Name: Zuhaib Akhtar

REPORT

DataSet has 166 Features having integral values. Each column (feature) has different mean and standard deviation, therefore it is necessary to convert each feature into Standard Normal Distribution. This means that each of the feature will have mean = 0 and standard deviation = 1 (also known as bell curve). This makes the data symmetrical about the mean and hence makes it mathematically convenient to manipulate. If we can work out the probabilities for a distribution with a mean of 0 and standard deviation of 1, then we can work them out for any similar distribution of scores with a very simple equation.
The function for converting data into Standard Normal Distribution is StandardScaler()

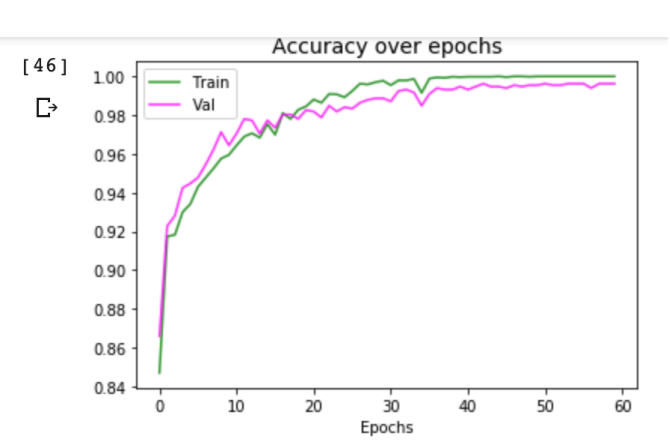
Now, we can divide our dataset into training set and validation set (80/20, given).

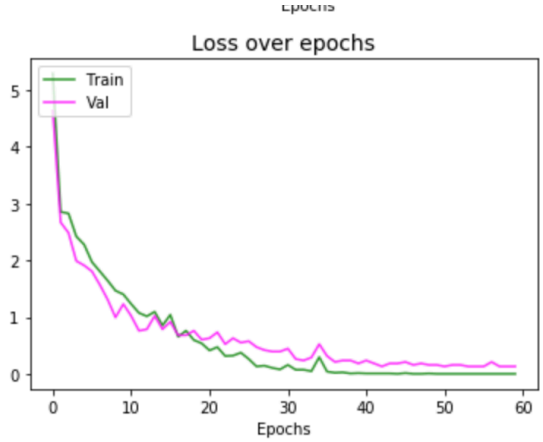
ARCHITECTURE:
For this problem, I chose Multi-Layered Perceptron (MLP) with 2 hidden layers. MLP with hidden layers can easily learn non-linear decision boundary in the data.
MLP has 166 input neurons (total features). First hidden layer has 270 neurons and second has 150 neurons. Output layer has 1 neuron.
Logistic sigmoid activation function is used in this architecture with Adam optimization algorithm. Adam generally works well for large datasets.

MLPClassifier(hidden_layer_sizes=(270,150,),activation='logistic',max_iter=500,verbose=False
)

Finally Training is performed in chunks with batch size of 128 using partial_fit() function. This function would let us update the model with incremental data. Learning in batch is also useful in cases where dataset is large.

After training on whole dataset by going over them batch-by-batch, one epoch is complete. At this time we can calculate our validation accuracy and loss.
I set the number of epoch to 60. After completing the training, we can finally predict the accuracy on validation set (or on test set if given) with other metrics such as precision, recall, f-measure and confusion matrix.
Below are the plots for training and validation accuracy and loss over 60 epoch.





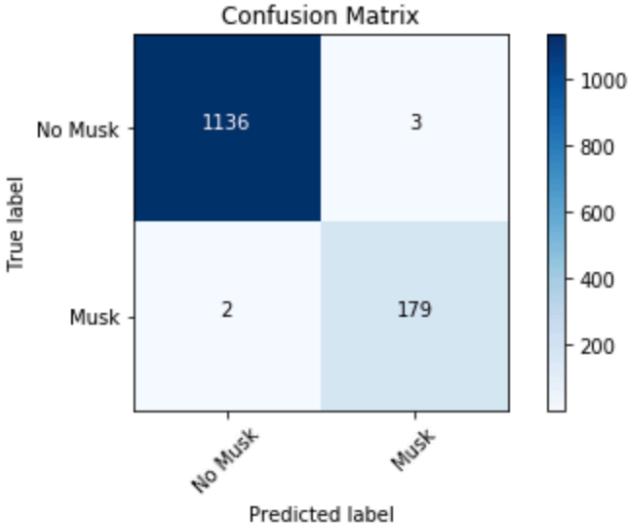
Name: Zuhaib Akhtar

The plots above suggests that the model converged well. Validation accuracy is high (almost same as Training accuracy) and Validation loss is very less (almost as less as training loss).

The diagram below shows true positive, true negative, false positive and false negative. Values of true positive and true negative are high and false positive and false negative is less which is a good thing for our model.

Finally, confusion matrix gives us the visualization of the performance of the model. Only 2 Musk Molecule has been classified incorrectly as Non-Musk Molecule, while 179 Musk Molecule has been classified correctly as Musk Molecule. Similarly, Only 3 Non-Musk Molecule has been classified incorrectly as Musk Molecule, while 1136 Non-Musk Molecule has been classified correctly as non-Musk Molecule.
Model has made very less error and almost all data-points of the validation set has been classified correctly.
Also shown in the plot below, other performance metrics such as Precision, Recall and F-Measure is very high, which is desirable.

```
true pos  1136
false neg 3
false pos 2
true neg  179
Precision:  0.9982425307557118
Recall:  0.9973661106233538
F-Measure:  0.9978041282389107
```

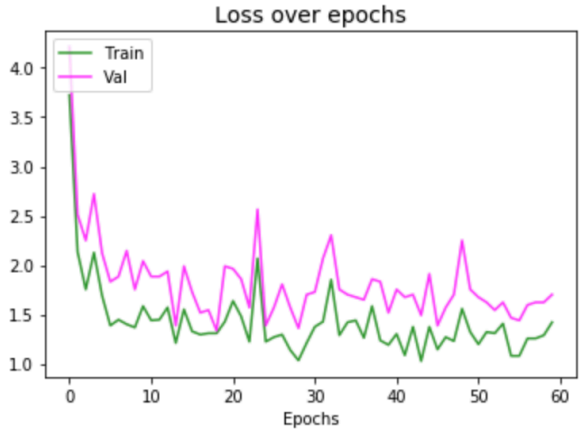


Finally, the overall accuracy of the model on validation dataset is **99.621%** shown below. This model is attached in the folder as 'Finalized_Model.sav'.

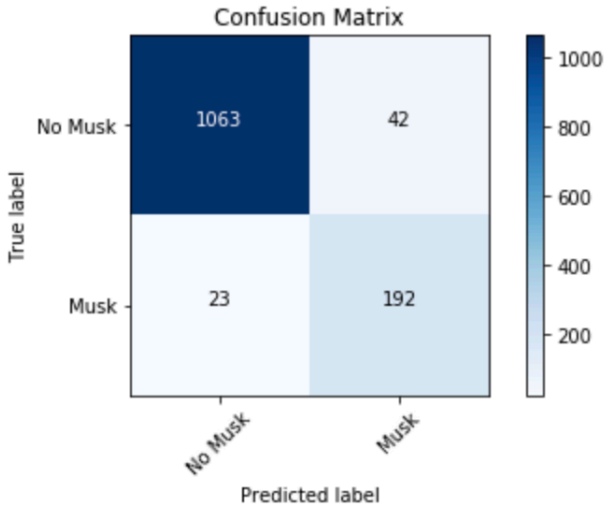
```
0.9962121212121212
```

Name: Zuhaib Akhtar

I also performed training by not converting the data into Standard Normal Distribution and plotted the training and validation loss and accuracy.
As we can see training is not smooth and there is lot of jittering during training. Loss is also high.
Plot of confusion matrix shows that performance worsens when compared with previous plot. Accuracy comes out to be **95.07%.** And in confusion matrix, several data points have been classified incorrectly.



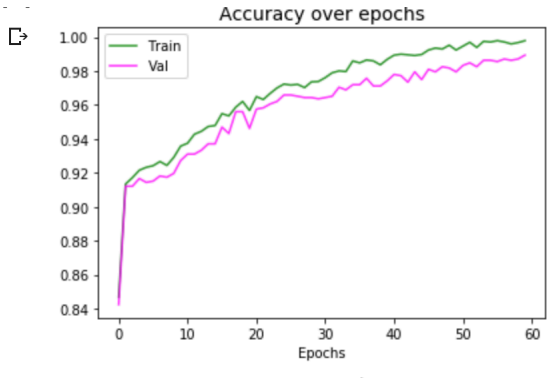

```
true pos  1063
false neg 42
false pos 23
true neg  192
Precision:  0.9788213627992634
Recall:  0.9619909502262444
F-Measure:  0.970333181195801
```

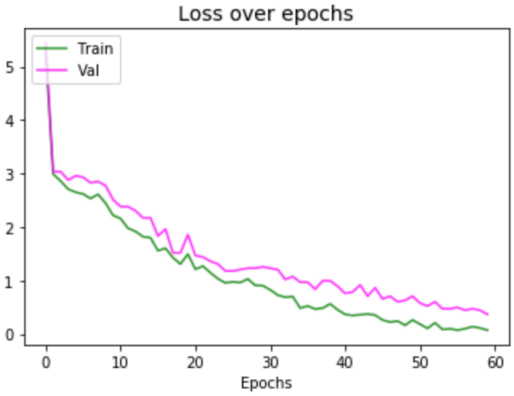



```
0.9507575757575758
```

Name: Zuhaib Akhtar

As there were lot of features, I also tried principal component analysis (PCA) which reduces the number of features by converting correlated features into linearly uncorrelated values. I did convert the data into Standard Normal Distribution in this case.
Below is the plot of training and validation accuracy and loss in this case. Although, it is not as good as the first case where all the features were taken, but it is acceptable when we have reduced the number of input features to a huge extent (PCA converted our data into 39 input features as opposed to 166 features in the first case).



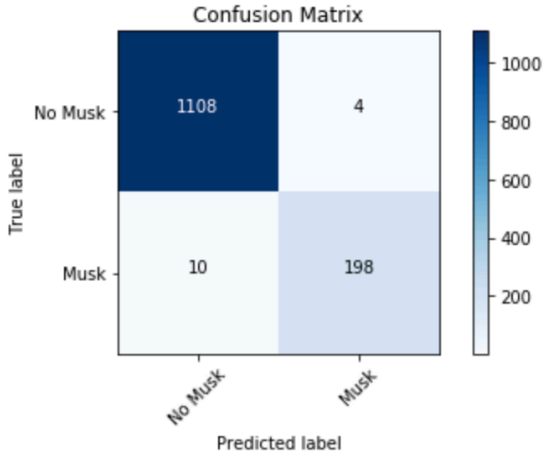


Code fo PCA (place this code after preprocessing and before splitting the dataset):

```
pca = PCA(.95)
pca.fit(X_scale1)
X_scale = pca.transform(X_scale1)
```

The confusion matrix and accuracy of validation shows the results are good but not as good as the first case. Hence, we can conclude that there is a compromisation between model size (total input features) and accuracy of the model. The accuracy in this case is **98.939%** (39 feature input) which is less than the first case **99.621%** (166 feature input)

```
true pos  1108
false neg 4
false pos 10
true neg  198
Precision:  0.9910554561717353
Recall:  0.9964028776978417
F-Measure:  0.9937219730941704
```




```
0.9893939393939394
```

Name: Zuhaib Akhtar

I also tried analyzing the data by reducing the number of features to just 2 components, but it didn't give a good insight (or visualization) i.e., data does not seem separable when reducing it into 2 dimensions. That is why PCA itself reduced the features into 39 input features, which it find suitable number of features to classify the dataset.
Plot shows the data plotted as two input features.