Introduction

In this exercise, A baseline model for classification of given dataset has been proposed. Following document outlines walkthrough of process that was followed for the completion of given task.

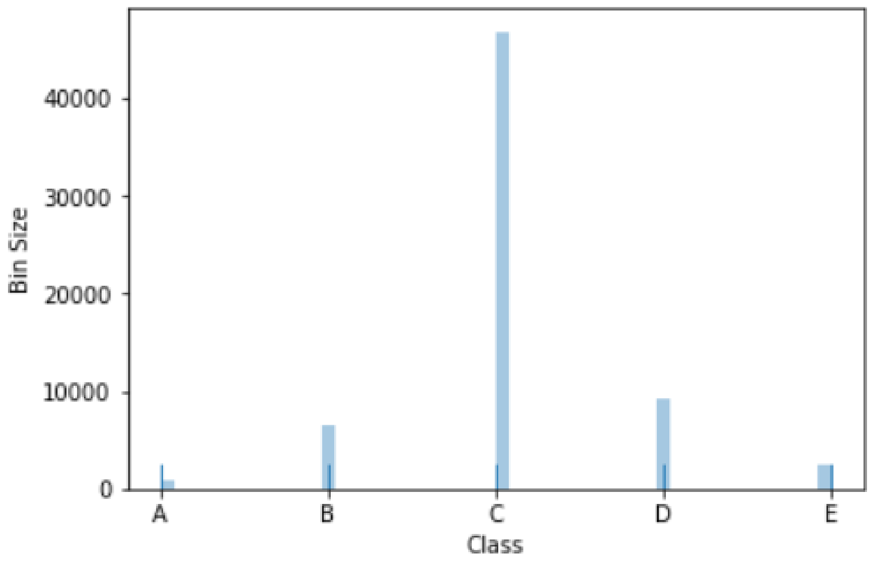
Method

Pre-Analysis

- Target variable had class labels instead of continuous data. There is no missing value for label in any observation.
- Class distribution was unbalanced, with Class C being over-represented and Classes A and E being under-represented. Labels indicate a Normal distribution with near perfect symmetry. This information will be used for output layer of neural network.

  Following output shows bin sizes of each class and histogram of said distribution.

```
array([  867,  6602, 46882,  9279,  2507], dtype=int64)
```



Class Distribution

Function: MLChallenge.get_class_distribution()

- Features had noticeable sparsity.
- 11 features had all-zero column vectors. These columns needed to be dropped for reducing unnecessary noise. Following output shows indices of such column vectors.

```
array([ 23,  59, 179, 268, 269, 270, 272, 273, 274, 275, 276], dtype=int64)
```

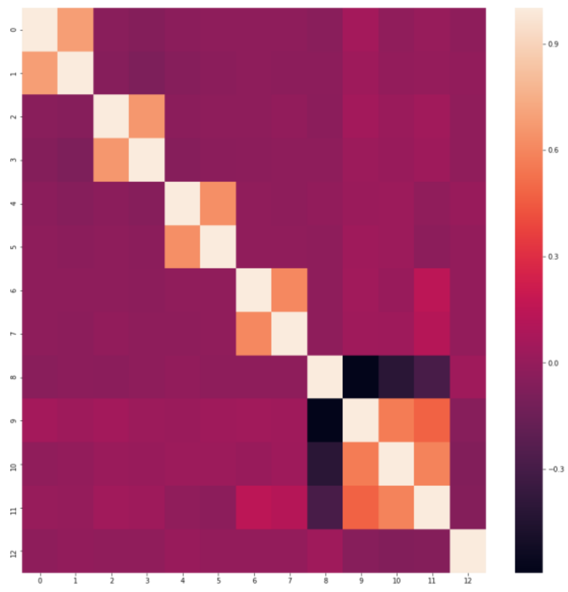
Function: MLChallenge.drop_all_zero_columns()

- There were some features which showed considerable correlation. This information suggested that the data quality is good enough for using it for further processing.

Following are the top 5 pairs of features with strongest correlation as indicated by Pearson Correlation Coefficient. First two columns are indices of column vectors and third column is the value of their Correlation Coefficient. (refer to table on left)

```
127  71     0.688079
140  72     0.661066
72   140    0.661066
80   278    0.628463
278  80     0.628463
1    195    0.602828
195  1      0.602828
11   6      0.592659
6    11     0.592659
43   3      0.584448
3    43     0.584448
```

Top 5 Strongly Correlated Feature Pairs



Heat Map of Top 5 Correlated Feature Vectors

Function: MLChallenge.get_top_n_correlated()

Feature Extraction and Data Cleaning

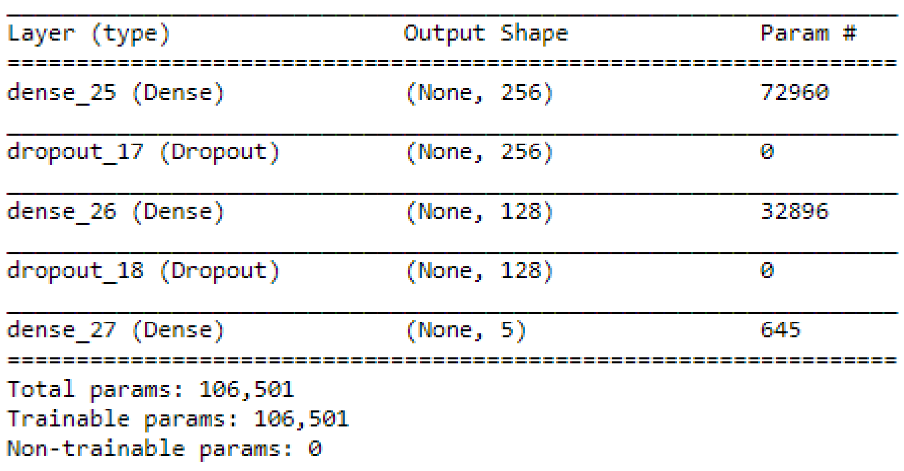
- Raw Features and Labels were obtained using slicing.
- All-Zero feature vectors were removed from input variable for reducing unnecessary noise during training phase. (Function: MLChallenge.drop_all_zero_columns())
- Features were further normalized using Min-Max Scaling within range of 0 and 1 for ensuring fair treatment of every individual feature. (Function: MLChallenge.normalize_matrix_by_min_max())
- Labels were encoded to numeric values as a pre-requisite of using SoftMax Activation function in output layer. (Function: MLChallenge.encode_vector_class_to_int())
- Lastly, features and labels were split into training and testing part for cross-validation. Size of training data was 33% of original data. (Function: MLChallenge.split_by_test_size())

Model Creation and Hyperparameter Selection

- Pre-processing resulted a tabular dataset with labelled observations. The question was to find a function that defines mapping of features and labels in best possible manner. Given the dataset, A Multi-Layer Perceptron (MLP) seemed to be the most viable option, and a good starting point for our baseline model, i.e. a non-linear mapping problem in ours case. As for the optimization of objective function, we used a Stochastic Gradient Descent optimizer (as suggested in SciKit-Learn Algorithm Cheat-Sheet by Andreas Mueller).
- In practice, for finding optimal hyperparameter combinations, Hyperparameter Optimization is usually carried out using Grid Search Cross Validation. Due to insufficient computing resources, said activity was not fully performed. For the purposes of demonstration, code for hyperparameter optimization was developed and tested on Hyperparameter Space with

limited options (MLChallenge.optimize_hyperparamters()). However, these results were not made part of final model.

- Following is the summary of model composition;



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_25 (Dense)             (None, 256)               72960     
_________________________________________________________________
dropout_17 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_26 (Dense)             (None, 128)               32896     
_________________________________________________________________
dropout_18 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_27 (Dense)             (None, 5)                 645       
=================================================================
Total params: 106,501
Trainable params: 106,501
Non-trainable params: 0
```

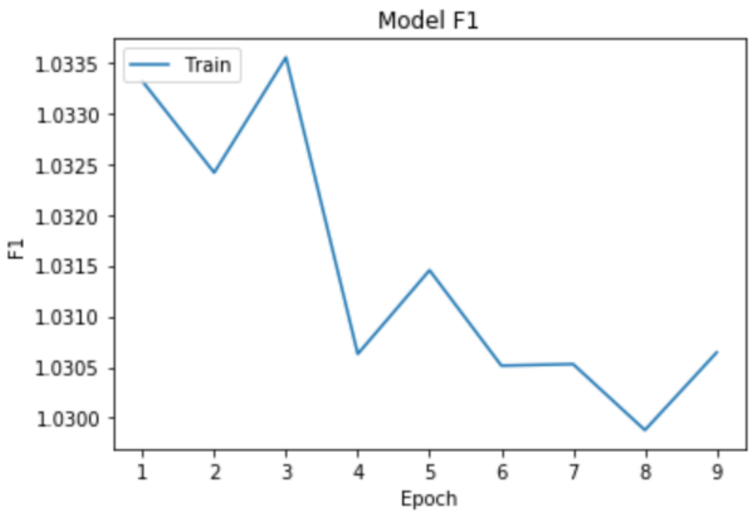
Model Summary

Function: keras.model.summary()

- As seen in above summary, following was the order of layers.
    1. Dense Layer:
        - A layer for fully connecting input layer with forthcoming layers
        - 256 unit neurons with "ReLU" activation function
        - Since, we know shape of feature beforehand, we have also defined input shape for this layer.
        - The number of parameters obtained for first layer turned out to be 72960 (284-InputFeatures + 1-Bias x 256-Neurons)
        - Class distribution of Features was found to be Normally distributed, hence for kernel initialization, we used Normal Distribution
    2. Dropout layer
        - Following layer was added for ensuring generalization and preventing model overfitting during training phase
    3. Dense Layer
    4. Dropout Layer
    5. Dense Layer
        - "SoftMax" was used in final layer for obtaining prediction probabilities for each class category

Model Evaluation

- Since we are dealing with unbalanced class distribution, F1 score was used as model evaluation metric. Said metric takes precision and recall metric into account for evaluating prediction quality.
- Following graph shows the progression of training history of F1 score over 10 epochs. F1 score progresses from 1.0335 to 1.0300, hence approaching to perfect F1 score of 1.



F1 Score History During Training

Function: keras.model.evaluate()

Conclusion

A baseline model was proposed for classifying given dataset. Proposed model was obtained as a result of performing following activities;

1. Pre-Data Analysis
2. Feature Extraction and Pre-Processing
3. Hyperparameter Optimization
4. Model Creation
5. Model Evaluation

Model in question achieved an F1 score of 1.0300.

Next Step

As discussed, proposed solution is a mere baseline model. Following measures can be taken for improving said model;

- Generalization performance is yet to be determined (and improved).
- Hyperparameter optimization (HO) can be performed on a wider Hyperparameter Space with varying options. HO is a compute-intensive process. These activities should be performed ideally on a GPU.
- Sparsity in features could have been reduced during pre-processing phase for getting better results
- Consultation with a person with dataset's domain knowledge will be a crucial step in improving this model