CS521 Assignment 3: Dimensionality Reduction & SVMs

Ashwin Goyal
2018meb1214@iitrpr.ac.in

Indian Institute of Technology
Ropar, Punjab

Abstract

The purpose of this assignment is to help us understand the various methods of Dimensionality Reduction and Visualization (PCA,LDA,t-SNE) and the implementation of SVMs. This assignment helped me gain a deep understanding of these techniques, and has upskilled me in terms of the various tools that I can now use for better data visualisation and effective dimensionality reduction.

1 Introduction

The assignment is divided into 3 parts. In the first part, I've performed PCA and computed the Eigen faces for face recognition. For this, I've used the **"Labeled Faces in the Wild"** data set.

In the second part, I've used the **Fisher Iris** data set. On this data set, I've used Principal Component Analysis, Linear Discriminant Analysis and t-SNE. These are used for reducing the dimensions and for better visualization

In the third and last part, I have classified data with Linear and Non-linear SVMs. Here also, the Iris data set has been used.

2 Q1: Principal Component Analysis and Eigenfaces for Face Recognition

2.1 Importing libraries, loading the dataset, Data preprocessing and train/test split

The libraries imported by me are: *Numpy, Pandas, Seaborn, and Matplotlib, SKLearn*. After that, I've imported the Labeled Faces in the Wild dataset using the data sets package in sklearn. For this, I've used: *lfw_dataset = sklearn.datasets.fetch_lfw_people(min_faces_per_pers*
This helped me obtain 100 faces/individual from the dataset.

The data set has then been split by using the train_test_split functionality of sklearn. I have taken the training set size to be 70% of the data set and the testing set is 30% of the data set. After this, I've preprocessed the Training subset of the data set through scaling the features. This is done to standardise the features.

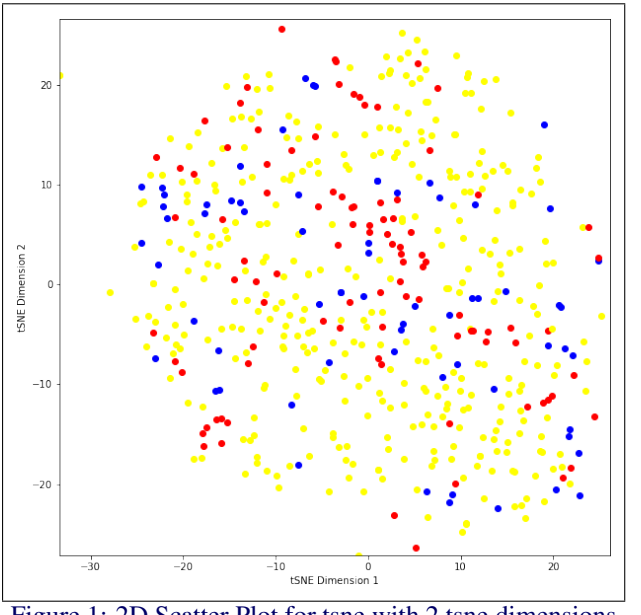


Figure 1: 2D Scatter Plot for tsne with 2 tsne dimensions

2.2 Principal Component Analysis (PCA)

PCA helps us to reduce the number of dimensions in a data set without losing valuable and important information. I've imported the PCA functionality from the decomposition package of sklearn library. PCA is based on keeping the maximum amount of variation possible in the least number of dimensions(features). In other words, it tries to preserve maximum information while reducing the number of dimensions. Here, I have performed PCA with the number of components as 100, as required in the question. The shape of X_train after PCA is (798, 100).

2.3 Choosing 3 personalities, projecting them to 2/3 dimensions and visualising through t-SNE

1. I have first found a subset of the original data set which corresponds to any three personalities (I have chosen 'Colin Powell', 'Donald Rumsfeld', 'George W Bush', the first 3 personalities in the data set). This can be done using a for loop and iterating over i to find a $X\_train_reduced, which has less ernumber of samples as compared to the original X or X\_train$.

2. We have already reduced the number of dimensions to 100 through, which means I now have a 100 dimensional vector. Next, I have used TSNE *(from the manifold package of sklearn)* and through it, I have further reduced the dimensions to 2, using the number of components=2. This reduces it to a 2 dimensional (or 3 dimensional) vector.

3. For the visualization, I have plotted a scatter plot with the **first reduced tsne dimension** replacing the x-axis and the y-axis is replaced by the **second reduced tsne dimension** The plot obtained is shown in fig. 1. In general, t-SNE preserves only the small pairwise distances, whereas on the other hand, PCA focuses on preserving large pairwise distances. This maximize variance in case of PCA.

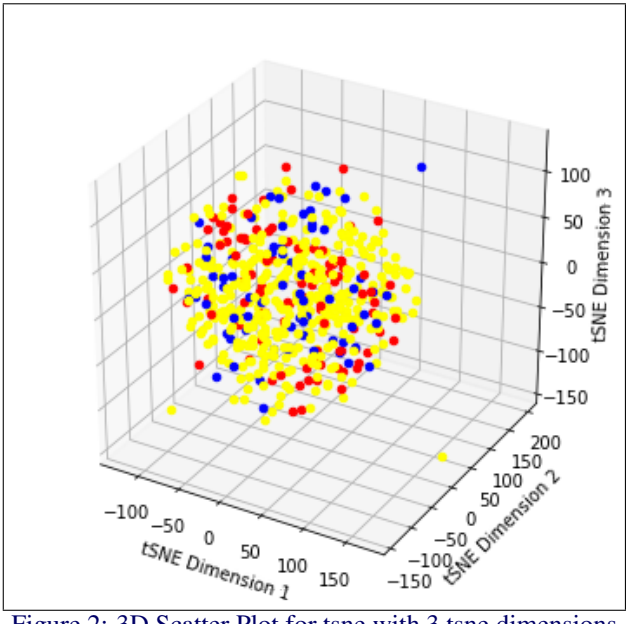


Figure 2: 3D Scatter Plot for tsne with 3 tsne dimensions

I have also used tsne for 3 components and reduced it to 3 dimensions. The 3D scatter plot is shown in fig 2.

From these 2 graphs, *no clear clusters were obtained*. This was essentially because it is practically not possible to keep information of the faces within 2/3 dimensions only. **While converting from 100 dimensions to 2/3 dimensions, most of the informative features and most of the attributes which help in differentiating between faces were lost.** *Already, we had reduced the dimenions from 2914 to 100, and now we are reducing it to 2/3.* The main purpose of t-SNE is the visualization through reducing the dimensions and it does not necessarily mean that the obtained dimensionality reduction is always useful. The input information/features of the faces are no longer very meaningful and we do not obtain distinct clusters now. In other words, the reduced dimensions have very less information are not able to classify the faces according to the personality.

2.4 Using K-nearest neighbours classifier

Next, I have used the K-nearest neighbours classifier for assigning each test face to the identity which has the closest training example. The *KNeighborsClassifier* function has been imported from the *neighbors* package of sklearn. Here, I have printed the classification report and the confusion matrix (imported from the metrics package of sklearn).The results (for all identities in the training set) are shown in fig 3.

From the classification report, we can see that the accuracy is 67% for the 5 classes, which is acceptable, given the complex nature of the dataset.

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[ 51   1  10   1   1]
 [  5  13   5   0   0]
 [ 21   3 138   2   2]
 [  9   2  12  11   8]
 [  7   2  21   1  16]]
              precision    recall  f1-score   support

           0       0.55      0.80      0.65        64
           1       0.62      0.57      0.59        23
           2       0.74      0.83      0.78       166
           3       0.73      0.26      0.39        42
           4       0.59      0.34      0.43        47

    accuracy                           0.67       342
   macro avg       0.65      0.56      0.57       342
weighted avg       0.68      0.67      0.65       342
```

Figure 3: Classification report for KNN

2.5 Plotting the first 20 eigenfaces

Eigen vectors are the 'core' of PCA, the eigenvectors (and corresponding eigenvalues) of a covariance matrix represent the principal components which we obtain after dimensionality reduction. They determine the directions of the new reduced feature space, and their magnitude is determined by eigenvalues. In face recognition problems, eigenvectors when seen in the form of images (faces) are called ***eigenfaces***.

For plotting the first 20 eigenfaces, I have used pca.components_ to get the eigen vectors and reshaped it according to *the number of comoponents (we used 100 in case of PCA), the height and the width* to get the eigen faces. Next, I have defined a function *grid*, which plots a gallery of the images. Through this function, I have plotted the first 20 eigenfaces (fig. 4)

Here, we can observe that using 100 dimensions, the images look like **'ghost images'**. This is because we are trying to visualise 2914 dimensions/features in 100 dimensions. *The first few eigen vectors represent the maximum information and have the maximum variance. In other words they conserve the most valuable features.* The later eigen vectors represent more specific information and do not account that much variation as the first few eigen vectors do.

To make the analysis more complete, I have also plotted 20 faces and their predicted and true names through PCA (figure 5). As we can see, only 2 out of 20 images were wrongly predicted through PCA.

2.6 Eigenfaces retaining 80% variance, KNN and comparison of results

Next, I have found out the explained variance ratio, which essentially means the ratio of the variance corresponding to each eigenvector and the sum of the variances of all the eigen vectors. This I have done using the following code: *variance = cov_matrix.explained_variance_ratio_* where cov_matrix is the covariance matrix calculated while performing PCA. Then I have calculated the cumulative variance through using the *cumsum* function in *numpy*: *vars=np.cumsum(decimals=3)*100)* This *vars* variable displays an array which tells the **cumulative explained variance ratio** which looks like this: [23.199999, 39.3 , 46.3 , 51.6 , 55.199997,

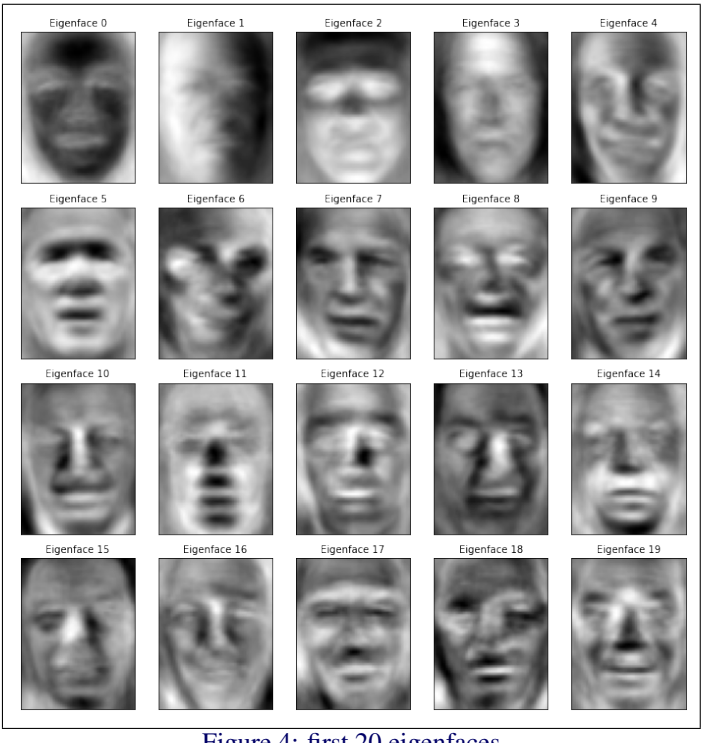


Figure 4: first 20 eigenfaces

Figure 5: 20 Faces predicted vs true values

```
[[ 41   4  13   5   1]
 [  5   7  10   1   0]
 [ 39  15 100   4   8]
 [  5   1  24   8   4]
 [  8   2  30   3   4]]
              precision    recall  f1-score   support

           0       0.42      0.64      0.51        64
           1       0.24      0.30      0.27        23
           2       0.56      0.60      0.58       166
           3       0.38      0.19      0.25        42
           4       0.24      0.09      0.12        47

    accuracy                           0.47       342
   macro avg       0.37      0.36      0.35       342
weighted avg       0.45      0.47      0.44       342
```

Figure 6: Classification report for KNN on 31 eigen vectors

57.999996, 59.999996, 61.799995, 63.499996, 65.1 , 66.5, 67.8, 69., 70., 71., 71.9, 72.8, 73.600006, 74.3, 75., 75.6, 76.2, 76.7 , 77.2, 77.7, 78.2, 78.7, 79.1, 79.5 , 79.9 , 80.3, 80.700005] **This indicates that the first 31 eigenfaces give 80.3% variance, this is the x which is mentioned in the question** Hence we now repeat the nearest neighbor classification procedure with x=31 eigenfaces. The reports obtained are shown in fig. 6.

We can notice from the classification report in this case that the accuracy has dropped to 47%, as expected. When KNN was performed on 100 components, the explained variance was 92% and accuracy with nearest neighbour classification was 67%. In this case, with 31 components, explained variance is 80% and accuracy has reduced to 47%. This clearly shows that reducing the number of components reduces accuracy as many important, useful features, are lost while reducing the dimension. This means that less information is captured by the reduced number of features.

3 Q2: Dimensionality Reduction and Visualization with PCA, LDA and tSNE

This question is based on the Fisher Iris data set. Fisher Iris comprises 150 4-dimensional data samples and they are associated with 3 iris flower species (Iris Setosa, Iris Virginica and IrisVersicolor).The 4 features/dimensions are sepal width, sepal length, petal width,petal length. We first import the libraries, and load the data set from sklearn.datasets. Then, we do data preprocessing on it through using the MinMaxScaler. Through this, we standardise the sample values.

3.1 Using PCA to reduce the dimensionality to 2 and visualization based on characterizing through various features

We then use PCA on the dataset and reduce the number of dimensions from 4 to 2 (so we use n_components=2). Next, we are required to visualise the data based on the features. For this, I have defined a function: *irisplot* which does a scatter plot with the horizontal axis denoting the 1st eigen vector and the vertical axis denoting the 2nd eigen vector. *Then I have divided*

the range of values of the features into 2 categories. These categories are: **above mean value (shown by Red points on the scatter plot) and below the mean value (shown by Blue points on the scatter plot).** The corresponding plots with the range of values divided into these 2 categories and based on each of the 4 different features are explained here:

1. **PCA(2 dimensions)- Characterization based on values of *sepal length***: Shown in figure 7. The red points are the values of sepal length above the mean value, and the blue ones below it. From this scatter plot, we can clearly identify that **along the eigen-direction 1 (horizontal axis), high values correspond to high values of sepal length** and **along the eigen-direction 2 (vertical axis), high values correspond to low values of sepal length.**

2. **PCA(2 dimensions)- Characterization based on values of *sepal width***: See figure 8. The red points are the values of sepal length above the mean value (higher values), and the blue ones below the mean(lower values). From this scatter plot, we can clearly identify that **along the eigen-direction 1, high values correspond to high values of sepal width only in cases where their values along the eigen-direction 2 are also high, but not in cases where their values along eigen vector 2 are low. Along the eigen-direction 2, high values correspond to high values of sepal length.**

3. **PCA(2 dimensions)- Characterization based on values of *petal length***: Shown in figure 9. The red points are the values of sepal length above the mean value, and the blue ones are the lower values. From this scatter plot, we can clearly identify that **along the eigen-direction 1, high values correspond to high values of sepal length** and **along the eigen-direction 2, high values correspond to low values of sepal length.**

4. **PCA(2 dimensions)- Characterization based on values of *sepal length***: Shown in figure 10. The red points are the values of sepal length above the mean value, and the blue ones are below it. From this scatter plot, we can clearly identify that **along the eigen-direction 1, high values correspond to high values of sepal length** and **along the eigen-direction 2, high values correspond to low values of sepal length.**

From the above inferences, it can be understood that, in general, *high values along eigen-direction 1 correspond to high values of sepal length and width and also petal length and width. Also, high values along eigen-direction 2 correspond to low values of petal length and petal width.*

3.2 LDA(Linear Discriminant Analysis)

LDA is used to reduce the dimensions so that we can better characterize or separate two or more classes. It is an unsupervised learning algorithm. It tries to find a linear combination of features that helps us in visualization. It projects C class data to C-1 dimensions. We can use LDA through the inbuilt functionality of sklearn using line of code: *from sklearn.discriminant_analysis import LinearDiscriminantAnalysis*

In this part of the question, we are required to perform LDA pairwise on the 3 classes. This means that we select 2 classes at a time and project the data (through using LDA) to 2-1=1 dimension. This would give a straight line plot. The lines of separation and the threshold are also plotted.

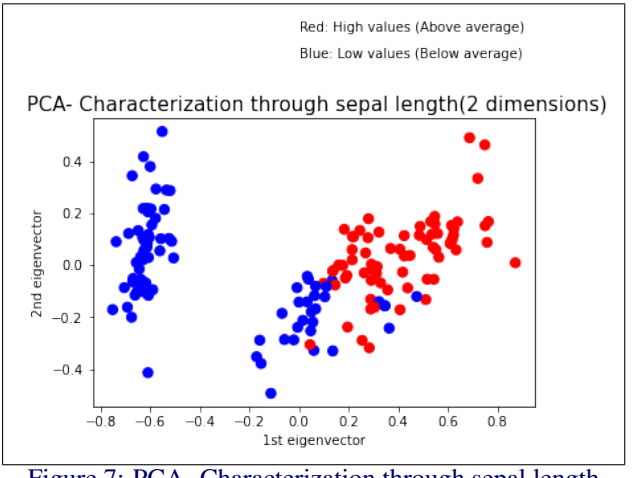


Figure 7: PCA- Characterization through sepal length

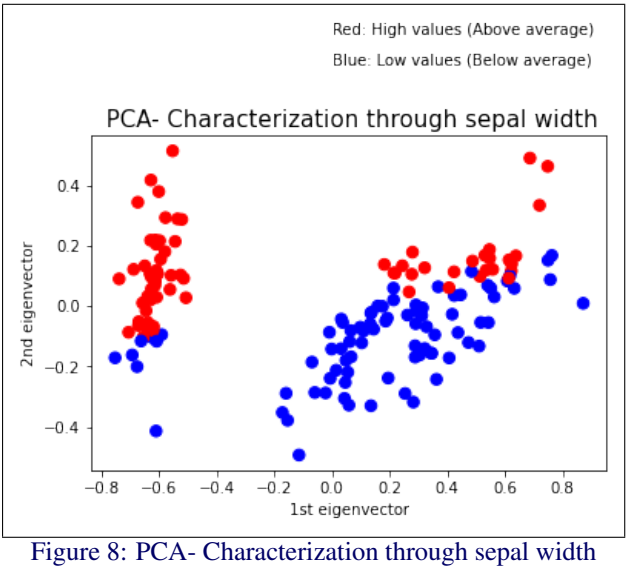


Figure 8: PCA- Characterization through sepal width

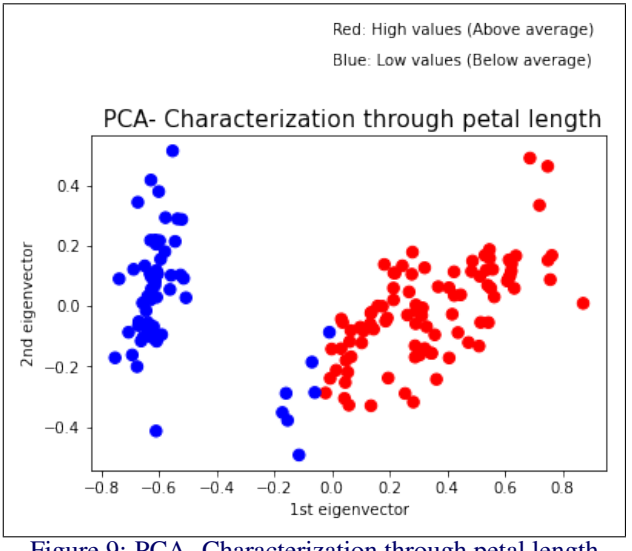


Figure 9: PCA- Characterization through petal length

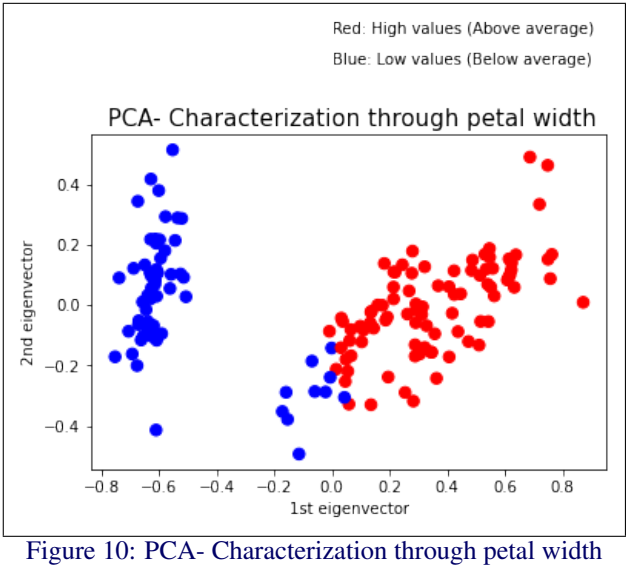


Figure 10: PCA- Characterization through petal width

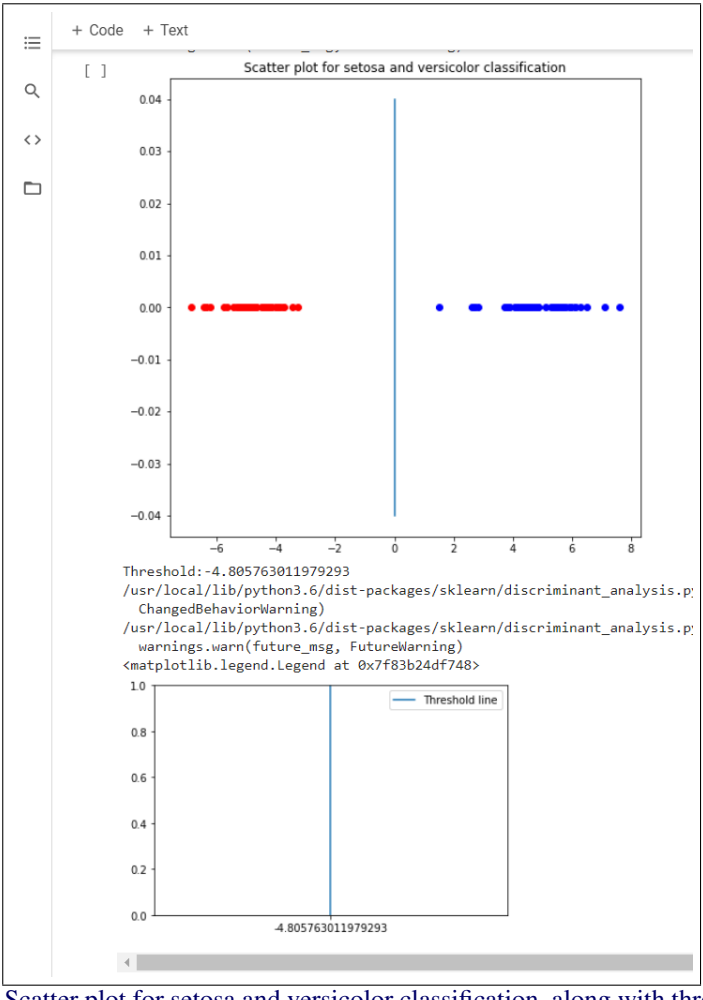


Figure 11: Scatter plot for setosa and versicolor classification, along with threshold line

1. **Scatter plot for setosa and versicolor classification:** Refer figure 11. Here we can see that the separation between classes is maximized through LDA and the line separating them has also been plotted. *The threshold for this classification is -4.805 as calculated through this piece of code: separation(lda.fit(setosa_versicolor, setosa_versicolor_tg .transform(setosa_versicolor), setosa_versicolor_tgts,0,1).*

2. **Scatter plot for versicolor and virginica classification:** Refer figure 12. Here we can see that the separation between classes is maximized through LDA and the line separating them has also been plotted. The threshold is -6.108.

3. **Scatter plot for setosa and virginica classification:** Refer figure 13. The threshold is -1.510.

Next, I have plotted the projected 3-class Fisher Iris data on the 2D feature space. See figure 14. Here, the separating lines have been plotted using SVM. Through this plot (and

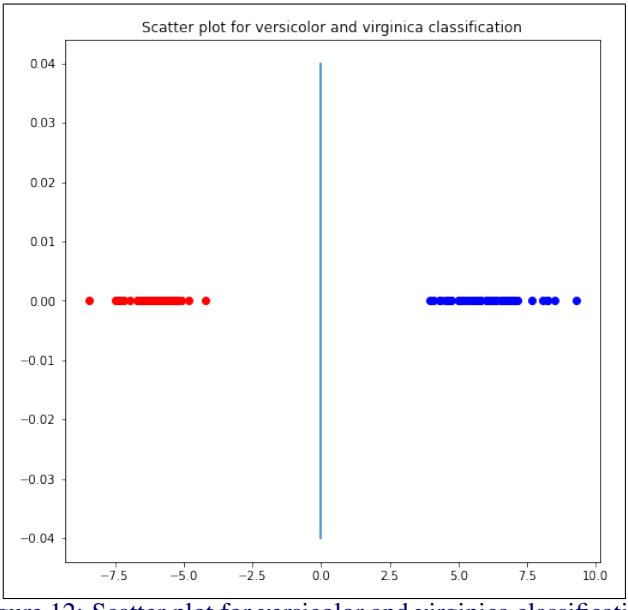


Figure 12: Scatter plot for versicolor and virginica classification

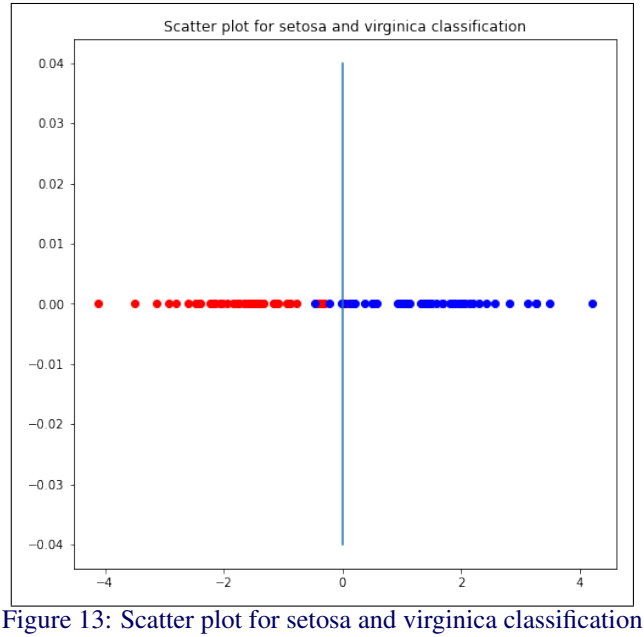


Figure 13: Scatter plot for setosa and virginica classification

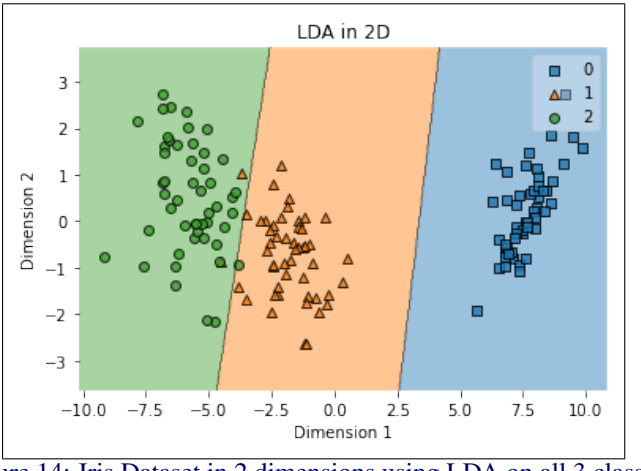


Figure 14: Iris Dataset in 2 dimensions using LDA on all 3 classes

the above 1D plots), we can clearly see that **LDA maximizes inter-class scatter while minimizing intra-class scatter.** In figure 15, I have plotted the same graph again, with a different color scheme.

3.3 2D and 3D Visualization through t-SNE

t-SNE or t-Distributed Stochastic Neighbor Embedding is a technique majorly used for data exploration and visualizing high-dimensional data. It is an unsupervised algorithm like PCA. It tries to calculate a similarity measure between instances in the high dimensional space and the corresponding low dimensional space. These two similarity measures are then optimized using a cost function. It keeps the less similar points away/distant and the similar points close to each other. It can be used through the *manifold* package in *sklearn* library of Python.

In this part of the question, we are required to project the 4-D Fisher Iris data onto 2 Dimensions and 3 Dimensions via tSNE. In tsne, we use a metric, which relates to the method of computing the distance between instances in a feature array. Some examples of metrics are: Euclidean, Cosine, Chebyshev, Minkowski, etc. The 2 metric parameters used here are: 'Euclidean' and 'Cosine'.

1. **Iris Dataset in 2 dimensions through t-SNE (metric= Euclidean):** Figure 16
2. **Iris Dataset in 2 dimensions through t-SNE (metric= Cosine):** Figure 17
3. **Iris Dataset in 3 dimensions through t-SNE (metric= Euclidean):** Figure 18
4. **Iris Dataset in 3 dimensions through t-SNE (metric= Cosine):** Figure 19

2D: From the above plots, we can observe that t-SNE has separated the 2 dimensions far apart (it punishes dissimilarities badly by keeping them distant). t-SNE hence helps visualise high dimensional datasets in an easily comprehensible and visually understandable manner. The clusters are easily visible in most metrics (I tested with Euclidean, Cosine, Chebyshev, Hamming), except hamming where the clusters are quite close to each other. The Cosine

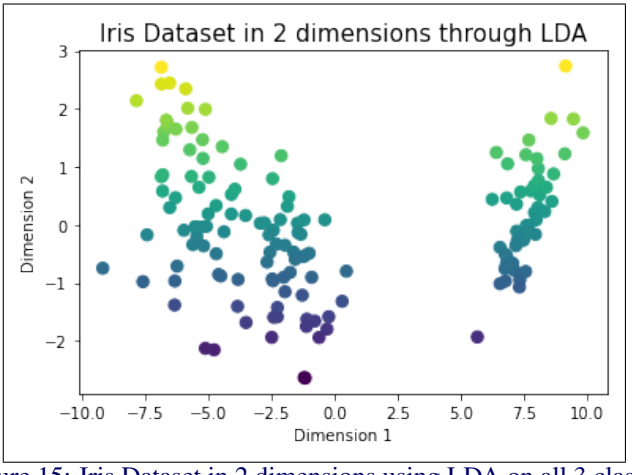


Figure 15: Iris Dataset in 2 dimensions using LDA on all 3 classes

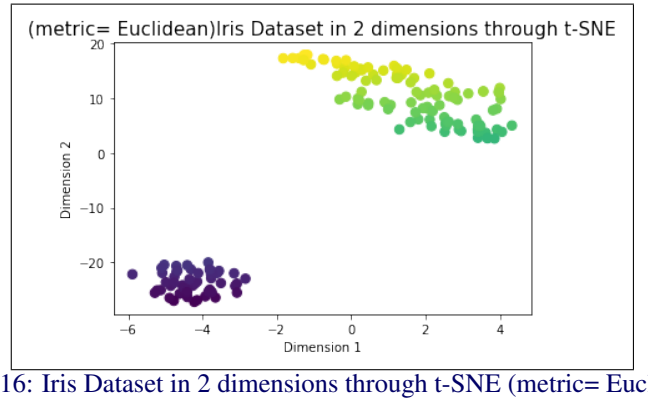


Figure 16: Iris Dataset in 2 dimensions through t-SNE (metric= Euclidean)

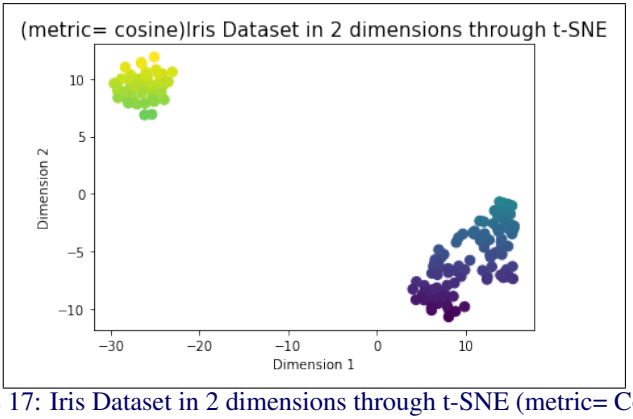


Figure 17: Iris Dataset in 2 dimensions through t-SNE (metric= Cosine)

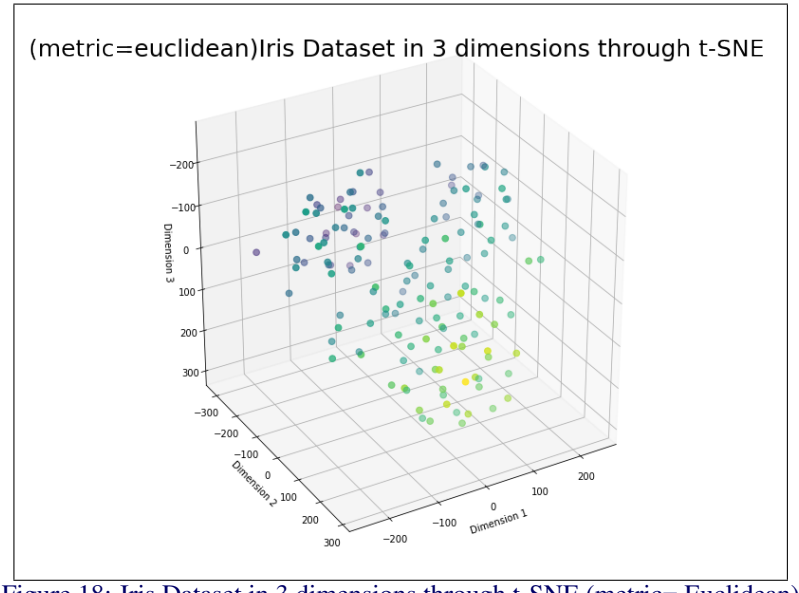


Figure 18: Iris Dataset in 3 dimensions through t-SNE (metric= Euclidean)

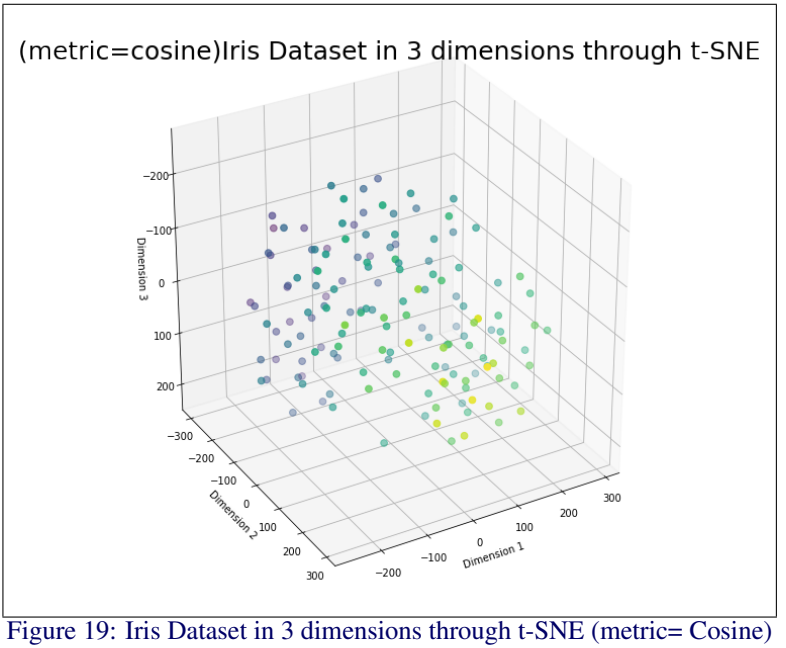


Figure 19: Iris Dataset in 3 dimensions through t-SNE (metric= Cosine)

metric has clustered the points most accurately and the clusters formed are very small and far apart.

We must also notice the scale of the plots. In Cosine, the scale of the axis is smaller as compared to Euclidean. This shows that the Euclidean might just be more effective in separating out very high dimension data where features are very close to each other. In this plot, the darker points (dark blue in colour) represent the lower class: 0 and the lightest (yellow) represent the higher class: Class 2. Light blue represents class 1.

3D: In 3D, in both Euclidean and Cosine, the darker points are shown on the top (higher dimension 3) and the lighter ones at the bottom (lower dimension 3). Although the pattern is not very clearly visible in this view, but it might be better visible if seen from another view. The pattern is there nevertheless, which can be noticed if we look closely. Along higher dimension 1, more lighter spots are seen , along increasing dimension 3, darker spots are more prominent and on the side of higher dimension 2, lighter points are more prominent.

4 Q3: Data Classification with Linear and Non-linear SVMs

In this question, we are required to consider either of the following feature pairs: petal length and petal width or sepal length and sepal width. I have chosen ***petal length (cm) and petal width (cm)***. After importing the libraries, I have imported the data set and created a data frame out of it. From this data frame, I have taken X to be the feature values corresponding to petal length and petal width only.

4.1 Using SVM to plot the max-margin hyperplane and a plot of the two class-data (taken pairwise from the 3 classes)

Support Vector Machine (SVM) is a supervised learning technique which helps to classify data into 2 classes. They can be used for various tasks (like regression, outlier detection) and are widely used in the industry as well. In SVMs, a hyperplane is constructed which separates two classes. The hyperplane that is chosen has maximises the distance between the nearest data points of 2 classes. The larger the margin, the lower the generalization error of the model. Support Vectors are the vector passing through the nearest points.

I have plotted the dataset using SVM on all 3 classes and 2 features (petal length and petal width) in figure 20. Note: I have used a linear kernel here.

In the question, we are required to plot the max-margin hyperplane for the three pairs of classes (see figure 21,22,23).

In these 3 figures, the dashed lines represent the margins and the bold line represents the chosen hyperplane. The highlighted points represent the Support Vectors. **The corresponding classification reports are also shown in the figures.**

4.2 Classification reports for C values of 0.001, 1 and 1000

The C parameter is very significant while implementing the SVM classifier function. It hardens or softens the boundaries. Accordingly the results get affected and the model performs better/worse. By hardening I mean that the boundaries are hard and if a point does not lie in

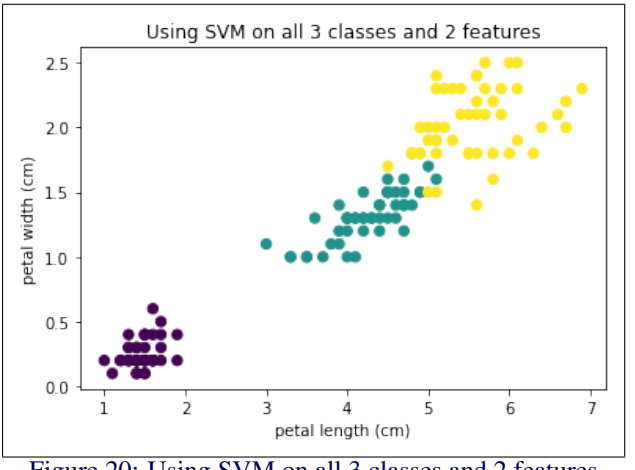


Figure 20: Using SVM on all 3 classes and 2 features

	precision	recall	f1-score	support
1	0.86	0.86	0.86	14
2	0.88	0.88	0.88	16
accuracy			0.87	30
macro avg	0.87	0.87	0.87	30
weighted avg	0.87	0.87	0.87	30

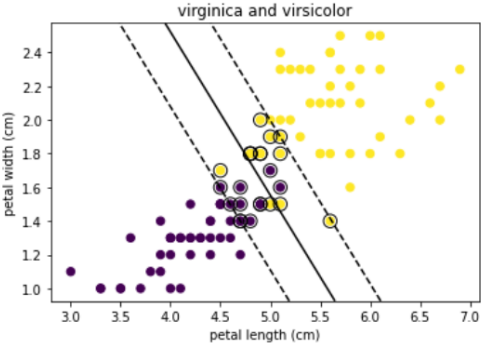


Figure 21: Linearly separating virginica and virsicolor

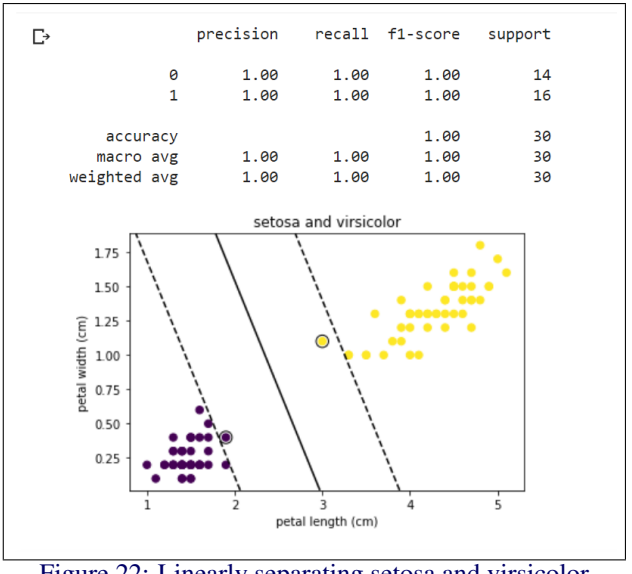


Figure 22: Linearly separating setosa and virsicolor

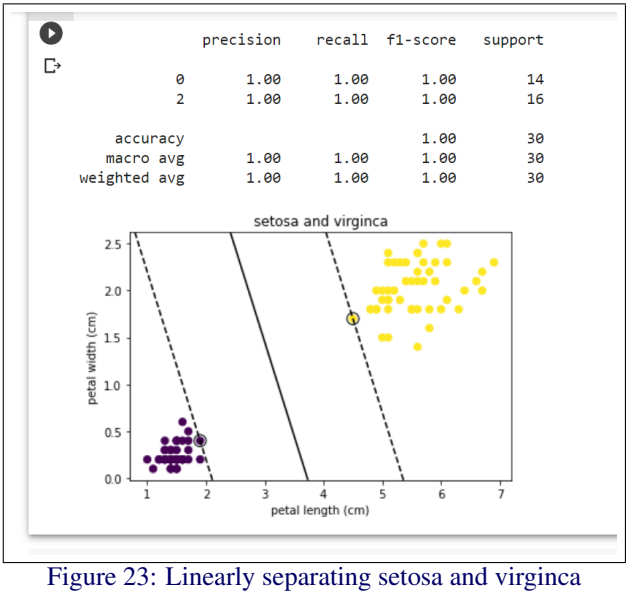
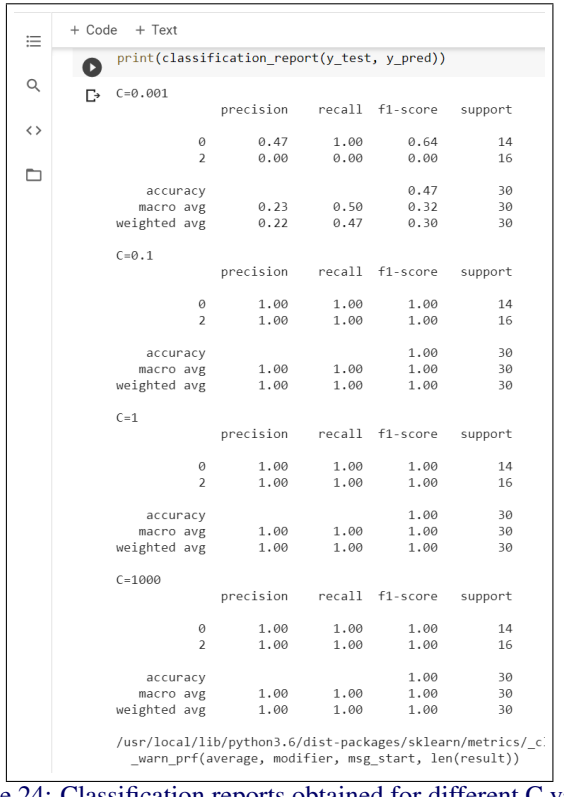


Figure 23: Linearly separating setosa and virginca



```
+ Code  + Text

print(classification_report(y_test, y_pred))

C=0.001
              precision    recall  f1-score   support

           0       0.47      1.00      0.64        14
           2       0.00      0.00      0.00        16

    accuracy                           0.47        30
   macro avg       0.23      0.50      0.32        30
weighted avg       0.22      0.47      0.30        30

C=0.1
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           2       1.00      1.00      1.00        16

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

C=1
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           2       1.00      1.00      1.00        16

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

C=1000
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           2       1.00      1.00      1.00        16

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_c
  _warn_prf(average, modifier, msg_start, len(result))
```

Figure 24: Classification reports obtained for different C values

the boundary it is not included. In case of a soft margin, the margin size can grow to include more points.

This is evident from the results as well (classification reports in *figure 24*). The harder boundary (C=1000), have better performance as it does not allow the data points of the other class to be included (perfect accuracy: 100%. This perfect accuracy in this case can be attributed to the small size of the data set: i.e. less number of samples). While the C=0.01 shows somewhat relaxed margins and hence the performance is worse (less accuracy: 47%).

In general, a large C value gives better results. But after a point, overfiitting starts and the model does not predict well on new, unseen points. From the classification report, it is evident that overfitting is attained well before C=1000. In fact, for C=1 and 0.1 also, the results are the same (100% accuracy).

4.3 Employing an RBF

In the previous parts, I had used the 'linear' kernel in the SVM. Here we are required to repeat the above parts for the RBF function. RBF (Radial Basis Function) accounts for the non-linear separation by adding an extra dimension. In general, it gives better results as the boundaries are more complex and capture more information as compared to simple linear boundaries. As a result, the accuracy might increase as well.

Inspite of its better/more complex classification, it is likely to overfit faster, as it has some

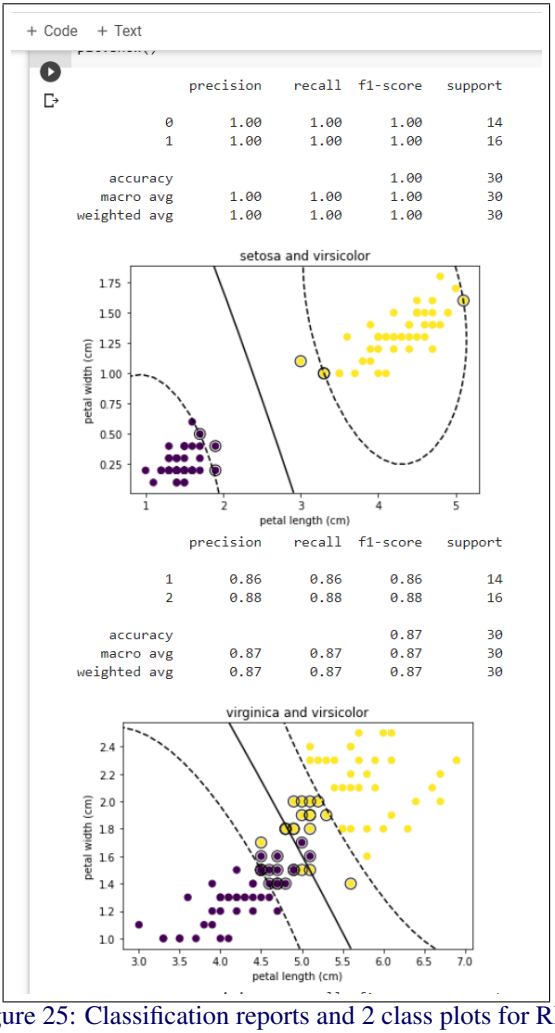
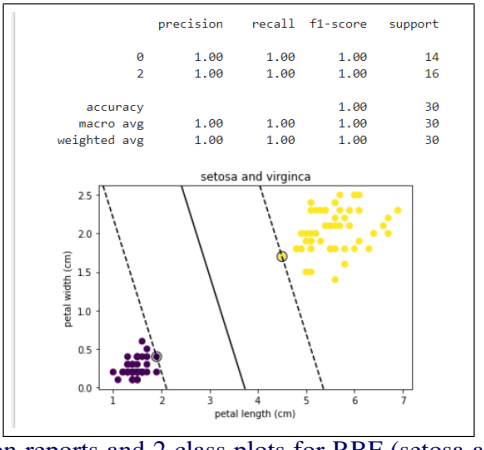


Figure 25: Classification reports and 2 class plots for RBF

complex boundaries also involved (unlike simple linear boundaries). So for an increasing C value, overfitting is likely to be achieved earlier (at a lower C) for RBF as compared to Linear kernel in SVM.

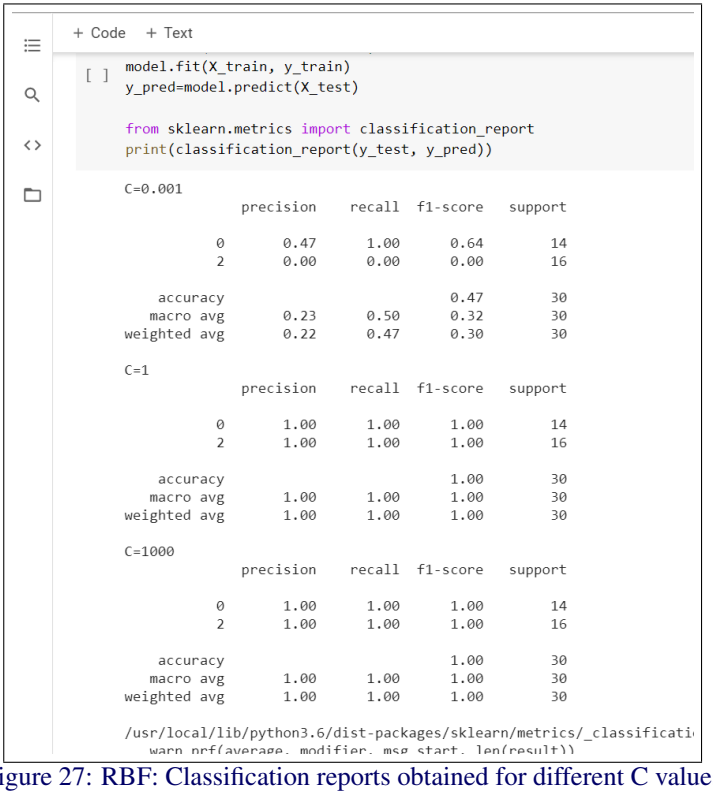
Some mis-classifications/ boundary cases in the previous part (using linear kernel) can now be classified correctly through RBF. For example, in figure 21, we can see the versicolor and virginica classification has lots of points on the boundary and some mis-classifications as well. These can be corrected by using a non-linear boundary, which we are using through RBF. These RBF plots and the corresponding classification reports are shown in *fig. 25, 26*.

Finally the classification reports corresponding to different C values have been shown in *figure 27*. In general, linear kernels are used more widely because RBF is more expensive both in terms of training and prediction (in case of large data sets).



	precision	recall	f1-score	support
0	1.00	1.00	1.00	14
2	1.00	1.00	1.00	16
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Figure 26: Classification reports and 2 class plots for RBF (setosa and virginica classification)



```
+ Code  + Text

[ ]  model.fit(X_train, y_train)
     y_pred=model.predict(X_test)

     from sklearn.metrics import classification_report
     print(classification_report(y_test, y_pred))
```

C=0.001

	precision	recall	f1-score	support
0	0.47	1.00	0.64	14
2	0.00	0.00	0.00	16
accuracy			0.47	30
macro avg	0.23	0.50	0.32	30
weighted avg	0.22	0.47	0.30	30

C=1

	precision	recall	f1-score	support
0	1.00	1.00	1.00	14
2	1.00	1.00	1.00	16
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

C=1000

	precision	recall	f1-score	support
0	1.00	1.00	1.00	14
2	1.00	1.00	1.00	16
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

```
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classificati
  warn_prf(average, modifier, msg_start, len(result))
```

Figure 27: RBF: Classification reports obtained for different C values

5 Conclusion

This assignment showed us the power of dimensionality reduction tools and SVMs. I am now able to better explore, understand and visualise large data sets. In particular it helped me understand the implementation of PCA, LDA, t-SNE and SVMs (linear and non-linear).

- The important concept of eigen-vectors and eigen-faces can really help a data scientist or any engineer to handle data in a more effective, and useful manner.
- The power of PCA can help to reduce data sets of high dimensions to very less dimensions while preserving the most relevant and useful information. This we saw through the "Labelled faces in the Wild" data set.
- t-SNE is another dimensionality reduction technique which is primarily used for visualisation and expolaration. In question 1, we reduced 2914 dimensions to 100 via PCA and 100 to 2 dimensions via t-SNE.
- A KNN (K-nearest neighbours classifier) was also used. It is a supervised technique which helps to assign the closest and most common values to the object/data point.
- LDA is another dimensionality reduction approach which reduces the dimension by 1 and helps us visualize the data better.
- The Support Vector Classifier of the SVM package helps to classify 2 classes based on the separating hyperplane. The separating hyperplane can be linear or non-linear (in case of RBF) depending on the kernel function we use.
- Finally, we learnt a lot of visualisation techniques, apart from the ones above. I learnt how to plot eigenfaces through defining a grid/gallery function.

This assignment gave us an exposure to the numerous powerful data visualisation, classification and dimensionality reduction tools that are out there. The assignment has equipped us with tools that are very important and useful for working on data sets in a more effective manner.

6 References

1. t-SNE Documentation:
   https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html
2. Towards Data Science (Medium):
   https://towardsdatascience.com/manifold-learning-t-sne-lle-isomap-
3. Geeks for Geeks:
   https://www.geeksforgeeks.org/python-numpy/
4. KNN Stack Abuse:
   https://stackabuse.com/k-nearest-neighbors-algorithm-in-python-and
5. Scikit-learn plot eigen faces:
   https://scipy-lectures.org/packages/scikit-learn/auto_examples/plot_eigenfaces.html

6. Stack Exchange
7. Documentation of Python Scikit Learn library
8. Scikit-learn PCA vs LDA:
https://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_vs_lda.html
9. PCA on Towards Data Science:
https://towardsdatascience.com/a-step-by-step-introduction-to-pca-c
10. t-SNE metrics:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.pdist.html
11. Wikipedia: https://en.wikipedia.org/wiki/Eigenface
12. Plotting separating hyperplanes:
https://scikit-learn.org/stable/auto_examples/svm/plot_separating_hyperplane.html
13. SVM Wikipedia: https://en.wikipedia.org/wiki/Support_vector_machine