Dataset used for both the models: LIAR- PLUS

Steps I took while building the classifiers:

1. The dataset:

LIAR-Plus dataset: for fake news detection is used for my analysis throughout to choose an appropriate classifier to give highest possible accuracy. After preprocessing the dataset, I used scikit-learn to build a predictive model.
There are two files: one for binary classification and the other for 6 class classification. Both use the LIAR plus dataset for training and testing.

2. Importing and exploring the dataset:

First step is to import the pandas module and use the appropriate path of the downloaded dataset. Next, the read_csv() method provided by the pandas module is used to read/load the tsv file which contains tab separated values and convert that into a pandas DataFrame. This method can also be used for csv file datasets. I observed from the dataset that the columns were hard to comprehend, hence i added a column array and used it while reading the dataset. For further exploring the dataset, using the pd.head() and pd.tail() method, I tried to print the first/last n-values for all the three files: train, test and val.

3. Data Pre-processing:

I observed that the dataset is uneven with different data-types, string values, categorical textual data, NaN (np.na) values and uppercase. To clean the data:

a. For building the binary classifier model, all the 'true and 'false' data had to be extracted from the 'Label' column for further processing. This can be done by *pd.loc()* method
b. Dropped the nan values by using panda's function *(.dropna())*

c. Cleaned the first column to remove the " .JSON " successor from every value

d. I also further tried to code the TFIDF for future information retrieval, if required. This value assigns a weight to every word - where the weight **is** a statistical measure **used** to evaluate how important a word **is** to a document.

e. Initial exploration that most of the columns in our data set are strings, but the algorithms in scikit-learn understand only numeric data

f. After performing the basic cleaning of the data, the main task is to convert textual string to numeric data to train and test further. For this, we can use either **label encoding** or **one hot encoding** of the data. The LabelEncoder() function of sklearn takes user defined labels as input and then returns encoded labels [1]. We can use fit_transform() function to change the data values. Similarly, the OneHotEncoder() function : it provides a representation of categorical variables as binary vectors. The input to this transformer should be an array-like of integers or strings, denoting the values taken on by categorical (discrete) features [1].

4. Building the Model- Multi-label (6 class classification)

At first, I tried used label encoding *(sklear.preprocessing.labelencoder())* [1] on the entire dataset then split the **train2.tsv** file data into train-test samples in a 80-20 ratio and subsequently ran the classifiers: *random forest ensemble*, *LinearSVM, Logistic Regression* and *multinomial Naive Bayes*. The following classifiers came into my mind after observing the LIAR dataset for the classification of both the tasks: 6 label classification and binary-classification. Since the text data involves the classification of news to categorize in multiple labels based on the level of truthfulness, this is a **multi-label classification.** One of its main advantages of MNB is that we can get good results even with limited data (thousands of tagged samples) and computational resources are scarce. Similarly, like naive bayes, SVM doesn't need much training data to start

providing accurate results. Random Forests are an ensemble learning method that fit multiple Decision Trees on subsets of the data and average the results

All scikit-learn classifiers are capable of multiclass classification, I chose to begin with the aforementioned four algorithms to look into the accuracy of the modeli train using the ***train2.tsv*** file comprising: *80% x_train and 20% y_train.* At first, they came out to be:

Classifiers	Accuracies
RandomForest	~35%
Logistic regression	~27%
Multinomial	~19.8%
Linear SVM	~235

Improving the accuracy:

To improve the accuracy, I tried One hot encoding the columns and then training the same model on the same classifiers to look into the accuracy again. I found Random Forest, LinearSVM and Logistic Regression outperformed Multinomial Naive bayes.

One hot encoding performs better in training because in label encoding, where we give sequential numbering to categorical data, there might not be any relationship or order that they might occur in. But f i may need to assign the values in a different order, then I might not be able to and the model will end up biasing that say category B is twice as large as A and C is thrice as

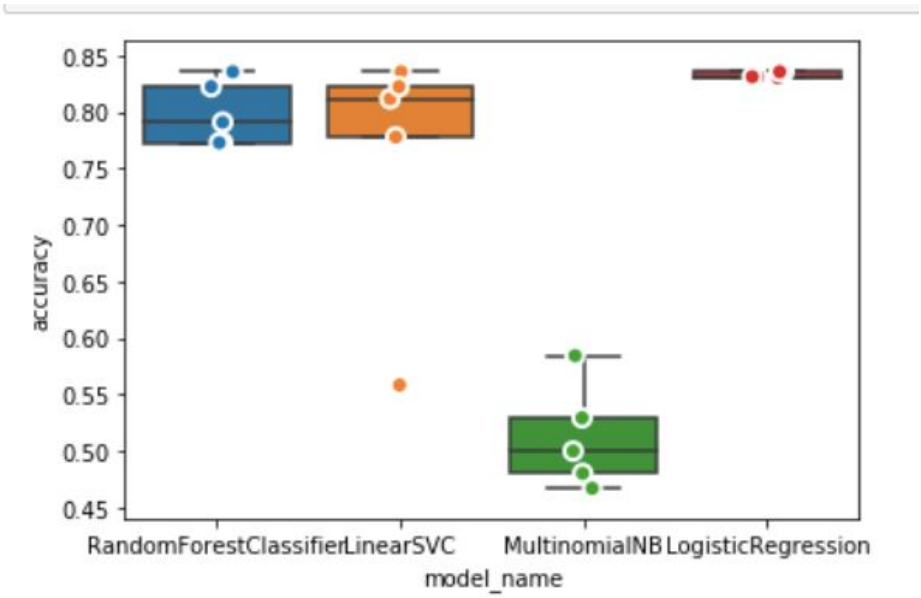
A.. so on. The problem here is since there are different numbers in the same column, the model will misunderstand the data to be in some kind of order, 0 < 1 <2. [2]

Hence, to avoid biasing of the model, I tried using one hot encoding the categorical values which has an advantage over label encoding because the result is binary rather than ordinal. Everything sits on an orthogonal vector space.For huge datasets, to avoid blowing of the feature space, one can also use algorithms like PCA for dimensionality reduction. The following accuracies achieved after one hot encoding.

After fitting the trained model into the testing data (**comprising *90% of test2.tsv file***):

```
Testing accuracy of LR 0.807843137254902
r square error for LR -0.3792067307692306
mean square error for LR 0.19215686274509805
Testing accuracy of MNB 0.5176470588235295
r square error for MNB -2.4620903649921506
mean square error for MNB 0.4823529411764706
Testing accuracy of RF 0.8248366013071895
r square error for RF -0.2572360675039247
mean square error for RF 0.17516339869281045
Testing accuracy of SVC 0.8196078431372549
r square error for SVC -0.29476550235478793
mean square error for SVC 0.1803921568627451
```

The cross validation score plot (cross_val_score) to give a visual representation of the estimated accuracies of all the classifiers used [5]



Using a Neural Network: MLP classifier [4]

Traditionally, machine learning algorithms like SVM and NB reach a certain threshold where adding more training data doesn't improve the accuracy. In contrast, deep learning classifiers continue to get better the more data you feed in. Hence for future perspectives, I decided to train the model on a basic neural network: Multi-layer perceptron and then noting the accuracies and errors on the testing dataset.

Input layer: Specifically, *the number of neurons comprising that layer is equal to the number of features (columns) in our data (14, in our case).*
How to I chose the number of hidden layers:

- The number of hidden neurons should be less than twice the size of the input layer. [3]
- The **number of hidden** neurons should be between the size of the input **layer** and the size of the output **layer**. The **number of hidden** neurons should be 2/3 the size of the input **layer**, plus the size of the output **layer**. [3]

- *The optimal size of the hidden layer is usually between the size of the input and size of the output layers*'. Jeff Heaton, author of **Introduction to Neural Networks in Java**. [3]

i. Following the above rules:
In our case, the number of hidden layers to be used is 27 (*less than 14*2- 28, input layer size***).**
The maximum possible accuracy achieved with 27 hidden layers and 10 units each is :

```
#using neural networks
from sklearn.neural_network import MLPClassifier

NN = MLPClassifier(solver='adam', alpha=1e-5, hidden_layer_sizes=(10, 27), random_state=1).fit(X_train, y_train)
NN.predict(X_test)
round(NN.score(X_test, y_test), 5) #Returns the mean accuracy on the given test data and labels, with weight: 5
```

```
0.80735
```

I also tried using the other optimizing algorithms: **sgd (Stochastic gradient descent)**, ***lbfgs****(optimizer from quasi-newton methods') but chose* ***'adam'*** *as works well on large datasets and was efficient in terms of training time and validation score*

ii. Mending our model based on getting the highest possible accuracy, I decreases the number of layers and increased the individual elements in it.

To get the highest accuracy for multi-class data: 0.8326

```
In [554]: #using neural networks- MLP
          from sklearn.neural_network import MLPClassifier

          NN = MLPClassifier(solver='adam', alpha=1e-5, hidden_layer_sizes=(170, 15), random_state=1).fit(X_train, y_train)
          NN.predict(X_test1)
          round(NN.score(X_test1, y_test1), 5) #Returns the mean accuracy on the given test data and labels, with weight: 5

Out[554]: 0.83268
```

Building the model- Binary (2-class) Classification of LIAR PLUS

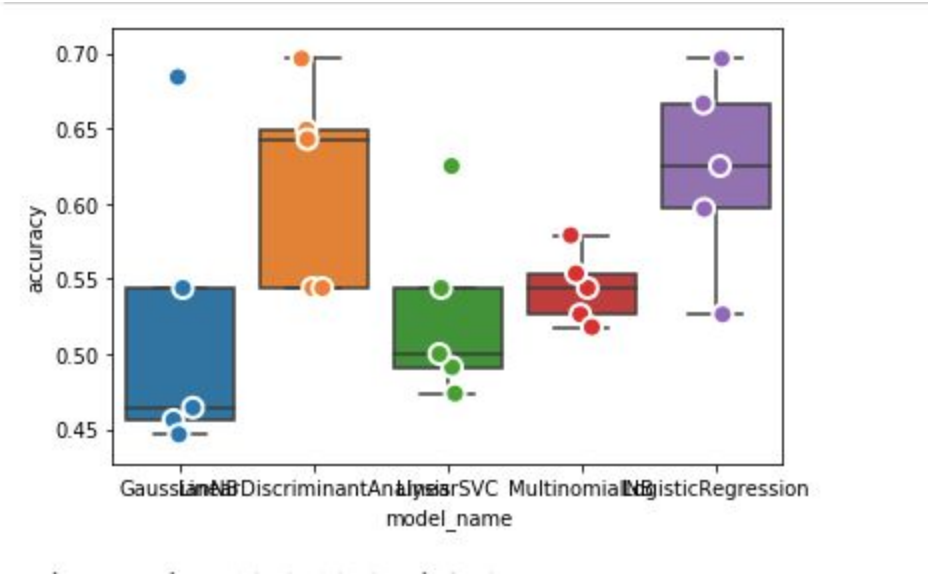
After pre-processing the data according to our need for only true and false values, the following is models are carried on the training data first from train2.tsv file and then testing on the test2.tsv file.

1. Assuming that the distribution of the data is Gaussian, that each variable is is shaped like a bell curve when plotted. I thought of using a Bayesian approach of Gaussian Naive Bayes. But the accuracy was not impressive and was about **0.53**
2. The LDA model estimates the mean and variance from our data for each class. Using Bayes' rule, fitting class conditional densities. The accuracy came out to be almost close to the maximum **- 0.63**
3. For Decision trees, a model is created that predicts the value of a target variable by learning simple decision rules inferred from the data features. The accuracy came out to be the maximum in this case **- 0.657**

The potential reason for the accuracy not improving in the current dataset is because we have selected only those rows which have 'true' or 'false labels and the dataset has shrunk down to some 2400 rows. Possible ways to improve the accuracy are:

1. Increasing the data points, maybe by data augmentation and replicating the rows or adding the average number of true or false statements soas to improve the accuracy.
2. For the future perspectives, Google's BERT for binary text classification using transfer learning with the help of tensorflow

Cross- validation score to improve the individual models



Accuracies and error predictions for binary text classification:

```
Testing accuracy of LR 0.657243816254417
mean square error for LR 0.34275618374558303
Testing accuracy of MNB 0.5618374558303887
mean square error for MNB 0.4381625441696113
Testing accuracy of RF 0.6360424028268551
mean square error for RF 0.36395759717314485
Testing accuracy of SVC 0.5441696113074205
mean square error for SVC 0.4558303886925795
Testing accuracy of GNB 0.49823321554770317
mean square error for GNB 0.5017667844522968
Testing accuracy of LDA 0.6289752650176679
mean square error for LDA 0.3710247349823322
Testing accuracy of DT 0.657243816254417
```

References:

1. https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics

2. https://towardsdatascience.com/choosing-the-right-encoding-method-label-vs-onehot-encoder-a4434493149b
3. https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw.
4. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
5. https://scikit-learn.org/stable/modules/cross_validation.html
6. https://scikit-learn.org/stable/modules/model_evaluation.html#mean-squared-error
7. https://stackoverflow.com/questions/17071871/select-rows-from-a-dataframe-based-on-values-in-a-column-in-pandas
8. https://scikit-learn.org/stable/modules/tree.html
9. https://scikit-learn.org/stable/modules/model_evaluation.html#mean-squared-error
10. Pandas:https://pandas.pydata.org/pandas-docs/stable/
11. Numpy:https://docs.scipy.org/doc/