Table of Contents

1. PCA & Eigenfaces

Question 1.1

PCA is applied, and the principal components are obtained; their proportion of variance and cumulative variance are plotted together in Figure 1.

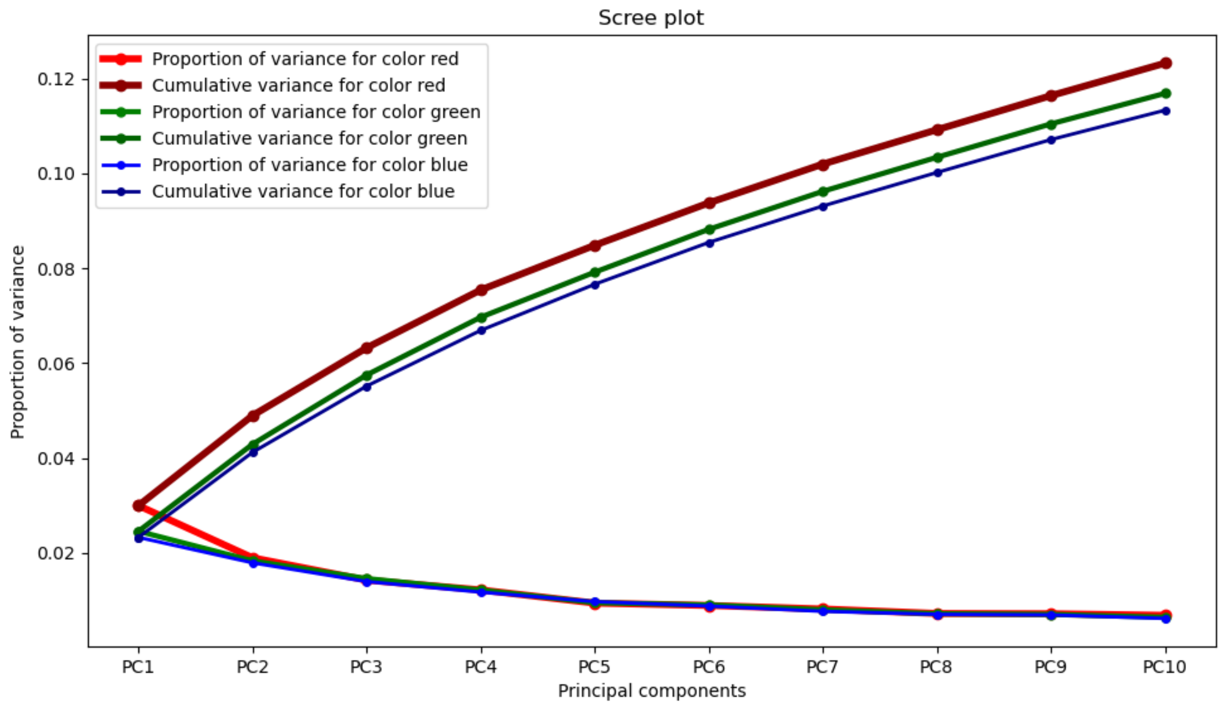


Figure 1: Scree plot for the first ten principal components for three color channels

The exact values can be found by running the script and displaying the variables that the .plotscree(10) method of the PCA class returns. The proportion of variance is how much of the total variance in the data can each principal component represents. k'th Cumulative variance is the sum of the proportion of variances of the first k principal component. From the values obtained from the previously mentioned method, we can say that the first principle of the red color accounts for the 3% of the total variance in the data, and the second one accounts for the 1.9%. The proportion of variance of the first two principal components of the red color map adds up to 4.9%, and this is the corresponding value of cumulative variance for red color at PC2.

Question 1.2

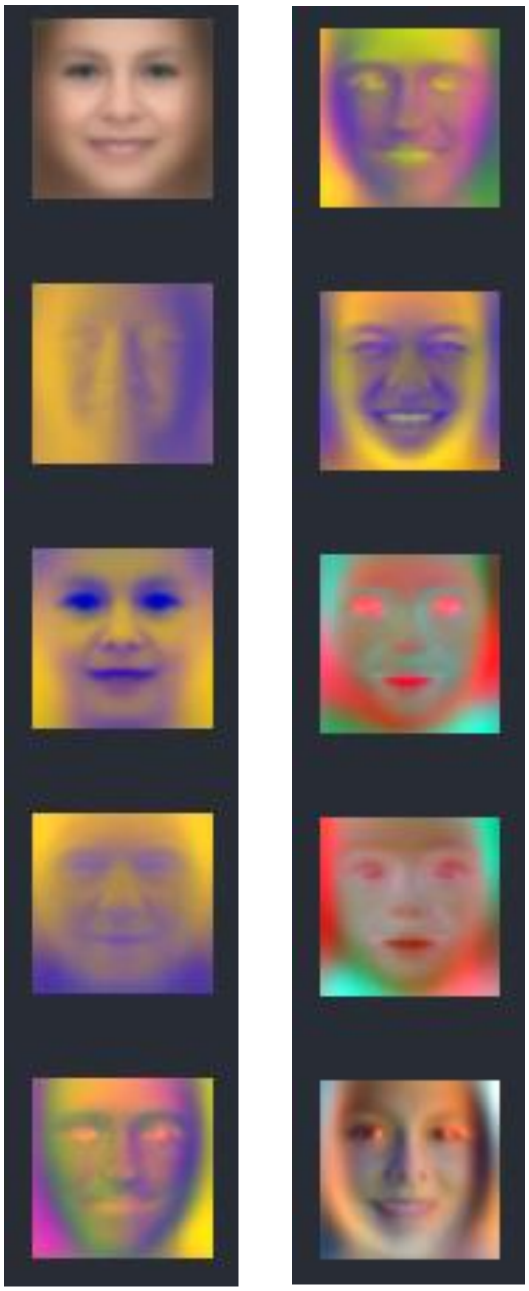


Figure 2: 64x64 Eigenfaces constructed from the first 10 principal components of 3 color channels

The arrays of the images are normalized by multiplying them with the variance extracted from all samples for three color channels and adding the means extracted for color channels. As the principal components with higher degrees model the average face better, they are normalized better with the variation and mean of the data. As the importance of the principal components decrease, they are not perfectly normalized with the mean and variance from the data. This is the reason for the color change in the eigenfaces.

The first eigenface is an average face and encapsulates the shapes. Notice that the background is dark human skin color. If we had not added the means, it would probably be colorless or close to colorless, which means the algorithm has extracted the face features alone in the first principal component reasonably well. The second eigenface looks like a contour that could be used to change the age of the face and increase/decrease the details.

Question 1.3

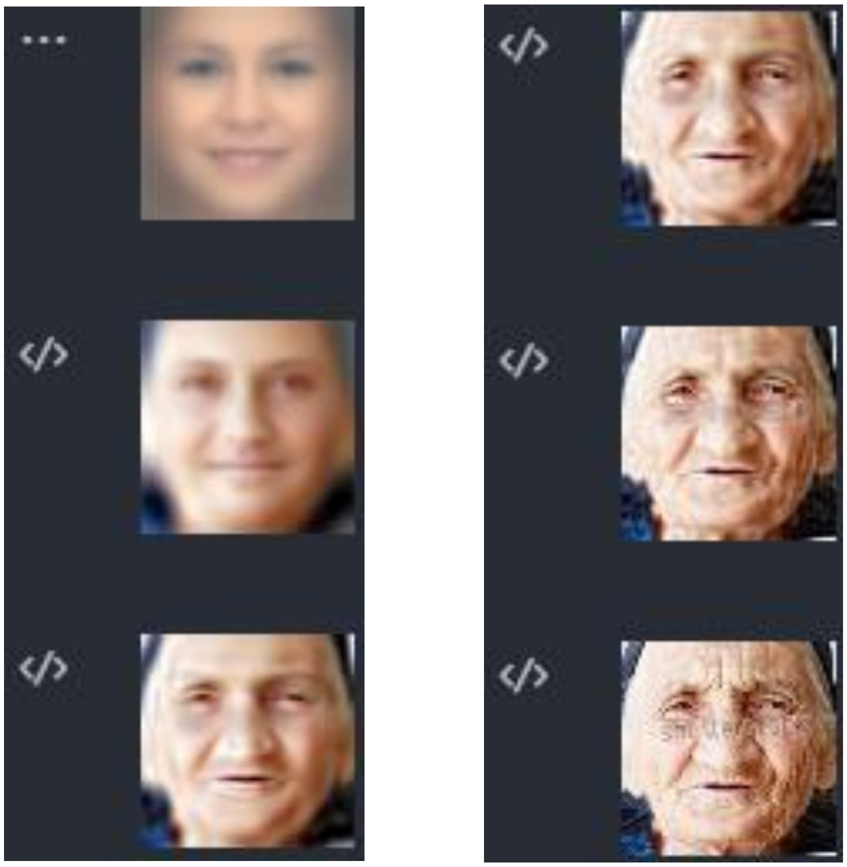


Figure 3: Reconstruction of the first image in the dataset with the given name using 1, 50, 250, 500, 1000, 4096 principal components

I have reconstructed the image using the weight vectors I obtained by dot producting the meanless image data and the eigenvectors. I dot producted the weights and eigenvectors and added the mean of the sample in the mean list.

As can be seen in Figure 3, when only one principal vector is used, the image is the same as the first eigenface in Figure 2 but with less contrast. As seen in the second-row first column image, 50 principal components are not enough for image reconstruction. 250 PCs are enough to get the idea but do not provide enough details. 1000 PCs offer enough details. 4096 PCs result in complete reconstruction.

2. Logistic Regression

Question 2.4

Accuracy tells us the model's capability to categorize the samples correctly. Higher precision means a more conservative model. This is good because the transactions will not likely be blocked if the action is not fraud, but it has a higher chance of missing frauds. Higher recall means the model misses less frauds, but more non-fraud actions are tagged as fraud.

F_1, F_2, and $F_{0.5}$ scores are different, and when we order the models with respect to those scores, we get different sequences. F_2 favors recall, $F_{0.5}$ favors precision, and F_1 is the harmonical mean of the precision and recall.

The best metric for the use case of banking fraud is the F_2 score. The model should favor the recall more than the precision, but both are important, so the recall itself is not the metric of choice. So to balance them, the F_2 score is used, which favors recall more than precision. Recall is more important as the fraud actions should not be missed, but occasional blocks are acceptable as they can also happen when POS machines do not have a nice connection.

The reason for different accuracies at each run is the random initialization of the weights.

Question 2.1

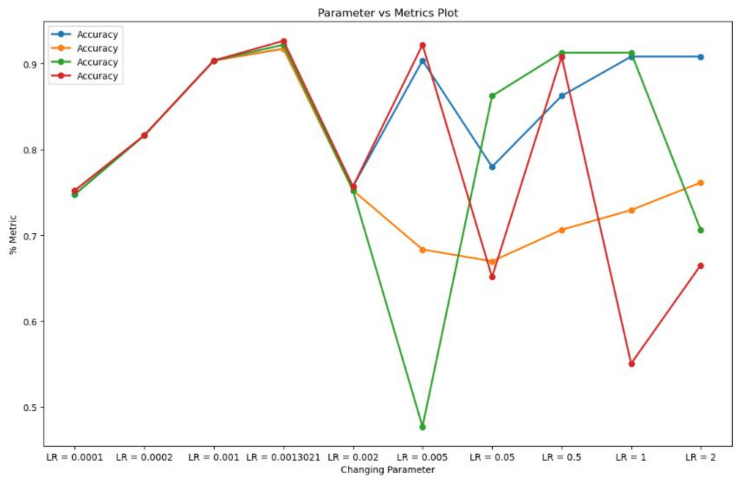


Figure 4: Accuracy measured on the test set after training the model for 40 epochs for different learning rates

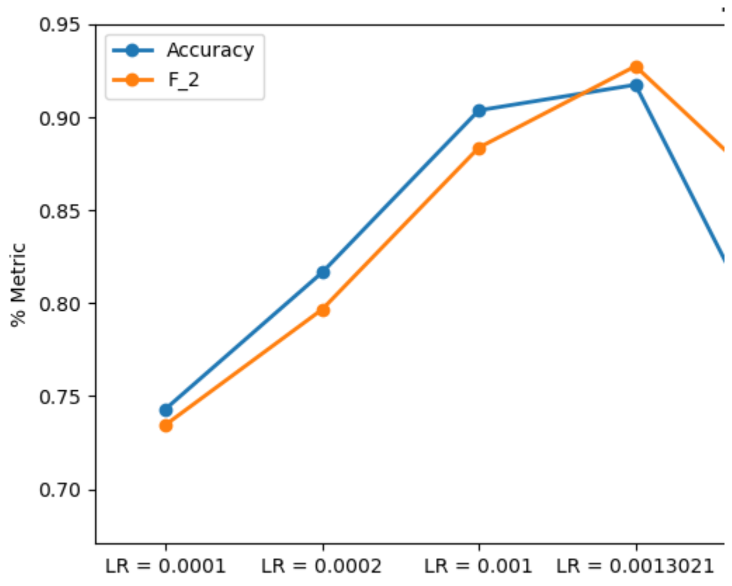
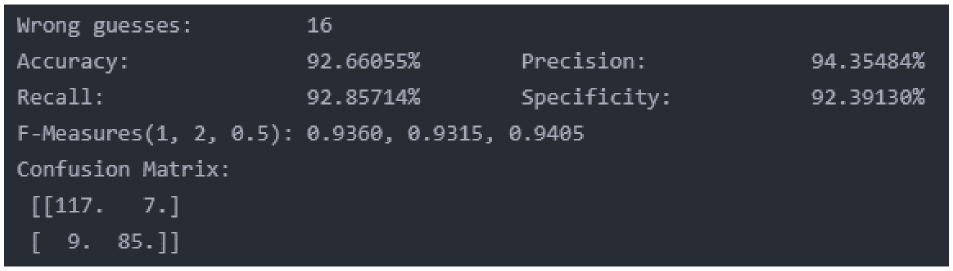


Figure 5: Accuracy and F_2 score measured on the test set after training the model for 40 epochs for different learning rates

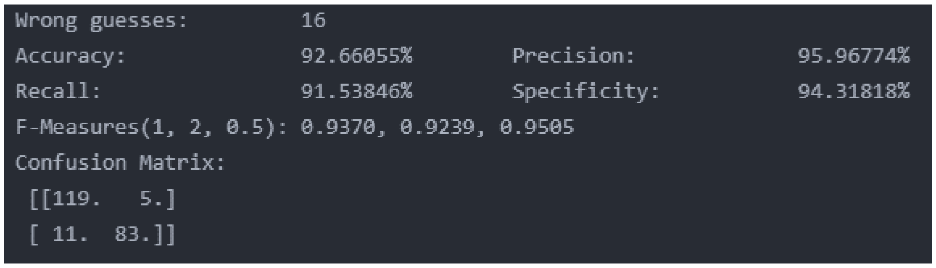
I have only used accuracy to determine the input range to look for. I have also used the F_2 score to validate my choice. In Figure 4, several runs for the same learning rates are given with different colors. Notice that after 0.002 point in Figure 4, the system is unstable and ends up at random points each time. The algorithm jumps around the minima in batch gradient descent, sometimes ending up very close to it and sometimes landing far away from it. In Figure 5, the learning rate of 0.0013021 gives the best accuracy and F_2 score.

I have found the peak value by hand using trial and error, and I thought there should be such a point for the given epoch count because the accuracy percentage was landing on values higher than the peak for higher learning rates from time to time. At the time, the closest learning value that is closest to the one that yields the peak value was 0.001, and I chose random points around it to find the least amount of wrong guesses that I have seen in random results yielded by higher learning rates. There could have been a better and automated approach to finding it, but I was very close to it already and only needed to find it once, so I did it by hand.



```
Wrong guesses:         16
Accuracy:              92.66055%        Precision:             94.35484%
Recall:                92.85714%        Specificity:           92.39130%
F-Measures(1, 2, 0.5): 0.9360, 0.9315, 0.9405
Confusion Matrix:
 [[117.   7.]
 [  9.  85.]]
```

Figure 6: Running the best model on the test set after the best learning rate of 0.0013021



```
Wrong guesses:         16
Accuracy:              92.66055%        Precision:             95.96774%
Recall:                91.53846%        Specificity:           94.31818%
F-Measures(1, 2, 0.5): 0.9370, 0.9239, 0.9505
Confusion Matrix:
 [[119.   5.]
 [ 11.  83.]]
```

Figure 7: Running the model on the test set for 10+ runs with the learning rate = 0.5

Figures 6 and 7 are given together for comparison reasons. It is possible to reach a good result with higher learning rates, but it is more like a dice roll than a systematic approach. The best learning rate is used in Figure 6, and its metrics are provided in the figure.

Question 2.2

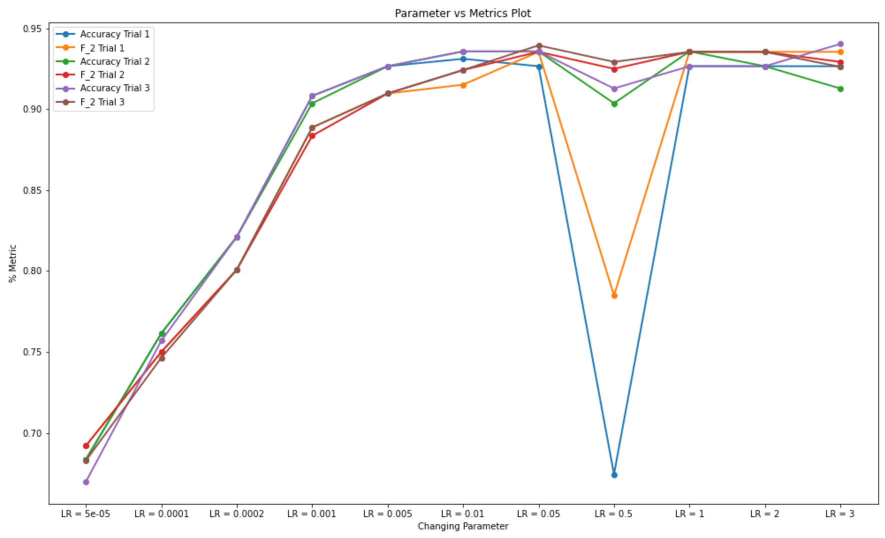


Figure 8: Metrics measured on a test set after training the model with mini-batch gradient descent for 40 epochs for different learning rates

The learning rate of 0.05 gives the best result as the values very close to it result in the same or worse for both metrics. I have also used the F_2 score alongside the accuracy. Mini-batch gradient descent seems to be unstable when the learning rate is greater than a value between 0.05 and 0.5. I have run the code several times and observed that 0.05 is a stable value and gives the best overall values for accuracy and used the F_2 score. The best learning rate is 0.05 for the mini-batch gradient descent algorithm in this case.

The reason for instability is the same as the instability on the batch gradient descent. When the learning rate is greater than some value, the system becomes unstable. The reason for getting different accuracy and F_2 score is random initialization of weights and random shuffling at each epoch.

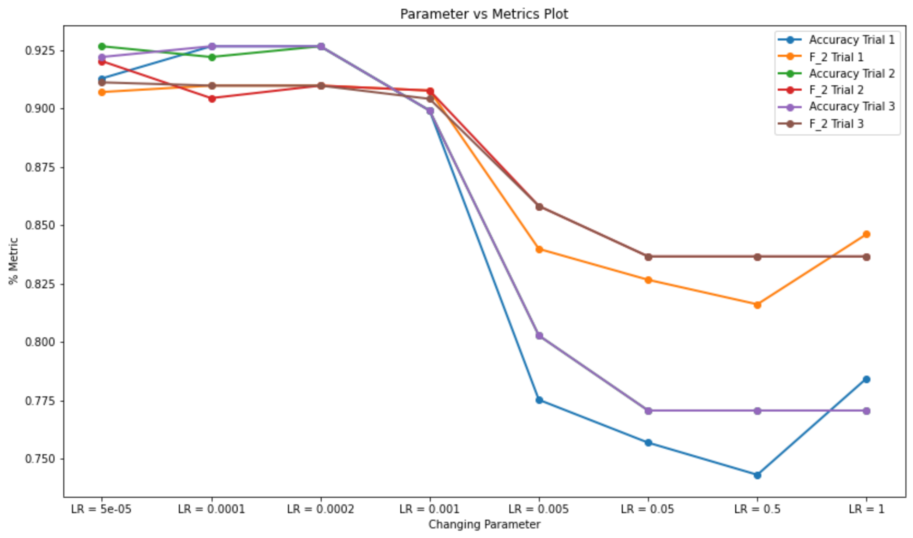
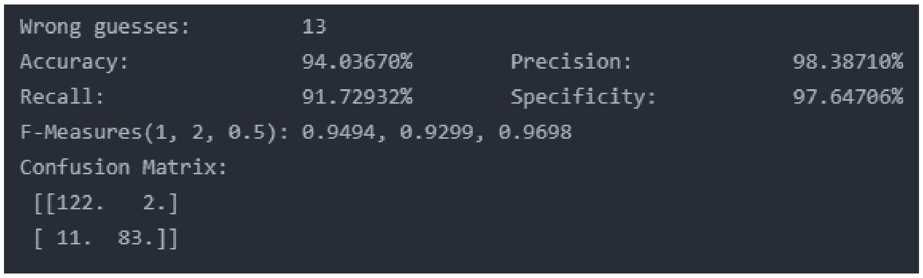


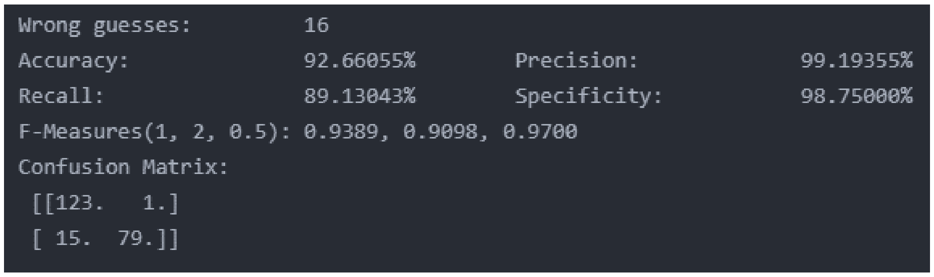
Figure 9: Metrics measured on the test set after training the model with stochastic gradient descent for 40 epochs for different learning rates

Similar metrics are used in measuring the stochastic gradient descent algorithm's performance. Notice that similar F_2 and accuracy scores are obtained for learning rates 0.00005, 0.0001, and 0.0002, but 0.0002 is the most stable one. No matter how many times the code is run, 0.0002 yields the same F_2 and accuracy scores. I chose 0.0002 as the optimum learning rate for the stochastic gradient descent algorithm as all three of them yield similar metric values, so choosing the more stable one makes the system more reliable, and bigger learning rates can increase the required time for convergence.



```
Wrong guesses:         13
Accuracy:              94.03670%       Precision:           98.38710%
Recall:                91.72932%       Specificity:         97.64706%
F-Measures(1, 2, 0.5): 0.9494, 0.9299, 0.9698
Confusion Matrix:
 [[122.   2.]
 [ 11.  83.]]
```

Figure 10: Running the best mini-batch descent model on the test set after the best learning rate of 0.05



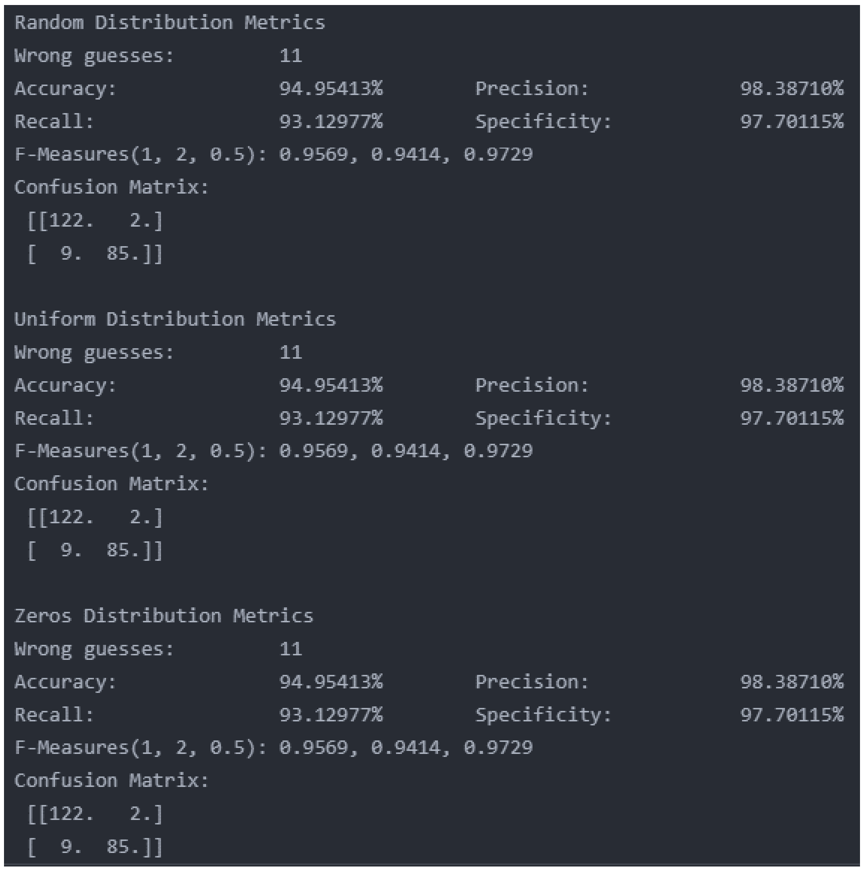
```
Wrong guesses:         16
Accuracy:              92.66055%       Precision:           99.19355%
Recall:                89.13043%       Specificity:         98.75000%
F-Measures(1, 2, 0.5): 0.9389, 0.9098, 0.9700
Confusion Matrix:
 [[123.   1.]
 [ 15.  79.]]
```

Figure 11: Running the best stochastic gradient descent model on the test set after the best learning rate of 0.0002

The models are trained with the optimum learning rate found for each of them, and confusion metrics are extracted from a trial on the test set. Comparing the two outputs in Figures 10 and 11, the mini-batch gradient descent is superior to stochastic gradient descent in all metrics except for precision and specificity.

Question 2.3

The best F_2 score belongs to the mini-batch gradient descent model, so I chose it. On top of that, it also holds the record for the best accuracy. Though it is slightly slower than full-batch gradient descent, it is significantly faster than stochastic gradient descent. I did not try to measure the precisely passed time, but VSCode automatically measured it up to 1 significant digit. The results are 0.2s, 0.4s, and 10.3s for full-batch, mini-batch, and stochastic gradient descent.



```
Random Distribution Metrics
Wrong guesses:          11
Accuracy:               94.95413%        Precision:              98.38710%
Recall:                 93.12977%        Specificity:            97.70115%
F-Measures(1, 2, 0.5): 0.9569, 0.9414, 0.9729
Confusion Matrix:
 [[122.   2.]
 [  9.  85.]]

Uniform Distribution Metrics
Wrong guesses:          11
Accuracy:               94.95413%        Precision:              98.38710%
Recall:                 93.12977%        Specificity:            97.70115%
F-Measures(1, 2, 0.5): 0.9569, 0.9414, 0.9729
Confusion Matrix:
 [[122.   2.]
 [  9.  85.]]

Zeros Distribution Metrics
Wrong guesses:          11
Accuracy:               94.95413%        Precision:              98.38710%
Recall:                 93.12977%        Specificity:            97.70115%
F-Measures(1, 2, 0.5): 0.9569, 0.9414, 0.9729
Confusion Matrix:
 [[122.   2.]
 [  9.  85.]]
```

Figure 12: Metrics calculated for mini-batch gradient descent initialized with different weight distribution

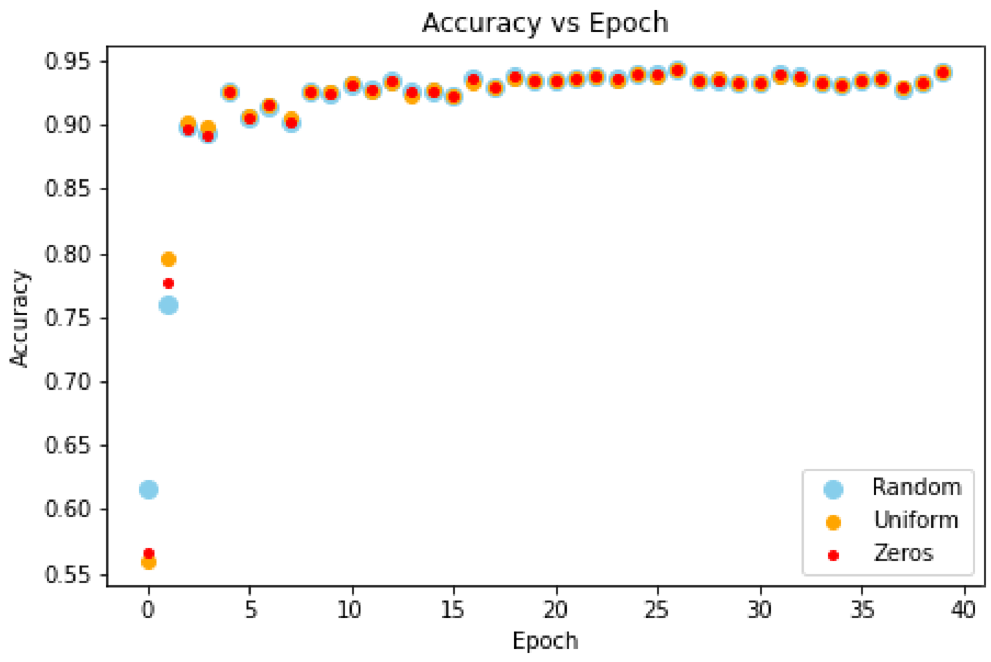
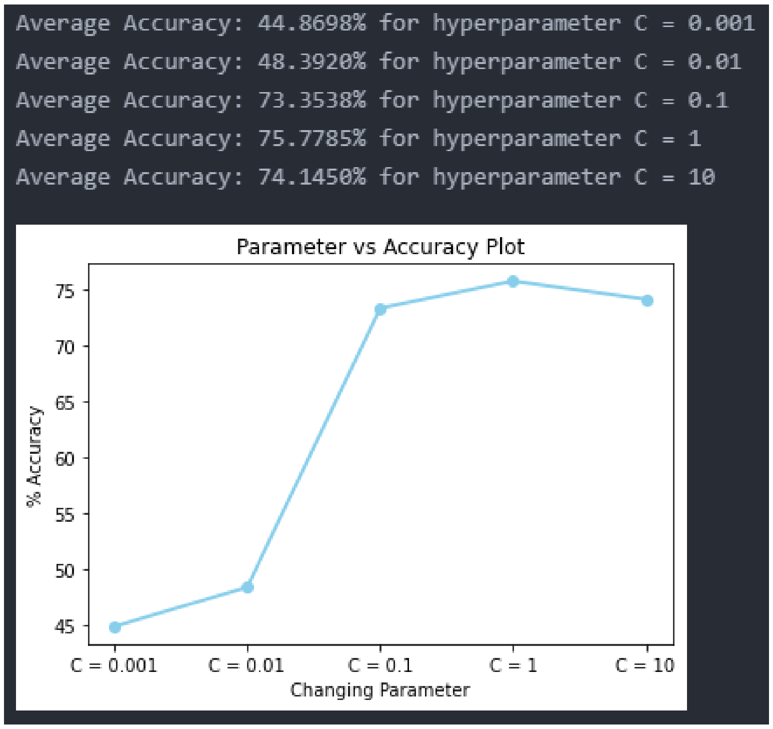


Figure 13: Accuracy of the models with different initial weights on the training set

Mini-batch gradient descent performed optimization successfully for different weight initializations. Numpy seed is fixed for sample shuffling so that we could observe the effects of the different weight initializations better. The result is the same for all of them and is the best score among the previously seen trials. If we inspect the plot in Figure 13, we can say that the models converge in less than 40 epochs, and all converge similarly. Random and zero initialization behave the same way except for the tiny difference in the first two epochs, whereas uniform weight distribution is slightly better in the first epochs. The similarity of behavior in the random and zero distribution is that the random distribution's weights are very close to zero but slightly greater than zero. In the second epoch, they are updated similarly, so they end up in a similar spot, but as the uniform distribution starts different, it follows a few different steps.

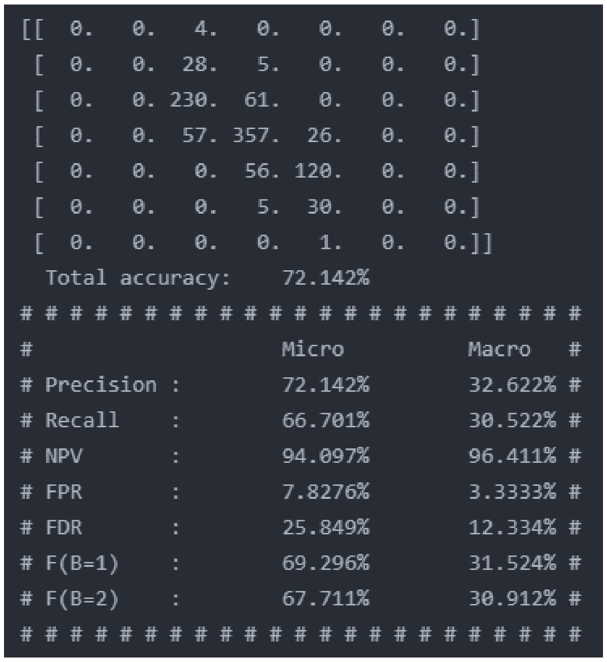
3. Support Vector Machines (SVMs)

Question 3.1



```
Average Accuracy: 44.8698% for hyperparameter C = 0.001
Average Accuracy: 48.3920% for hyperparameter C = 0.01
Average Accuracy: 73.3538% for hyperparameter C = 0.1
Average Accuracy: 75.7785% for hyperparameter C = 1
Average Accuracy: 74.1450% for hyperparameter C = 10
```

Figure 14: Parameter vs. Accuracy plot for changing C on a linear soft margin model

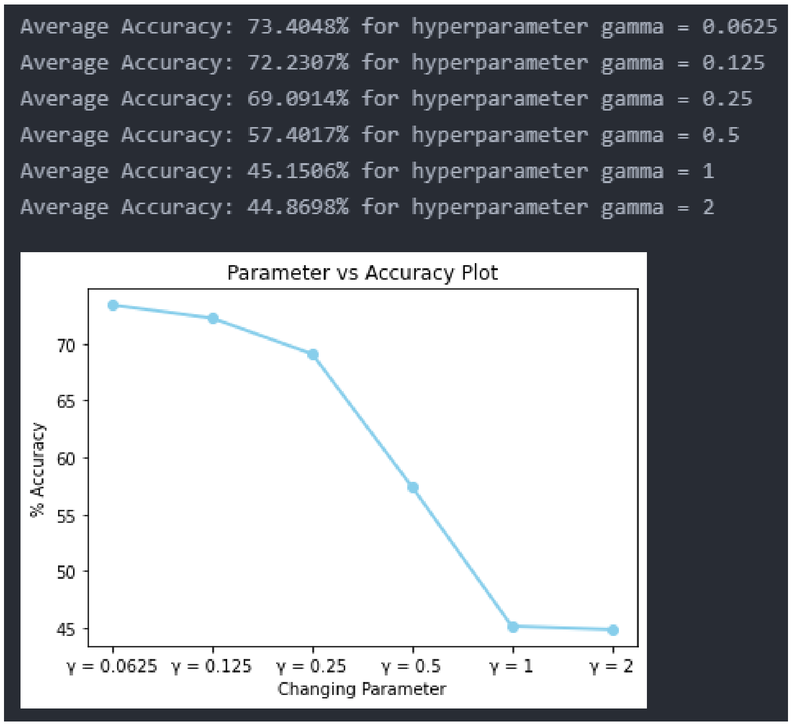


```
[[  0.   0.   4.   0.   0.   0.   0.]
 [  0.   0.  28.   5.   0.   0.   0.]
 [  0.   0. 230.  61.   0.   0.   0.]
 [  0.   0.  57. 357.  26.   0.   0.]
 [  0.   0.   0.  56. 120.   0.   0.]
 [  0.   0.   0.   5.  30.   0.   0.]
 [  0.   0.   0.   0.   1.   0.   0.]]
  Total accuracy:    72.142%
# # # # # # # # # # # # # # # # # # # # # # #
#                    Micro          Macro   #
# Precision :        72.142%        32.622% #
# Recall    :        66.701%        30.522% #
# NPV       :        94.097%        96.411% #
# FPR       :        7.8276%        3.3333% #
# FDR       :        25.849%        12.334% #
# F(B=1)    :        69.296%        31.524% #
# F(B=2)    :        67.711%        30.912% #
# # # # # # # # # # # # # # # # # # # # # # #
```

Figure 15: Confusion matrix and metrics on the test set for C = 1

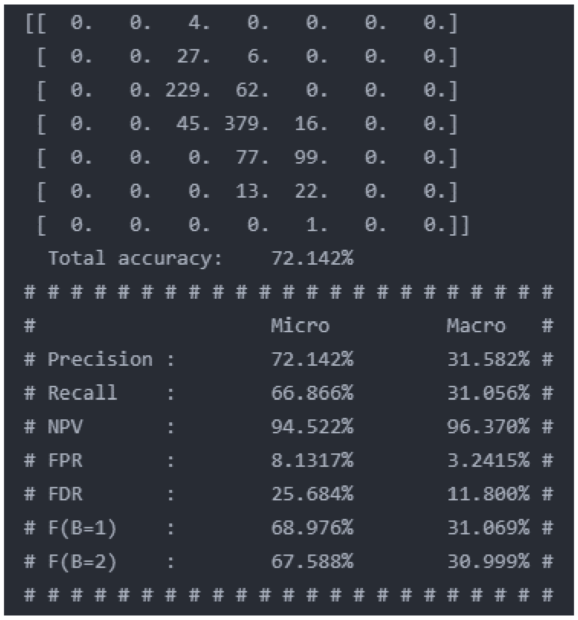
The best C value turned out to be 1, as it yielded the highest accuracy in Figure 14. The trained model with hyperparameter C $= 1$ is used to get the estimates based on the samples in the test set then a confusion matrix is obtained. The metrics are found and put into an ASCII table nicely by a function defined in the script, which can be seen in Figure 15.

Question 3.2



```
Average Accuracy: 73.4048% for hyperparameter gamma = 0.0625
Average Accuracy: 72.2307% for hyperparameter gamma = 0.125
Average Accuracy: 69.0914% for hyperparameter gamma = 0.25
Average Accuracy: 57.4017% for hyperparameter gamma = 0.5
Average Accuracy: 45.1506% for hyperparameter gamma = 1
Average Accuracy: 44.8698% for hyperparameter gamma = 2
```

Figure 16: Parameter vs. Accuracy plot for changing gamma on RBF kernel



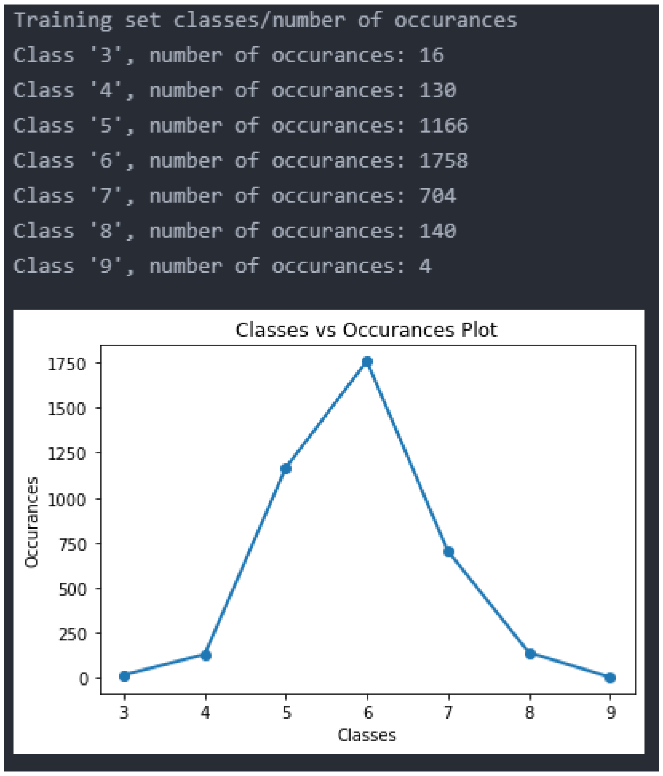
```
[[  0.   0.   4.   0.   0.   0.   0.]
 [  0.   0.  27.   6.   0.   0.   0.]
 [  0.   0. 229.  62.   0.   0.   0.]
 [  0.   0.  45. 379.  16.   0.   0.]
 [  0.   0.   0.  77.  99.   0.   0.]
 [  0.   0.   0.  13.  22.   0.   0.]
 [  0.   0.   0.   0.   1.   0.   0.]]
  Total accuracy:    72.142%
# # # # # # # # # # # # # # # # # # # # # # #
#                    Micro         Macro   #
# Precision :        72.142%       31.582% #
# Recall    :        66.866%       31.056% #
# NPV       :        94.522%       96.370% #
# FPR       :        8.1317%       3.2415% #
# FDR       :        25.684%       11.800% #
# F(B=1)    :        68.976%       31.069% #
# F(B=2)    :        67.588%       30.999% #
# # # # # # # # # # # # # # # # # # # # # # #
```

Figure 17: Confusion matrix and metrics on the test set for C = 0.1 and gamma = 0.0625

The best γ value turned out to be 0.0625, as it yielded the highest accuracy in Figure 16. The trained model with hyperparameter $\mathrm{C} = 0.1$ and $\gamma = 0.0625$ are used to get the estimates based on the samples in the test set then a confusion matrix is obtained. The metrics are found and put into an ASCII table nicely by a function defined in the script, which can be seen in Figure 17.

Question 3.3

Based on the confusion matrices in Figures 15 and 17, we can see the imbalance in the test set. It is possible to understand that the training set is imbalanced as in the confusion matrix of the test sets, only 3 classes are ever correctly classified, and all the predictions belong to the same 3 classes as well. We can fathom that the algorithm is biased to choose those 3 classes as opposed to the other 4.



```
Training set classes/number of occurances
Class '3', number of occurances: 16
Class '4', number of occurances: 130
Class '5', number of occurances: 1166
Class '6', number of occurances: 1758
Class '7', number of occurances: 704
Class '8', number of occurances: 140
Class '9', number of occurances: 4
```

Figure 18: Classes vs Occurrences count in the training dataset

The reason of the previously described behaviour of the classifier can be explained using Figure 18. As the classes 5, 6, 7 dominate in the dataset by a significant magnitude, the classifier is inclined to classify the sample as 5, 6 or 7.

Precision and recall are good metrics to use in biased datasets as they measure how well the data is classified correctly among two different subsets. We can use macro precision and recall so that the classes will be equal and it will be noticed if the classifier cannot detect one class well.

F_1 score can also be used as it is measure that consists of both precision and recall and we do not have a preference for one over another so it is okay to use $\beta = 1$. Calculating it as macro will be more useful for the same reason pointed out for precision and recall.