently might not add anything for the accuracy, hence can be removed.

- First target to improve accuracy was done in pre_process where now it takes two parameters, the text and the Boolean stopword with default False, it does tokenization followed by lemmatization for each token using WordNetLemmatizer by converting all tokens to their basic dictionary forms. The words are then turned into Bigram by gluing two consecutive tokens with a * in between (* used only so that we can unglue using this if required). Stop word removal is not done hence set as false as it reduced the accuracy in our case.

- Certain words appeared more for Real and Fake sentences as seen in the file from question 4 hence, using real ones only, firstly there were joined in the same way, i.e., using a *, then compared with the incoming tokens and then more weights were given to those tokens which match. The other method tried was to unglue the tokens and then check in the list of real words. The latter is not chosen as it reduces the accuracy.
- Then the parameters of Logistic Regression are then altered cost parameter and the class weight (as we have given more weight to Real word hence giving more weight to Real class).
- The accuracy percentage change is shown for each of the method used which are done in sequence of the table itself with starting percentage as 56.86% from question 3 and ending ad 60.27%, all the negative changes have been discarded and either not implemented or present as false in the code.
- Just for the sake of reference other MultinomailNB, Logistic Regression and Decision Tree have also been tested on our implemented logic so far, their percentage change is shown with respect to final accuracy percentage of LinearSVC 60.27%.

Method	Accuracy % change
Bigram	1.1
Lemmatization	0.4
Stop word removal	-1.8
More weights to positive words	0.02
Unglue tokens	-1.1
Linear SVC(Class Weight)	
Real>Fake	0.3
Linear SVC(Cost Parameter)	
0.1	0.6
0.2	-0.3
0.3	-0.01
0.4	-0.2
Increasing cost parameter, i.e., penalizing samples inside the margin more is decreasing the accuracy	
0.06	0.8
0.006	1.2
0.0006	-1.8
Penalizing samples inside the margin less i.e., reducing c up to certain level increases the accuracy, hence taking 0.006	

Model	% Change
MultinomailNB	0.7
Logistic Regression	
Decision Tree	

- Here all the columns including statement have been included to predict the label, changes done in the functions:
- Parse_data_line: It now returns a tuple of value other columns as a dictionary, text and label, thus the same is appended to raw data.
- To_feature_vector: takes two parameters: tokens from which it creates the dictionary as earlier and the new dictionary of the column values. Returns the addition of both dictionaries, this is then appended to test and train data.

Column	Matrix	Accuracy	Combination
total_barely_true_counts	[[400 79] [202 131]]	62.8	y
total_false_counts	[[405 74] [186 147]]	64.5	y
total_half_true_counts	[[411 68] [200 133]]	63.5	x
total_mostly_true_counts	[[416 63] [212 121]]	63.8	x
total_pants_on_fire_counts	[[386 93] [195 138]]	63.2	y
Combination Accuracy x	65.9		
Combination Accuracy y	64.7		
Combination Matrix x	[[408 71] [183 150]]		
Combination Matrix y	[[399 80] [178 155]]		

- It is clear from the table that when we use mostly true and half true count columns, they result in better accuracy as we have given more weight to the Real class. Also, we can see that individually barely true, false and pants on fire results in good numbers of True negative values, also their combination does the same. Similarly, individually half and mostly true and their combination results in good numbers of True Positive values.
- The combination of all the columns does not give a good result because of our model being a little bias towards Real class.

Conclusions

The model shows reasonable performance with final accuracy of the classifier on the training data folds as 65.93, when the same deployed on the actual 80:20 training test split, the accuracy comes out to be 65.42. For Real class the predictions are more accurate. The model generalizes with around same score for accuracy