Smai-mini-project -2

Feature	Classifier	Accuracy	F1 Score(micro)
Raw Data	Soft Margin Linear SVM	38.03%	38.03%
Raw Data	Logistic Regression	41.71%	41.71%
Raw data	MLP	46.31%	46.31%
Raw data	Decision Tree	32.65%	32.65%
LDA	Soft Margin Linear SVM	36.344%	36.344%
LDA	Logistic Regression	36.71%	36.71%
LDA	MLP	36.56%	36.56%
LDA	Decision Tree	34.27%	34.27%
PCA	Soft Margin Linear SVM	40.5%	40.5%
PCA	Logistic Regression	39.45%	39.45%
PCA	MLP	47.45%	47.45%
PCA	Decision Tree	30.34%	30.34%

Executing the code

python3 smai-mini-project-2.py -c ‘classifier name’ -m ‘method of dimensionality reduction’ -hp ‘hyper parameters’

Classifier name - LogReg , LinearSvm ,DecisionTree

Method name - PCA, LDA, raw

Hyper parameters - 1 value for Logistic Regression, Linear SVM and Decision Tree

- - 2 for Multi Layered Perceptron

Summary

I first explored the many ways of solving the assignment such as using **pytorch, tensorflow, scikit-learn** and accordingly weighing the pros and cons and then started doing the assignment. After deciding on using **scikit-learn** I went through the official-documentation.I subtracted the mean to center the data and divided by variance to scale it and then I have trained the input-data and then tuned the hyperparameters on the outputs to obtain the accurate ones. I have added the graphs in the report.

Observation

- I have chosen the number of components for **Principal Component Analysis** such that the amount of variance becomes 95%.

  > If `0 < n_components < 1` and `svd_solver == 'full'`, select the number of components such that the amount of variance that needs to be explained is greater than the percentage specified by n_components.

  which is from the official sklearn documentation.
  I chose 95% by testing for various values of variance and found 95% to be the most accurate

- I have chosen the number of components for **Linear Discriminant Analysis** as '9' after testing for various values-https://chrisalbon.com/machine_learning/feature_engineering/select_best_number_of_components_in_lda/(taken reference from here)

  .

  Create Function Calculating Number Of Components Required To Pass Threshold

```python
# Create a function
def select_n_components(var_ratio, goal_var: float) -> int:
    # Set initial variance explained so far
    total_variance = 0.0

    # Set initial number of features
    n_components = 0

    # For the explained variance of each feature:
    for explained_variance in var_ratio:

        # Add the explained variance to the total
        total_variance += explained_variance

        # Add one to the number of components
        n_components += 1

        # If we reach our goal level of explained variance
        if total_variance >= goal_var:
            # End the loop
            break

    # Return the number of components
    return n_components
```

- For **Logistic Regression** I have varied the hyperparameter - '**c**' over a set of values and chose the most optimum value obtained from the results. The rest of the parameters are standard such as l2 regularisation.

- For **Multi-layer Perceptron** I have varied the hyperparameter - '**alpha**' and '**learning rate'** over a set of values, keeping **RELU** as the activation function, **early_stopping = true** and the rest as standard.

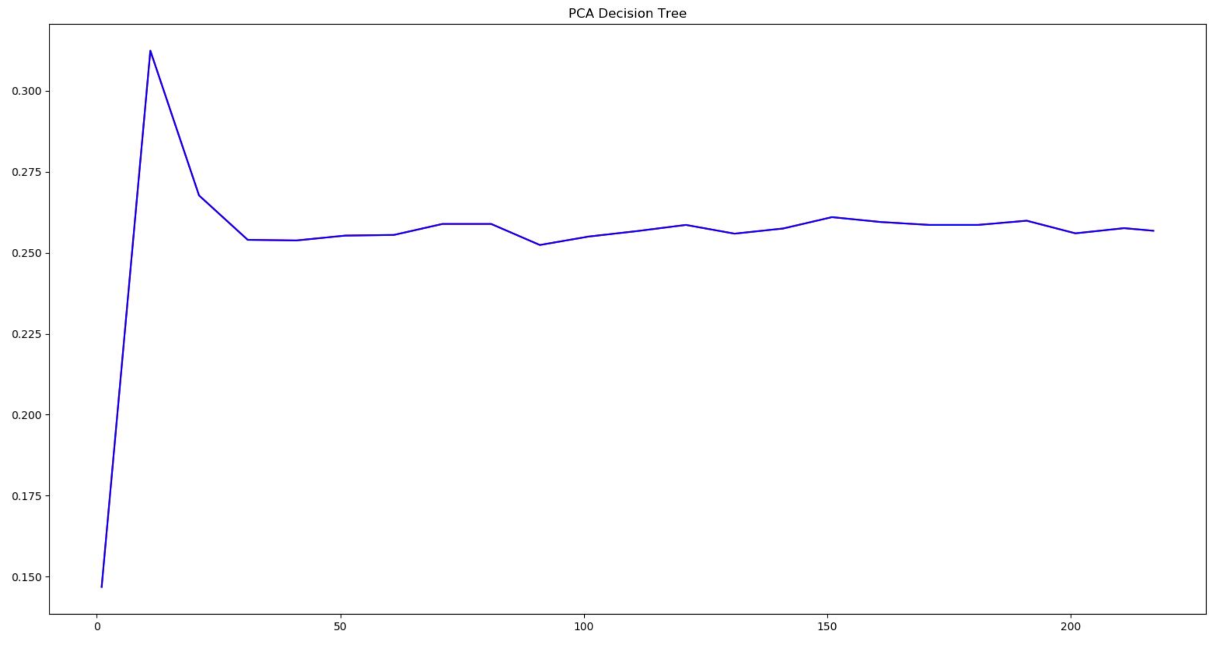
- For **Linear - SVM(soft margin)** again I have varied hyperparameter - '**c**' over a set of values and kept the rest of them as the default parameters.

- For **Decision-Tree** max-depth of the tree was varied over 1 to number of **features**.

Overfitting

- Overfitting occurs when the model or the algorithm fits the data too well.

  In the graph below the accuracy for PCA on Decision Tree decreases after reaching a peak. The x-axis is the number of features, as it increases the classifier tries to overfit hence reducing accuracy for test data.



- Overfitting can be prevented by using **cross-validation** to compare their predictive accuracies on test data.ross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the *training set*), and validating the analysis on the other subset (called the *validation set* or *testing set*) In ***k*-fold cross-validation**, the original sample is randomly partitioned into *k* equal sized subsamples. Of the *k* subsamples, a single subsample is retained as the validation data for testing the model, and the remaining *k* − 1 subsamples are used as training data. The cross-validation process is then repeated *k* times, with each of the *k* subsamples used exactly once as the validation data. The *k* results can then be averaged to produce a single estimation. The advantage of this method over repeated random sub-sampling (see below) is that all observations are used for both training and validation, and each observation is used for validation exactly once.

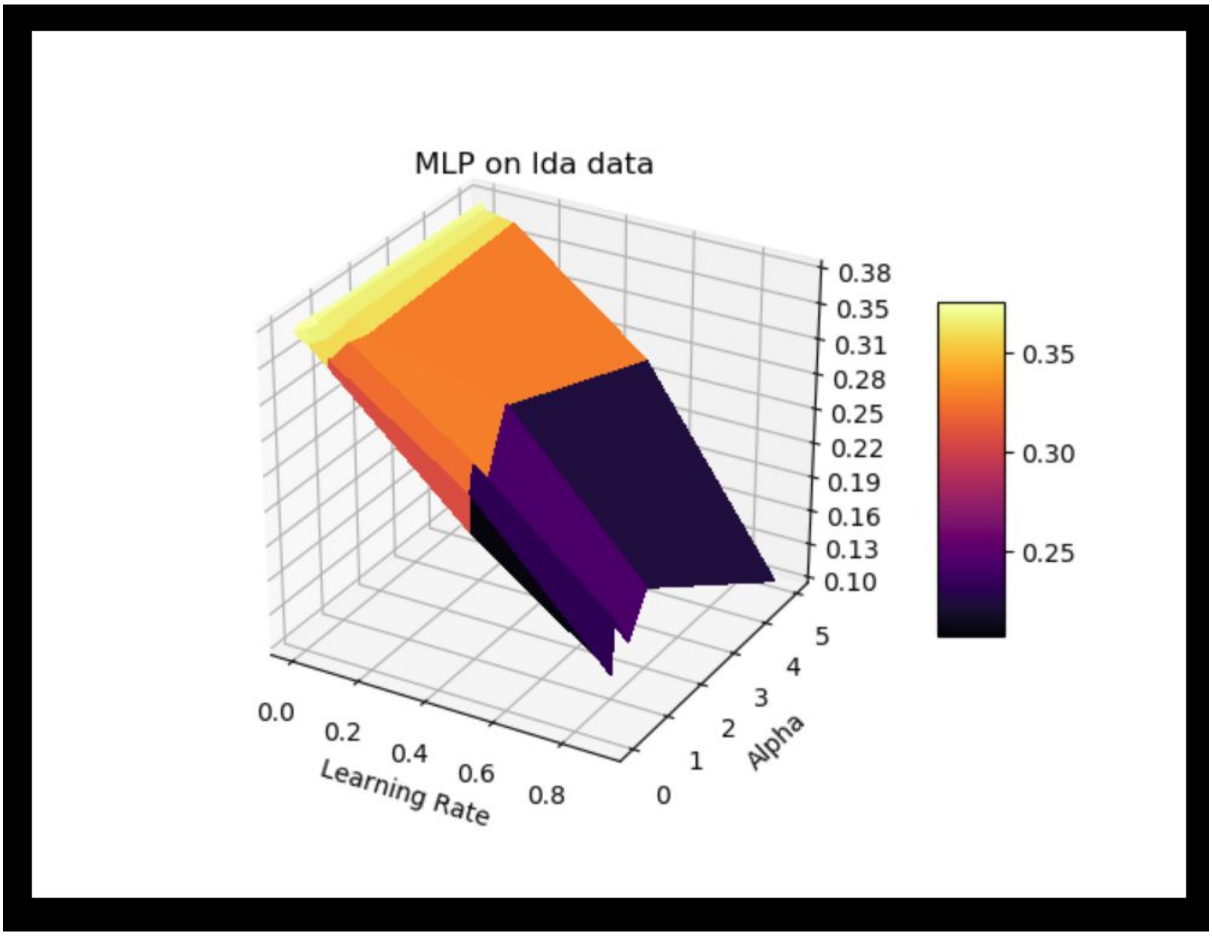
Problems Faced

- In order to tune the data to obtain the finest parameters. I can use the inbuilt sklearn function Gridsearch to iterate over all the parameters but due to the lack of computation power I could not do this.

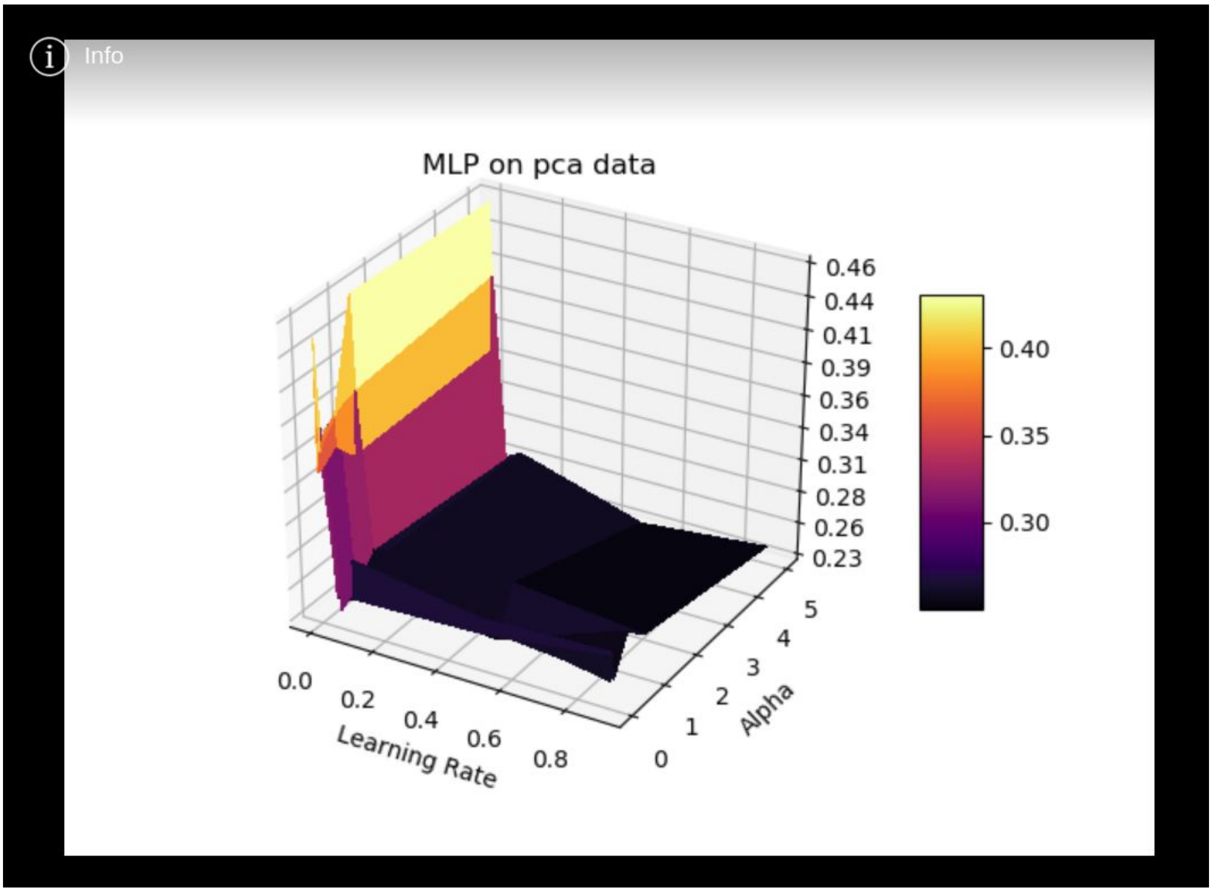
- Time taken to compute output for each input was very slow.

Graphs

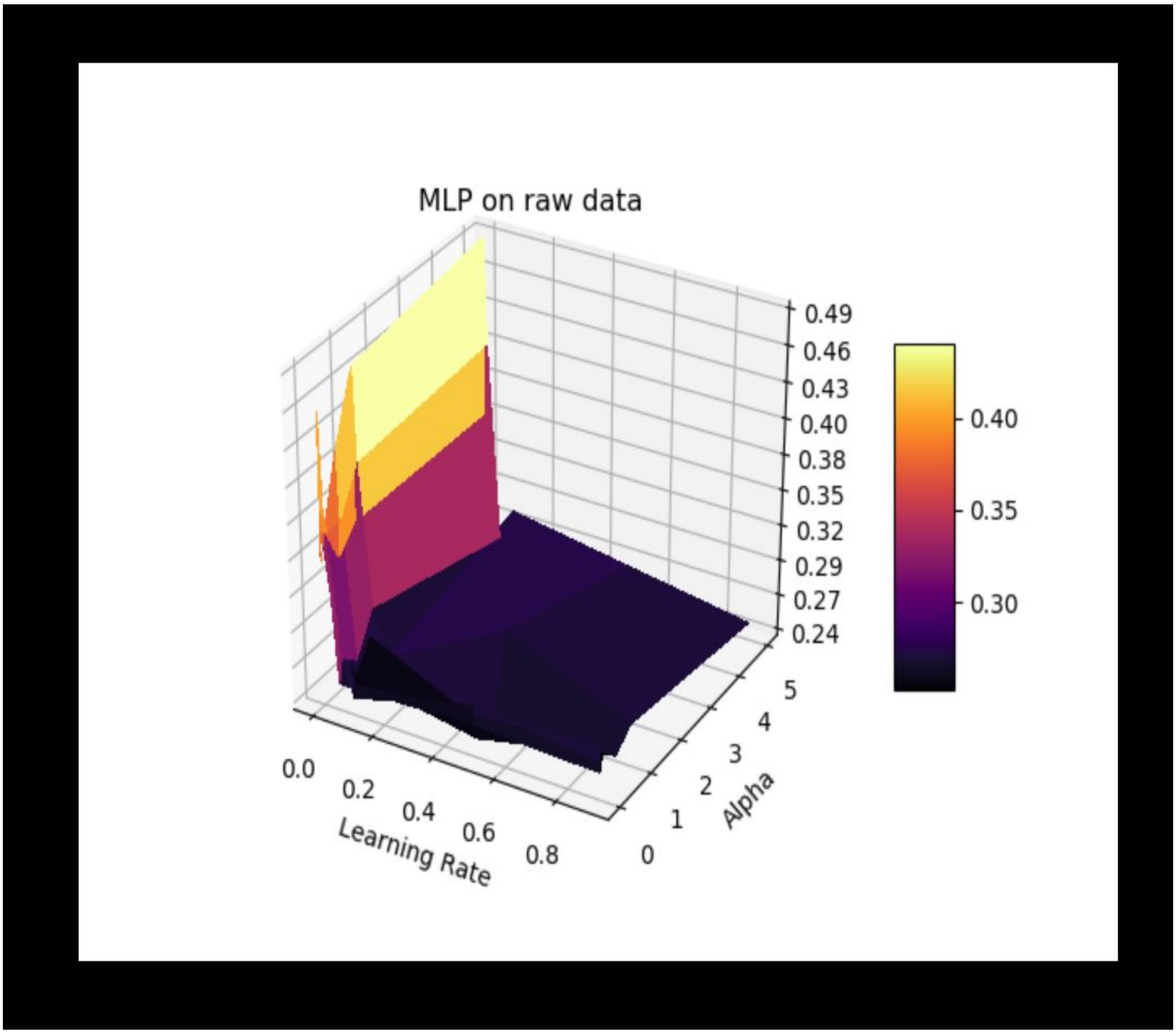
1)LDA + MLP



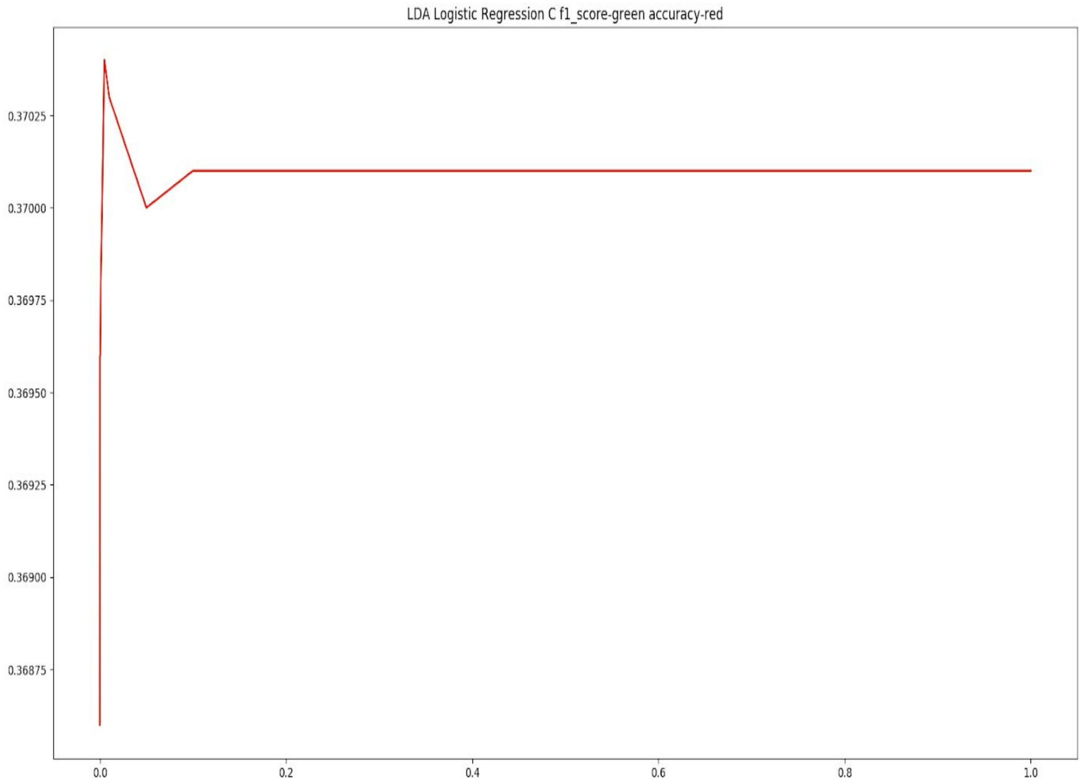
2)PCA + MLP


Info
MLP on pca data
0.46
0.44
0.41
0.39
0.36
0.34
0.31
0.28
0.26
0.23
0.40
0.35
0.30
0.0
0.2
0.4
0.6
0.8
Learning Rate
0
1
2
3
4
5
Alpha

3)RAWDATA + MLP

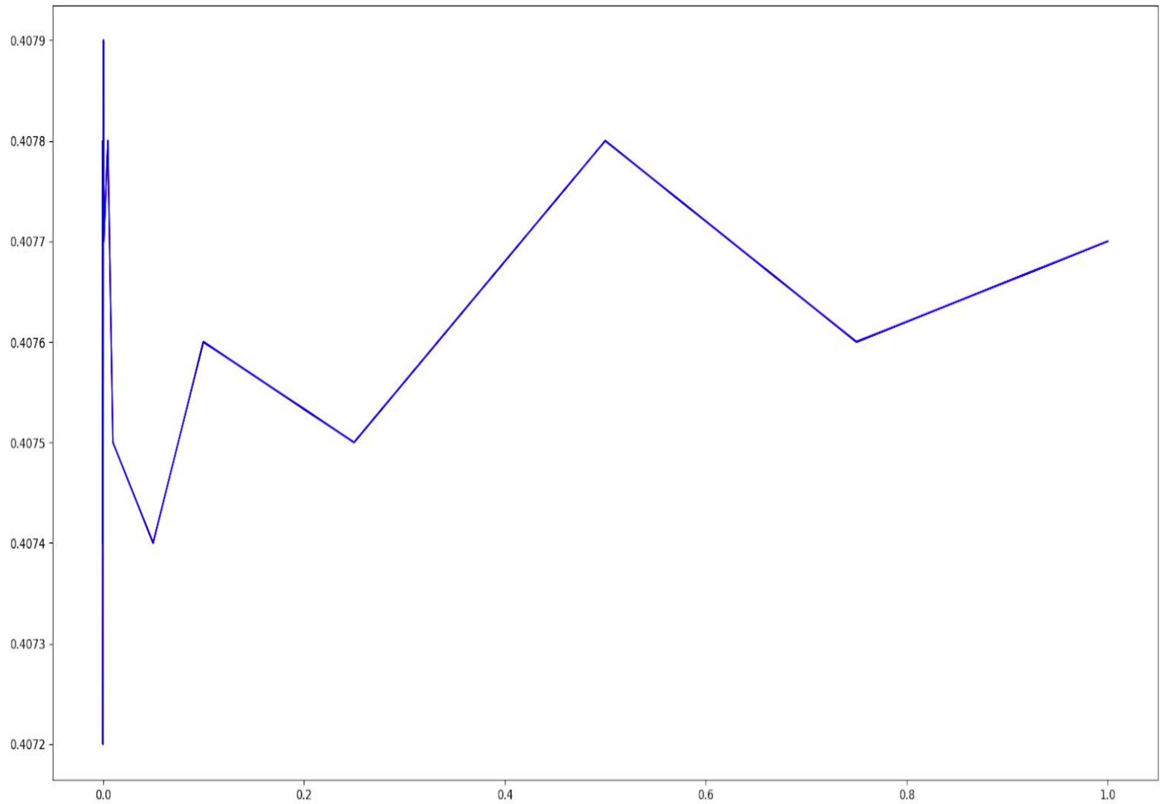


4)LDA + LogReg



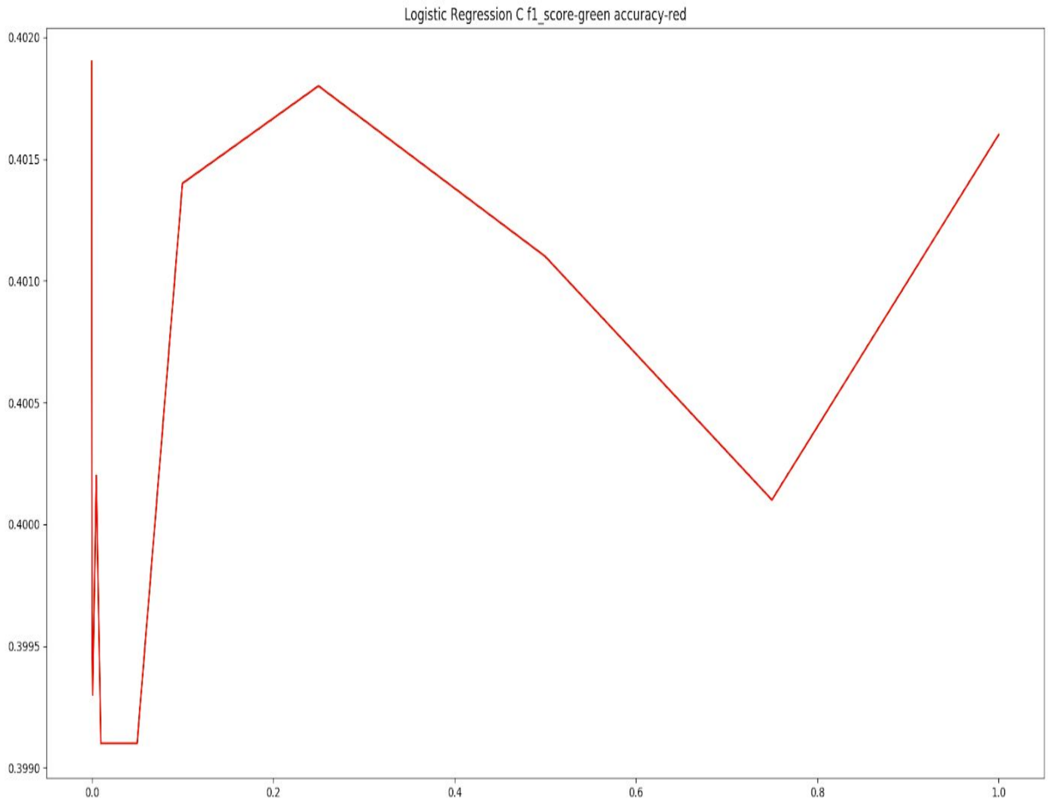
LDA Logistic Regression C f1_score-green accuracy-red
0.37025
0.37000
0.36975
0.36950
0.36925
0.36900
0.36875
0.0
0.2
0.4
0.6
0.8
1.0

5)PCA + LogReg

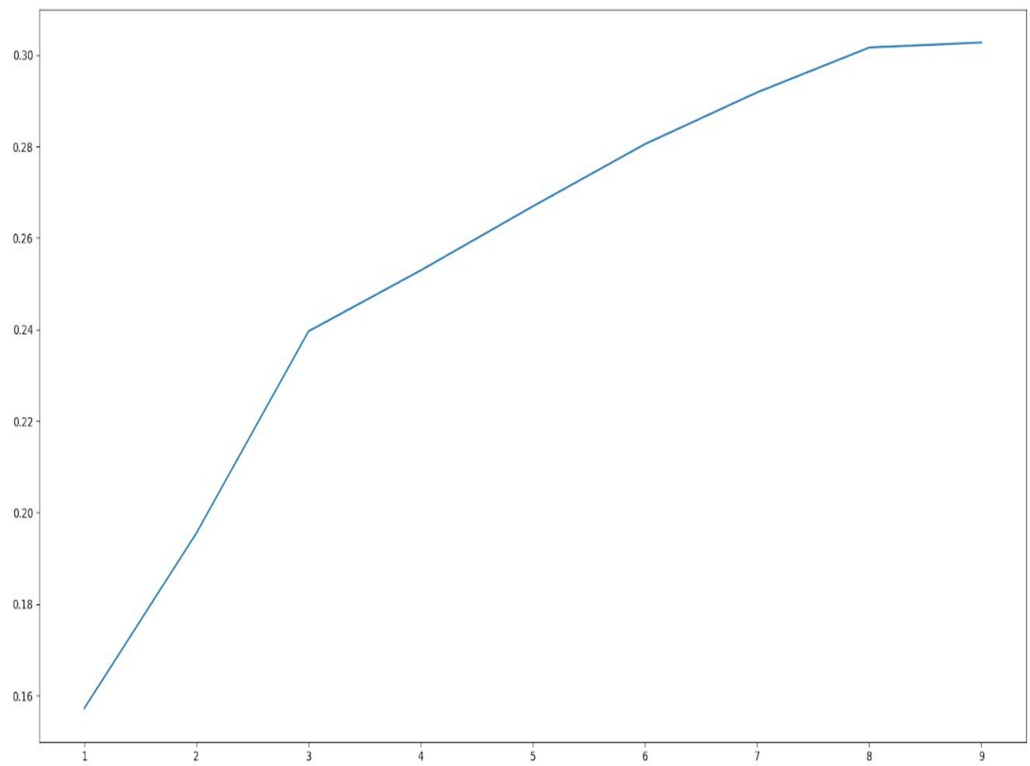


0.4079
0.4078
0.4077
0.4076
0.4075
0.4074
0.4073
0.4072
0.0
0.2
0.4
0.6
0.8
1.0

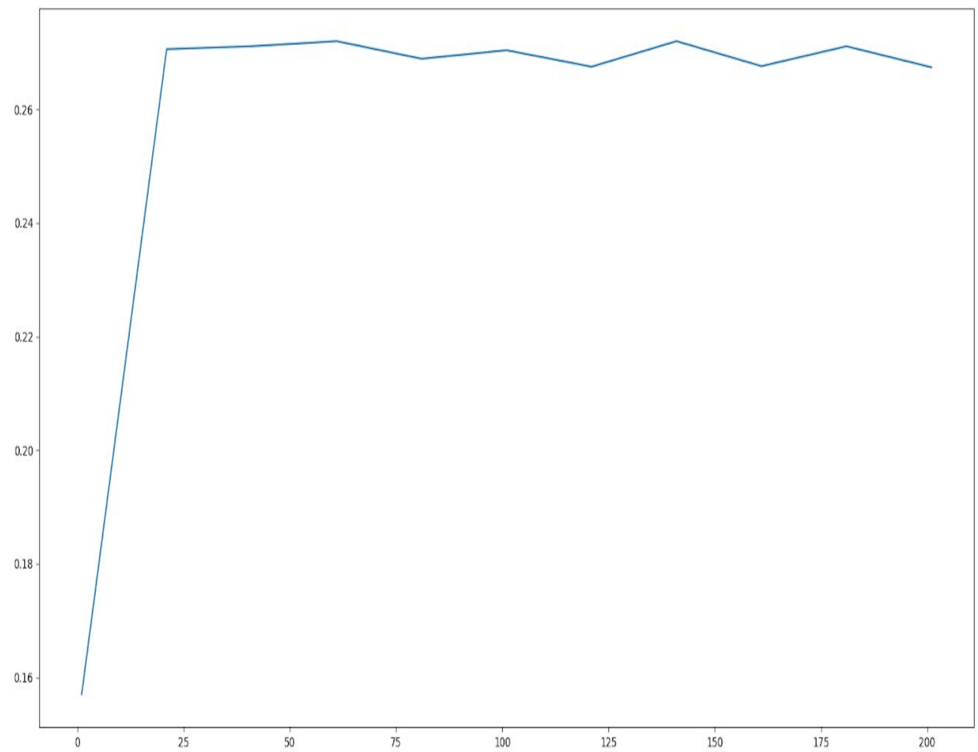
6)RAW DATA + LogReg



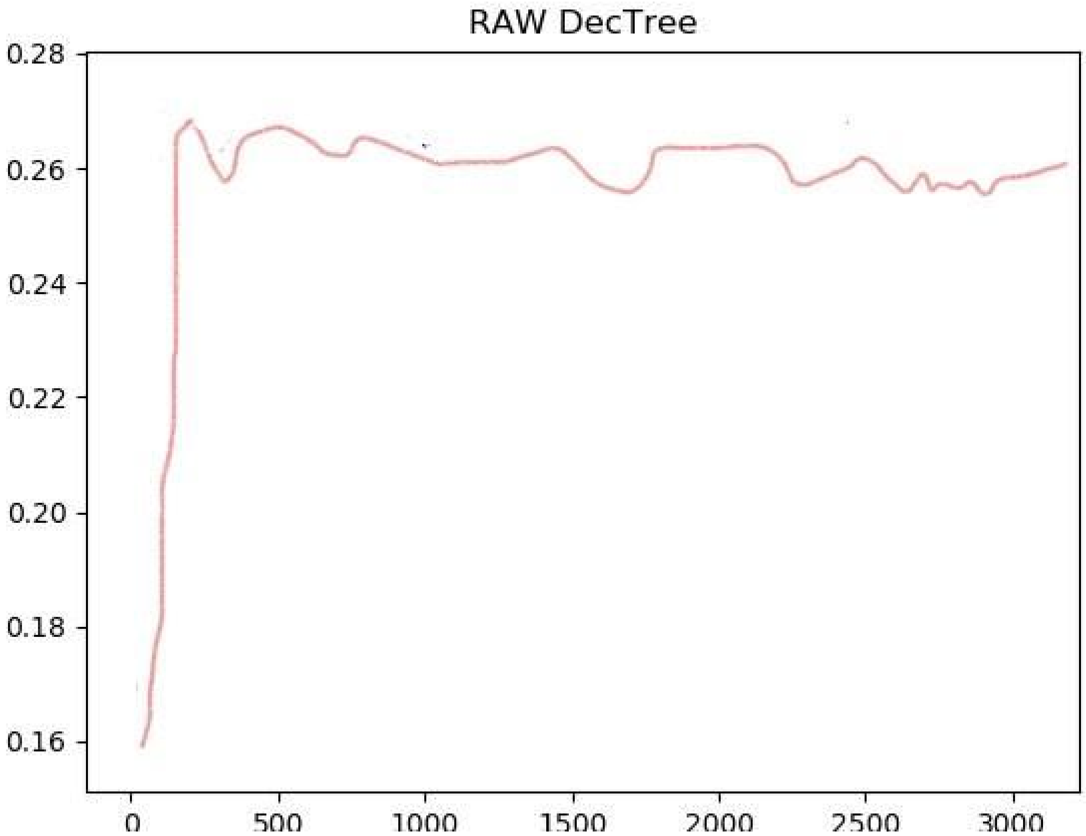
7)LDA + Decision Tree



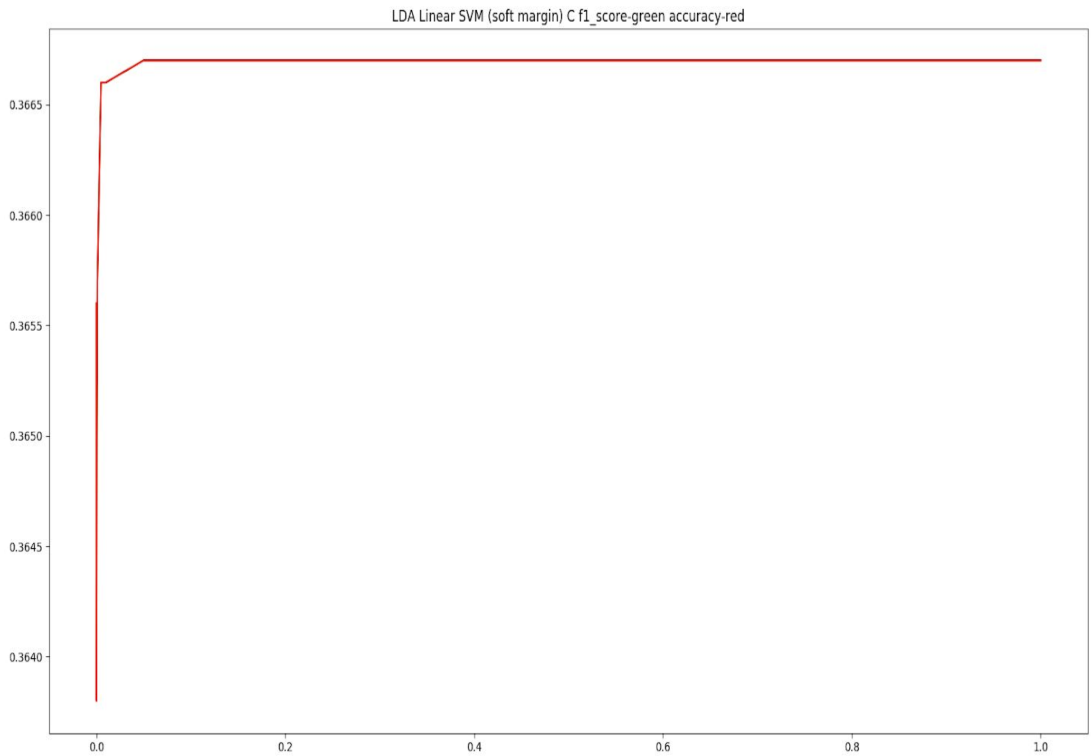
8) PCA + Decision Tree



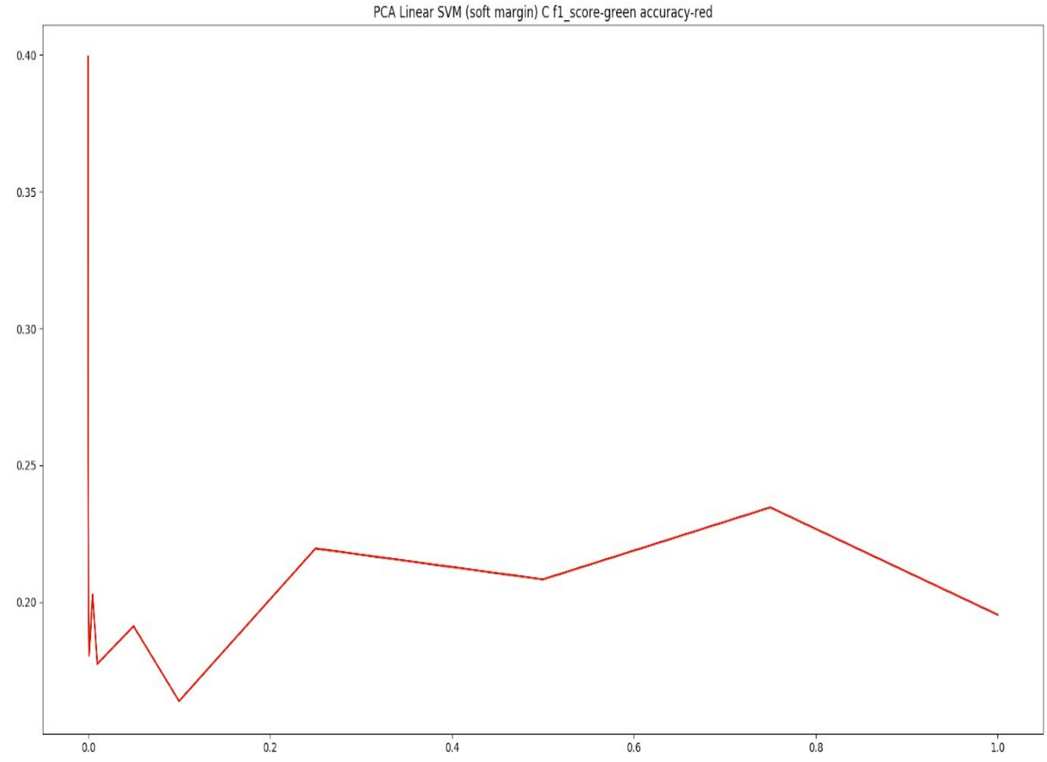
9)RAWDATA + Decision Tree



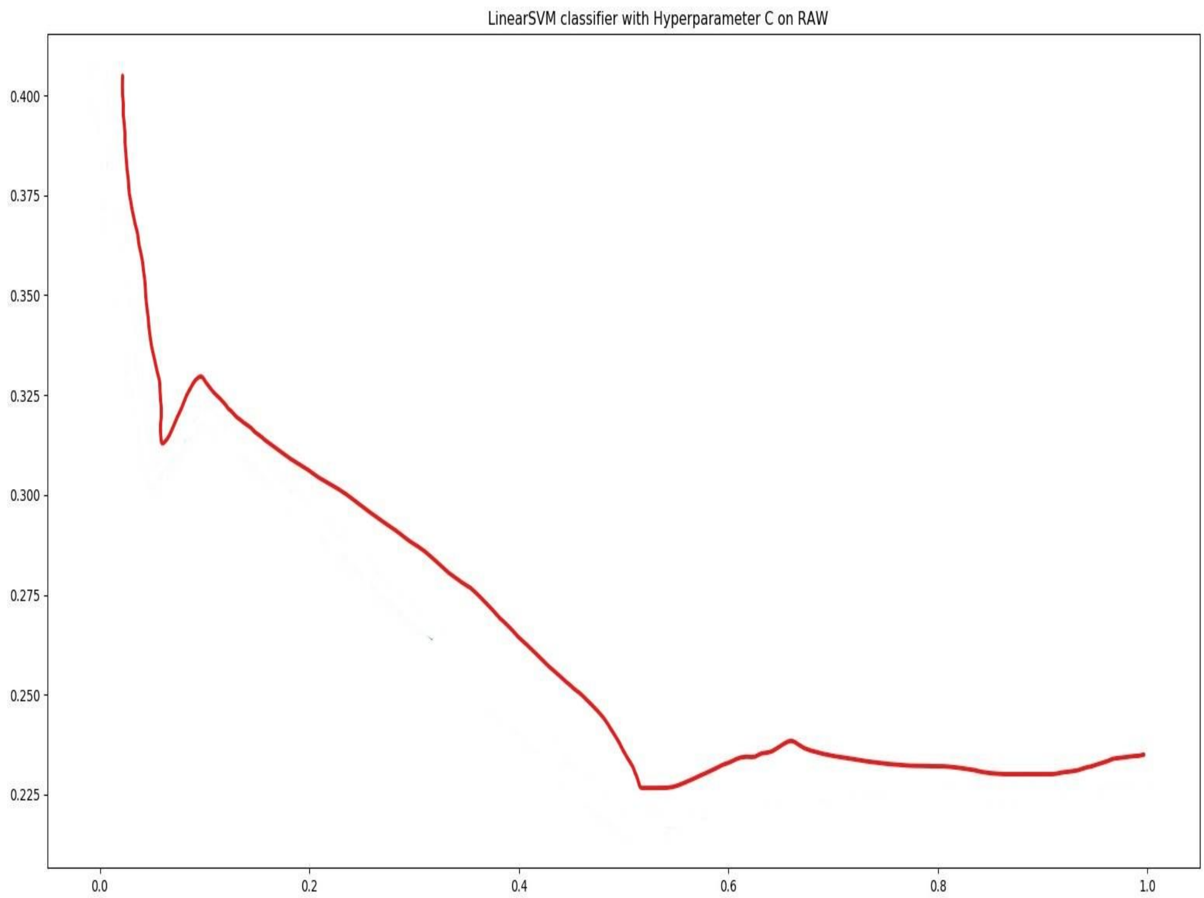
10)LDA + Linear Svm



11)PCA + Linear Svm



12)RAWDATA + Linear Svm



Sidhant Subramanian
20161051