

Assignment 3- PCA, LDA, tSNE and SVMs

Ankur Khosla ¹
2018meb1211

¹IIT Ropar
Rupnagar, Punjab

Abstract

This assignment was focused on the dimensionality reduction techniques. PCA, LDA and tSNE were not only deployed, but also visualized, through examples. A comparative analysis was also drawn out between the three techniques. In the first part, the concept of eigenfaces was also used. By the end of the assignment, linear and non-linear SVMs were used and data classification was performed. The performance metrics were used to analyze the overall efficiency of each technique. Also, for each part, theoretical expectations and practical outcomes have been compared, hence, providing a complete picture of each model.

1 PCA and eigenfaces

PCA stands for Principal Component Analysis. It was a technique discovered in mid 1900s and has been used ever since. PCA is one of the most basic algorithms to reduce the dimensions of the data. PCA is performed on unsupervised data. The main idea behind this algorithm was to identify the important features and eliminate the less important ones. Obviously, there is some loss of data, when we eliminate less important features, but the total loss in accuracy versus the number of features dropped are very less. The complexity of the model reduces significantly. For example, a model is exhibiting accuracy of 0.87 using 1000 features, and through PCA, we achieve an accuracy of 0.82 using 100 features. Hence, the trade-off between the accuracy of the model and reduction in complexity is quite logical.

Our primary target in PCA is to reduce the number of features. For this purpose, we had to define a parameter, that signifies the importance of a feature. The 'feature' used in PCA is variance. It is logical to assume that the direction, in which the variance of data is more, contains more information, as compared to the direction in which variance is less. Hence, we define completely new reference direction vectors, and sort these vectors priority-wise. Using simple mathematical equations, we can derive that the directions, in which variance is maximum is given by eigen vectors, and eigen values, associated with a particular eigen vector, denotes the variance of data in that direction.

Through PCA, we retain major part of the important information. During the dimensionality reduction through PCA, we can keep a cap on the numbers of dimensions, or on the total variance, we want to include in our model. In the further sub-sections, we will explore the various possibilities in PCA.

In the first part of our assignment, we will use the concept of eigenfaces. The ideology behind the eigenfaces is almost similar to that of PCA. A 320 X 240-pixel image could be read as 76,800-dimensional vector and PCA could be applied on the generated data-set

from the pixel values. Finally, after dimensionality reduction, we will generate new image with the retained data, and loss of information could be analyzed from the 'ghost-images'.

1.1 Applying PCA

Prior to using any model to a dataset, it is very important to understand the nature of data. In this assignment, we are working with images. Each image can be visualized as matrix of pixels, where value of each matrix cell denotes the brightness of that corresponding pixel. An image with x rows and y columns, would have $x*y$ different values in the matrix, and hence, number of input features will be $x*y$.

The dataset of the images was imported through `sklearn.datasets` library and 'fetch_lfw_people' command. To understand the organization of imported dataset, following values were used.

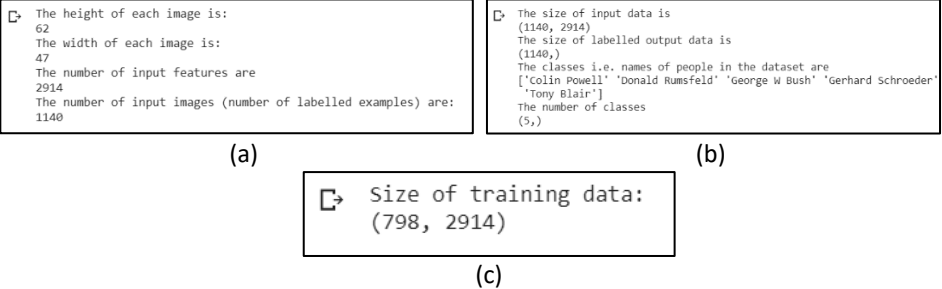


Figure 1: (a) size of each image and number of total images (b) Labelled input and output matrices and number of classes (c) training data after `train_test_split` command

Total of 1140 images were imported, with pixel-size of each image as 62 X 47. Hence, the number of input features were $62*47 = 2914$. In the figure 1(b), we can observe the input matrix of size (1140, 2914). The number of classes corresponds to the faces of 5 people. Hence, all the labelled outputs are in the range [1, 5]. 1 denoting Collin Powell, 2 denoting face of Donald Rumsfeld and so on.

After splitting test and training data in ratio 30:70, the number of labelled samples changed in the input matrix (70% of 1140 = 798), however, the input features remained same i.e. 2914. Through PCA, our main target is to reduce the number of input features from 2914 to 100.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=100)
pca.fit(x_train)
x_train_new = pca.transform(x_train)
x_test_new = pca.transform(x_test)

print("The size of input data after applying PCA:")
print(x_train_new.shape)
#The number of features must have reduced to 100
```

The size of input data after applying PCA:
(798, 100)

Figure 2: Applying PCA on the input matrix

As shown in the figure 2, we reduced the number of input features to 100. Maximum 100 eigen values of covariance matrix (after normalization), were selected and corresponding eigen vectors were arranged in columns.

1.2 Projection on eigenface space

In the previous section, we retained the values of eigen vectors, through command `PCA.fit()`. `PCA.transform()` command projects our initial data on the eigen space. As depicted in figure 2, `x_train_new` was our projected data. Hence, we have converted each image into a 100-dimensional vector.

For the next part, we will use this projected vector, and compute two dimensions via tSNE. tSNE is another dimensionality reduction technique, used for visualization purpose. Through tSNE, we further squeeze 100 dimensions into two, and plot a scatter plot, and observe if any clusters are formed. Since, tSNE is a stochastic (random) process, it can't be used on new test data, but could be used for analyzing the existing data. tSNE will simply analyze the distance between two points and compute the projection of distance on a gaussian distribution, (T-distribution in low dimension space). After normalizing the corresponding pdf values, we could observe if any clusters are formed.

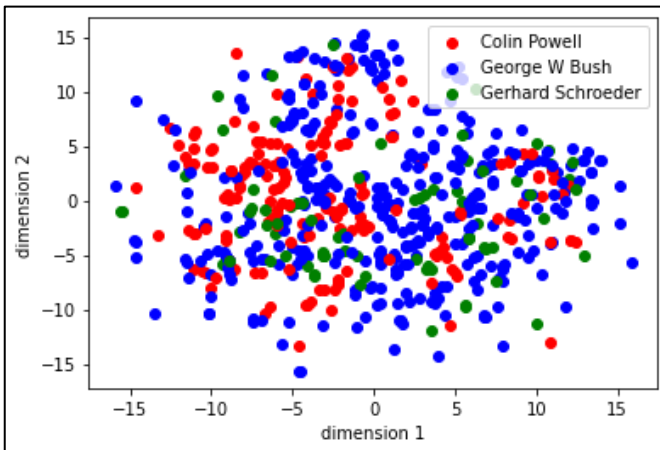


Figure 3: 2-D tSNE plot of projected vectors on eigenspace

In the above graph, we can clearly observe that there is no major cluster formation. We reduced the dimensions and plotted the points corresponding to Powell, Bush and Schroeder. As we know, tSNE is a non-linear unsupervised algorithm. The stochastic perplexity parameter was kept 40 in the above example. tSNE works well, when the data is well-defined in clusters. Since the above plot depicts that data is uniformly distributed over the eigenspace. No well-defined centers could be observed, and hence, we could conclude that tSNE isn't a good algorithm for visualization of our data. The distribution of pixel features over the eigenspace is almost similar in all three cases.

1.3 Nearest Neighbor Classifier

K-nearest neighbor classifier is an example of supervised machine learning. It works well with labelled data, and in our case, the data had 5 classes. Each class denoted different person's face, as shown in figure 1(b). KNN is a discriminative model (DM), and maps test data to provided labelled data. It could also be considered as conditional or backward model, since it maps each new set of features to pre-existing features.

In this assignment, KNN was deployed on the labelled data generated after performing PCA. Hence, for each example, 100 features were given as input. The test data was reduced to 100 dimensions before feeding to KNN. The predicted output was compared to labelled output, and following classification report was generated.

↗	precision	recall	f1-score	support
0	0.56	0.80	0.66	61
1	0.52	0.31	0.39	39
2	0.67	0.85	0.75	158
3	0.29	0.05	0.09	37
4	0.71	0.32	0.44	47
accuracy			0.62	342
macro avg	0.55	0.47	0.46	342
weighted avg	0.60	0.62	0.58	342

Figure 4: KNN – Classification report

The leftmost values - 0, 1...4 denotes the classes. We can see the accuracy of this model is 0.62 or 62% of the predicted values are correct. Precision is ratio of relevant to retrieved output, whereas, recall is ratio of retrieved to relevant output. We can observe that precision value of class 4 (Tony Blair) is quite good, but simultaneously, recall value is bad. In case of class 0 (Colin Powell), the scenario is completely opposite. Precision-recall is balanced only in class 2 (George W. Bush). These statistics also depend on the quality of input images provided. If the initial images are blurred and ghost images produced after dimensionality reduction results in further loss of information, and hence, performance of KNN-algorithm degrades significantly.

1.4 Eigenfaces

Initially, image with (62 X 47) dimensional data was provided to us. We reduced the number of features to 100-dimensions. When we again plot this 100-dimensional image, we generate ghost faces. There is slight blurriness in the generated images, denoting the loss of information during dimensional reduction. Although, the newly generated images could be identified, as the major chunk of data was preserved, when we selected the directions of highest variances through eigen-values.

The following figure depicts the eigenfaces. These images were generated after projecting original vectors to the eigenspaces.



Figure 5: First 20 eigenfaces

1.5 Eigenfaces and data variance

In the previous sections, we pre-defined the number of eigen vectors (100 in our case), i.e. number of directions, we want under consideration. Apart from the number of directions of highest variances, we could also keep a cap on the net variance we want to include in our model. The eigen-values gives the variance associated with a particular eigenvector. If we include n eigenvectors and total eigenvectors are k, then the ratio of summation of n eigenvalues to k eigenvalues will give the percentage of total variance, we have taken into account. For example, we want to include 0.82 part of the total variance, we will keep adding eigenvectors (directions) in the reduced dimension model, until the ratio of eigenvalues included to the sum of total eigen values becomes equal or greater than 0.82.

```
pca.explained_variance_ratio_.sum()
```

0.9235462

Figure 6: Ratio of variance included in case of 100 eigenvectors

As depicted in the above figure, when we included 100 eigenvectors i.e. 100 directions of maximum variances, we included 92% of total variance, during dimensionality reduction.

pca.explained_variance_ratio_ provides an array of n X 1, where n is the number of eigenvectors (100 in above case). We could use this array to compute ith eigenvector, where variance ratio crosses the mark of 0.80.

```

ans = 0
for f in range(0,100):
    pca3 = PCA(n_components=f)
    pca3.fit(x)
    if pca3.explained_variance_ratio_.sum() >= 0.8:
        ans = f
        break

ans
32

```

Figure 7: number of eigenvectors for variance ratio 0.80

Hence, when **32** eigenvectors are used, we retained 80% of the original total variance. When we applied nearest neighbor classifier, classification report generated was as follows:

	precision	recall	f1-score	support
0	0.51	0.71	0.60	70
1	0.44	0.30	0.36	40
2	0.67	0.81	0.73	159
3	0.11	0.04	0.06	27
4	0.47	0.17	0.25	46
accuracy			0.58	342
macro avg	0.44	0.41	0.40	342
weighted avg	0.54	0.58	0.54	342

Figure 8: KNN - classification report for 32 eigen vectors

We reduced the number of eigen vectors from 100 to 32. There was loss of some information, and hence, it was natural to witness some reduction in accuracy. Accuracy in this case fell down to 0.58. Apart from overall accuracy, there is significant decrease in the precision and recall vales of all the classes. The performance of model has degraded, but if we compare the trade-off, with 100 directions, accuracy was 0.62. There is 4% decrease in accuracy on dropping 68 directions (or eigenvectors). This is because we preserved directions with large variances.

2 PCA, LDA and tSNE

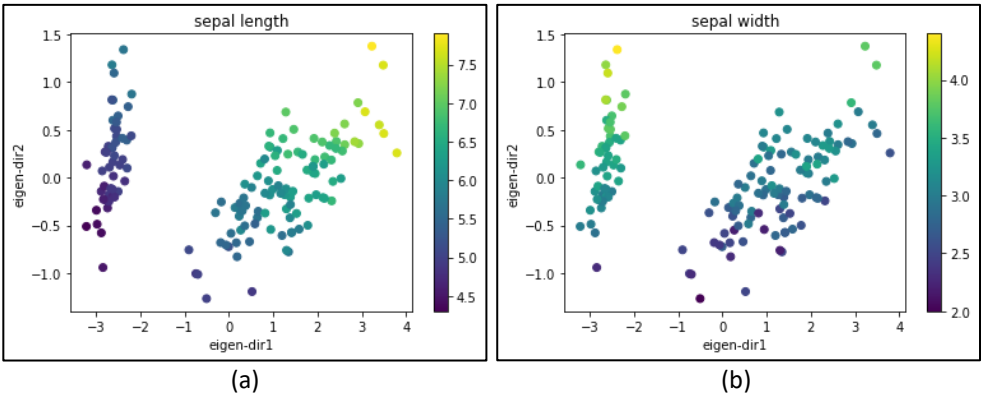
In this section, we will apply PCA, LDA and tSNE techniques and perform dimensionality reduction on Fisher – Iris Dataset. We have observed PCA and tSNE in previous section. LDA stands for Linear Discriminant Analysis. In LDA, we focus on preserving class specific technique as much as possible. PCA is purely mathematical based technique, and hence could be considered on unsupervised technique. LDA gives priority to the class information, and hence, it comes under supervised dimensionality reduction technique. LDA takes into account, the variability within class, and tries to maximize the distance between means of two different classes through projection. We use within class scatter and between class scatter to reduce the number of dimensions of data.

In PCA, we used eigenspace corresponding to covariance matrix of input data. But in LDA, we use the eigenspace associated with $S_w^{-1}S_B$. (S_w is within class scatter, S_B is between class scatter).

2.1 Dimensionality Reduction - PCA

We use fisher-iris dataset in this section. Fisher Iris dataset comprises of 4 features i.e. sepal length, sepal width, petal length and petal width. Corresponding to all the observations, class 0, 1 or 2 is marked. These represents flowers of Setosa, Versicolor and Virginica respectively. Here we will use PCA to reduce number of features to two.

After determining two different eigenvectors, we will plot previous 4 features separately, and each plot will have eigen-direction-1 on x-axis and eigen-direction-2 on y-axis.



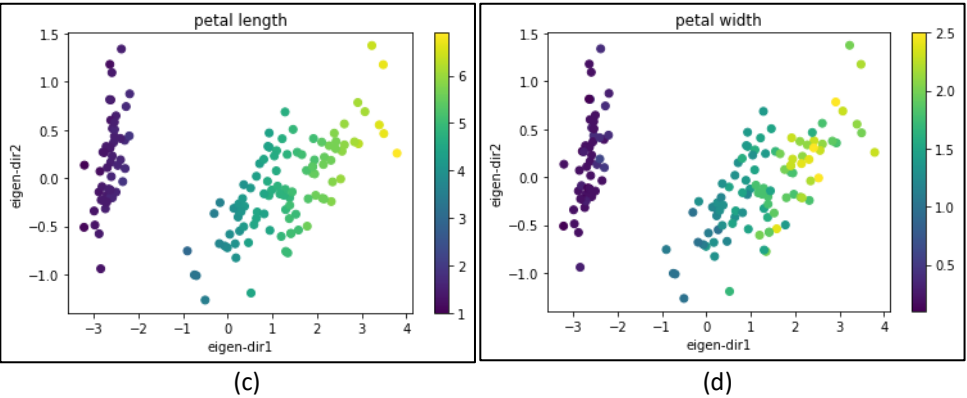


Figure 9: Variation of eigen direction 1 (x) and eigen direction 2 (y) versus (a) sepal length (b) sepal width (c) petal length (d) petal width. Colorbar denotes the values of 4 attributes

The graphs in figure 9 represents the variation of original four attributes versus new eigen directions. The observations, which we could deduce from the above graphs are:

- (a) Sepal length increases with eigen direction 1
- (b) Sepal width increases with eigen direction 2
- (c) Petal length increases with eigen direction 1
- (d) Petal width increases with eigen direction 1

Interestingly, three out of four features vary with eigen direction-1 (direction with highest eigenvalue, and hence highest variance). Clusters can also be observed in the graph of eigen direction 1 versus eigen direction 2. We can also plot four features independently against two eigen directions, but this graph gives better visualization. The change of single feature could be analyzed in both the direction simultaneously.

2.2 Dimensionality Reduction - LDA

LDA is a very common dimensionality reduction technique. It reduces data corresponding to C classes to C-1 dimensions. LDA finds a hyperplane such that distance between means of two clusters is maximized and spread within the class (within class scatter) is minimized.

Fisher Iris dataset has 3 classes of flowers, and it could be easily be projected on a 2-D plane. Although, intra-class scatter is reduced, LDA maximizes inter-class scatter. It attempts to find a hyperplane, such that projection of labelled clusters on that hyperplane produces visible clusters.

The following figure shows the projection of fisher – Iris dataset, on 2-D hyperplane.

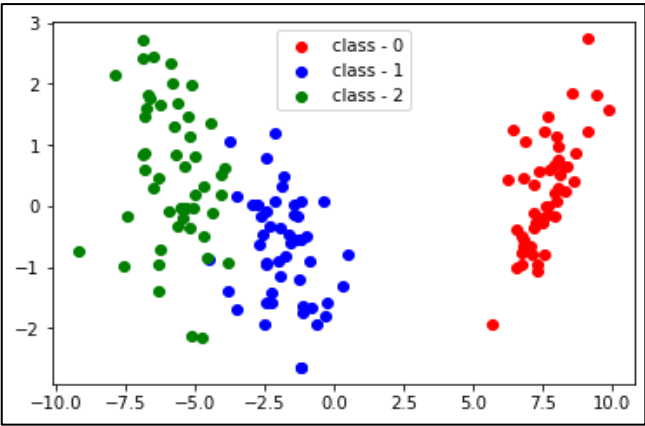


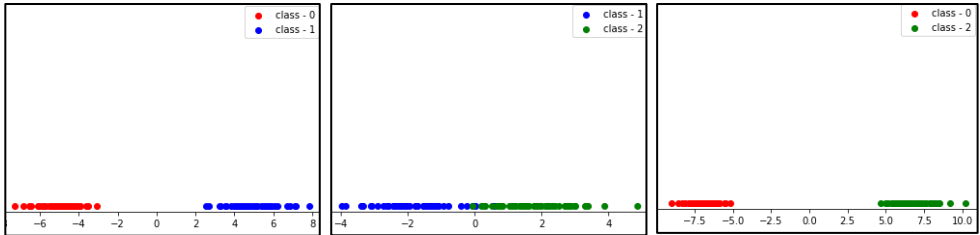
Figure 10: LDA on fisher-iris dataset

The above figure is projection of 3 classes on 2-D axes. We can clearly observe three separate clusters corresponding to class – 0, 1 and 2. However, there is some overlap between class 1 and class 2 clusters.

Further, this data is projected on 1-D axis. For that part, the number of classes has to be two. Hence, we have ${}^3C_2 = 3$ combinations possible for 3 classes. The three cases possible are:

- (a) Class 0 and Class 1 – Setosa and Versicolor
- (b) Class 1 and Class 2 – Versicolor and Virginica
- (c) Class 0 and Class 3 – Setosa and Virginica

Each of the above case was handled separately. In first case, only class 0 and class 1 were assumed to exist on figure 10. A hyperplane was plotted and projections were plotted on that hyperplane, hence producing a 1-Dimension data. Similarly, the process was repeated for second and third case, and graphs obtained were as follows:



(a) (b) (c)
Figure 11: LDA on fisher-iris dataset, considering 2 classes at a time

From figure 11, we could conclude that LDA performed well in every case, in which class – 0 was involved. Interestingly, in figure 10, it was class 0 only, which had completely separate cluster from the rest two. Hence, we produced projections on 2-D hyperplane (figure 10) and 1-D hyperplane (figure 11).

2.3 Dimensionality Reduction - tSNE

tSNE is a non-linear unsupervised dimension reduction algorithm. This method is preferred for visualizing the data. Although, it cannot be used on new data, but it is a powerful tool, to analyze the clusters present in data, if any. As explained in previous section, tSNE works well when mean of different clusters are considerably far apart. From figure 10, we could predict that tSNE would work perfectly for class 0, however, there could be some issues with class 1 and 2, due to their overlap.

Fisher iris dataset had 4 features. We reduced these features into 3 and then 2. New originated features were then plotted against each other, after original data was mapped on t-distribution (normalized). The 2D and 3D graphs were generated as follows:

2.3.1 tSNE – 3D

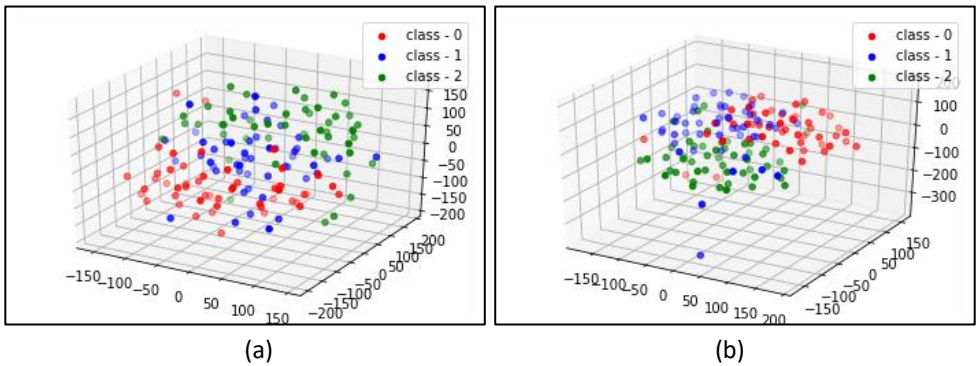


Figure 12: tSNE 3D plots (a) metric = ‘cityblock’ (b) metric = ‘correlation’

The above two plots represent dimensions of dataset reduced to 3 from 4. Comparing the two metric parameters, correlation performs much better. We could easily identify different clusters in 12(b).

2.3.2 tSNE – 2D

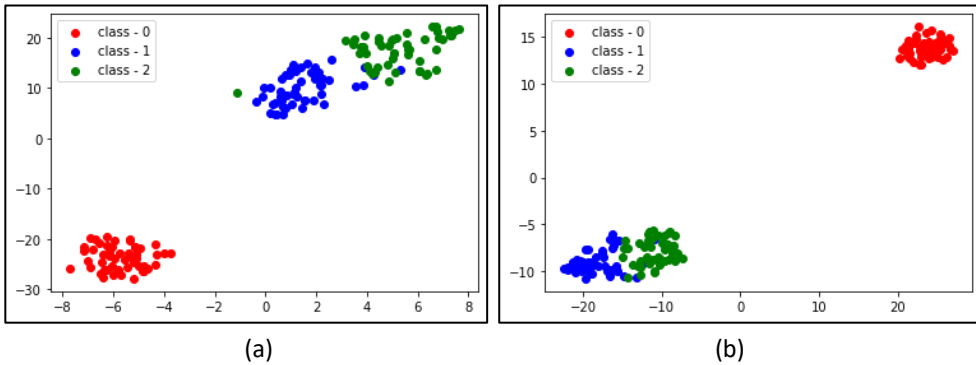


Figure 13: tSNE 2D plots (a) metric = ‘cityblock’ (b) metric = ‘correlation’

Both the metrics performed well in this case. There are well defined three clusters clearly visible. Overall, tSNE performed better in 2D case. In ‘cityblock’ metric, mean of the clusters are sparser, however, clusters are more closely packed in ‘correlation’. Overall, the plots of tSNE have justified their part well, and we could visualize three classes of data easily.

3 SVMs

SVM or Support Vector Machines, are Discriminatory Classifier Model. It works on the principle of figuring out the most optimal decision boundary, which separates the different classes. Hence, this algorithm comes under supervised machine learning. SVMs are the maximum margin estimators, i.e. there might be infinite hyperplanes or decision boundaries, but SVM will prefer the decision boundary with maximum margin.

Usually, hyperplanes are estimated by projecting low dimension data into high dimension feature space. For example, if initial feature space has m features, the projected feature space will have $O(m^p)$ dimensions. Further, to avoid overfitting in such high dimensional feature space, we need $k.O(m^p)$ examples. Practically, this process is not feasible. SVM solves both these problems at once. It learns the optimal hyperplane by maximizing class separation. The training complexity of SVM is directly proportional to number of examples in the input data.

Besides the hyperplanes, SVMs also include the concept of margin. The hyperplane built via SVM is a ‘thick’ hyperplane, and comes with a margin associated with it. For an optimal hyperplane, there should be no overlap of the margin area and any training example. This margin is the uncertain region. For example, if a test example falls in the margin region of class-1, we can claim that it belongs to class-1, but there would be some uncertainty associated with it.

In this assignment, we have fisher-iris dataset, and 4-dimensional feature space. We will reduce it to 2-dimensional feature space. We will only consider **sepal length and sepal width** for our classification problem. Hence, we will have 3 combinations possible in 2-D space and we will use SVM and print classification report in each case.

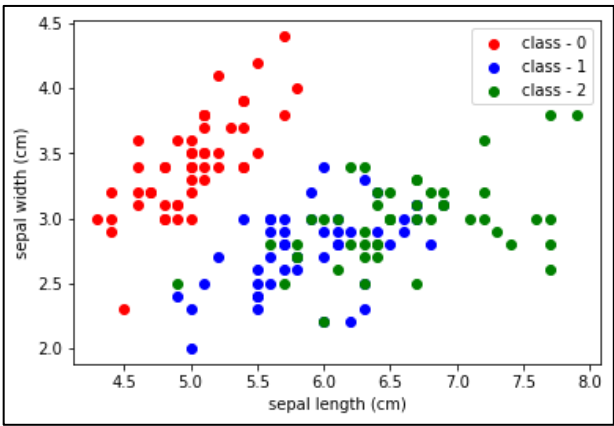


Figure 14: sepal length versus sepal width

3.1 Max-margin Hyperplane

The three cases and respective classification report are shown below.

3.1.1 Class 0 and Class 1

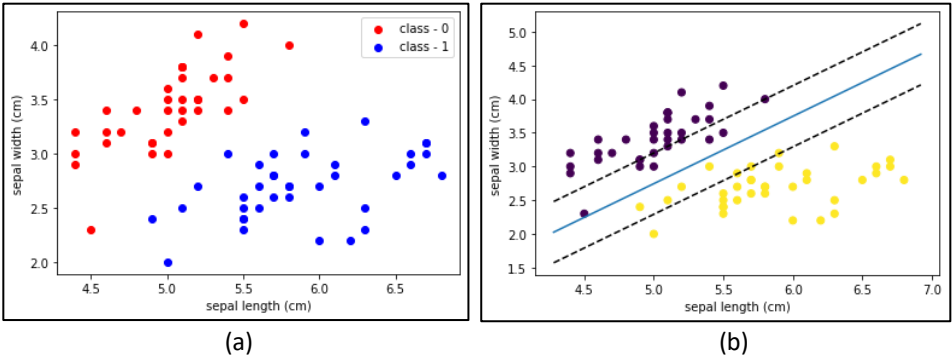


Figure 15: (a) scatter plot of sepal length versus sepal width (only class 0 and 1)
(b) SVM on the data from figure 15(a)

	precision	recall	f1-score	support
0	1.00	1.00	1.00	16
1	1.00	1.00	1.00	14
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Figure 16: Classification report of SVM generated in figure 15(b)

3.1.2 Class 1 and Class 2

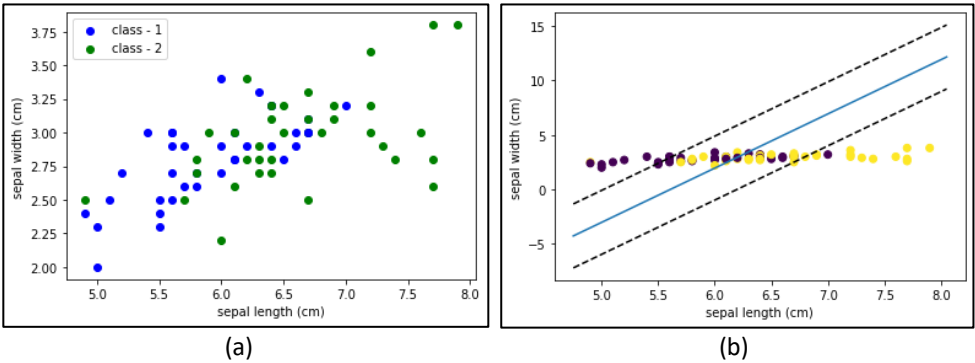


Figure 17: (a) scatter plot of sepal length versus sepal width (only class 1 and 2)
(b) SVM on the data from figure 17(a)

	precision	recall	f1-score	support
1	0.77	0.62	0.69	16
2	0.65	0.79	0.71	14
accuracy			0.70	30
macro avg	0.71	0.71	0.70	30
weighted avg	0.71	0.70	0.70	30

Figure 18: Classification report of SVM generated in figure 17(b)

3.1.3 Class 0 and Class 2

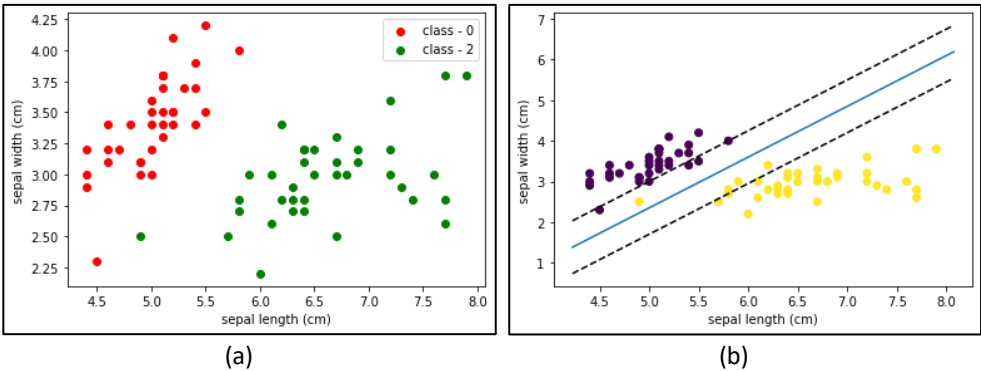


Figure 19: (a) scatter plot of sepal length versus sepal width (only class 0 and 2)
(b) SVM on the data from figure 19(a)

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

Figure 20: Classification report of SVM generated in figure 19(b)

3.1.4 Analysis

As we could observe, class 0 is quite separated, whereas, class1 and class 2 have considerable overlap (figure 14). Hence, SVM worked excellent in both cases, where case 0 examples were involved i.e. figure 15(b) and figure 19(b). Accuracy in both the cases was 100%. Also, in figure 19(b), SVM generated a classifier, however the hyperplane exhibited efficiency of only 70% on the test data.

Hence, we could conclude that SVM is a powerful tool. There were examples, which had overlap with the margin area, but SVM performed quite well on the test data.

3.2 Penalty Parameter (C)

Penalty parameter describes, how much we want to penalize the misclassified points. For example, if training data has many outliers, we know that many points are going to be misclassified. If we keep a higher value of C, we will penalize these points more, and hence higher value of C is expected to make better predictions. Too low values of C almost ignore the misclassified points. Hence, low values of C are similar to case of underfitting. The model won't have good accuracy and fitting of the hyperplane might not be optimum. Hence, we can assume that accuracy should increase and testing error should decrease if value of C is increased.

	precision	recall	f1-score	support
0	0.27	1.00	0.42	12
1	0.00	0.00	0.00	16
2	0.00	0.00	0.00	17
accuracy			0.27	45
macro avg	0.09	0.33	0.14	45
weighted avg	0.07	0.27	0.11	45

Figure 21: Classification report of SVM when C = 0.001

	precision	recall	f1-score	support
0	1.00	0.92	0.96	12
1	0.56	0.56	0.56	16
2	0.61	0.65	0.63	17
accuracy			0.69	45
macro avg	0.72	0.71	0.72	45
weighted avg	0.70	0.69	0.69	45

Figure 22: Classification report of SVM when C = 1000

As we predicted, accuracy in case of C = 1000, is almost 0.70, whereas, in C = 0.001, accuracy is almost 0.27. Also, all other parameters – precision, recall and f1 score are poor in low value of C. When C = 0.001, no value of class 1 or class 2 was predicted correctly (figure 21), i.e. true positives in both these cases are zero.

Important point to note here is that these classification reports were generated for all three classes simultaneously. These classes were not divided in pairs of twos for three times. Through this process, we could analyze whole data at once. Hence above classification reports provide compact analysis of whole fisher-iris dataset.

3.3 Kernel – ‘RBF’

Till now, our hyperplane was linear. When we change our kernel from linear to rbf (radial basis function), the decision boundary doesn’t remain linear. If our data is in form of concentric circles, then linear classifier can never construct a decision boundary with high accuracy. However, ‘rbf’ kernel works as a non-linear SVM. The following table lays down classification reports of all the sets, as used in section 3.1. Kernel used in all the cases is rbf.

C = 1	Class-0 vs Class-1		precision	recall	f1-score	support
		0	1.00	1.00	1.00	16
		1	1.00	1.00	1.00	14
		accuracy			1.00	30
		macro avg	1.00	1.00	1.00	30
		weighted avg	1.00	1.00	1.00	30
C = 1	Class-1 vs Class-2		precision	recall	f1-score	support
		1	0.79	0.69	0.73	16
		2	0.69	0.79	0.73	14
		accuracy			0.73	30
		macro avg	0.74	0.74	0.73	30
		weighted avg	0.74	0.73	0.73	30
C = 1	Class-0 vs Class-2		precision	recall	f1-score	support
		0	1.00	1.00	1.00	16
		2	1.00	1.00	1.00	14
		accuracy			1.00	30
		macro avg	1.00	1.00	1.00	30
		weighted avg	1.00	1.00	1.00	30

Table 1: Classification matrices for different cases: kernel function = rbf and C = 1

C = 1000	Class-0 vs Class-1		precision	recall	f1-score	support	
			0	1.00	1.00	1.00	16
			1	1.00	1.00	1.00	14
			accuracy			1.00	30
			macro avg	1.00	1.00	1.00	30
			weighted avg	1.00	1.00	1.00	30
	Class-1 vs Class-2		precision	recall	f1-score	support	
			1	0.75	0.56	0.64	16
			2	0.61	0.79	0.69	14
accuracy					0.67	30	
macro avg			0.68	0.67	0.67	30	
weighted avg			0.69	0.67	0.66	30	
Class-0 vs Class-2		precision	recall	f1-score	support		
		0	1.00	1.00	1.00	16	
		2	1.00	1.00	1.00	14	
		accuracy			1.00	30	
		macro avg	1.00	1.00	1.00	30	
		weighted avg	1.00	1.00	1.00	30	

Table 2: Classification matrices for different cases: kernel function = rbf and C = 1000

C = 0.001	Class-0 vs Class-1		precision	recall	f1-score	support
		0	0.00	0.00	0.00	16
		1	0.47	1.00	0.64	14
		accuracy			0.47	30
		macro avg	0.23	0.50	0.32	30
		weighted avg	0.22	0.47	0.30	30
	Class-1 vs Class-2		precision	recall	f1-score	support
		1	0.00	0.00	0.00	16
		2	0.47	1.00	0.64	14
accuracy				0.47	30	
macro avg		0.23	0.50	0.32	30	
weighted avg		0.22	0.47	0.30	30	
Class-0 vs Class-2		precision	recall	f1-score	support	
	0	0.00	0.00	0.00	16	
	2	0.47	1.00	0.64	14	
	accuracy			0.47	30	
	macro avg	0.23	0.50	0.32	30	
	weighted avg	0.22	0.47	0.30	30	

Table 3: Classification matrices for different cases: kernel function = rbf and C = 0.001

RBF kernel has overall improved the performance of every SVM classifier. Even at $C = 0.001$, linear kernel performed worse than rbf kernel. Worst-case accuracy of both $C = 1$ and $C = 1000$, is more than 70%. Results of this section could be compared with results from both of the above sections. Hence, we could observe the effect of kernel and effect of penalty parameter (C) on the accuracy of SVM model.

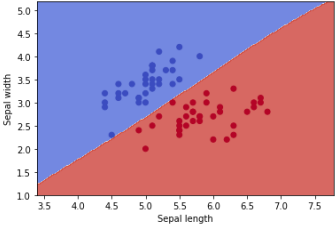
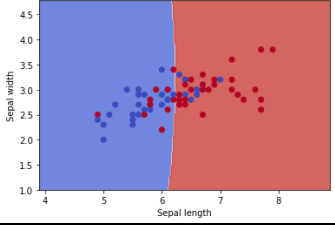
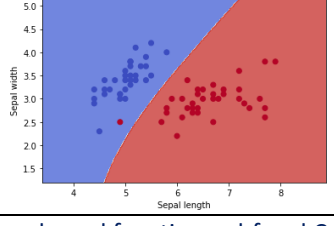
C = 1	Class-0 vs Class-1	
	Class-1 vs Class-2	
	Class-0 vs Class-2	

Table 4: Plots for different cases: kernel function = rbf and C = 1

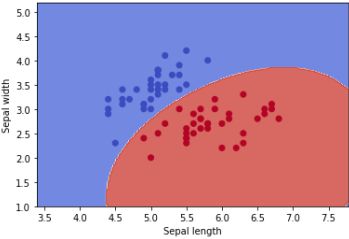
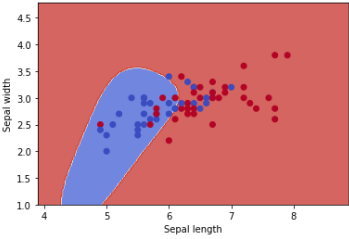
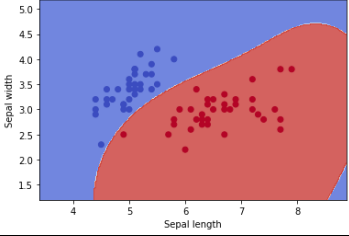
C = 1000	Class-0 vs Class-1	
	Class-1 vs Class-2	
	Class-0 vs Class-2	

Table 5: Plots for different cases: kernel function = rbf and C = 1000

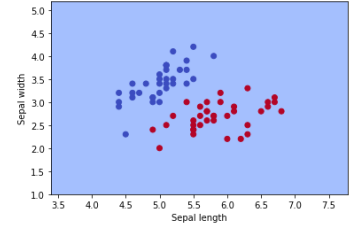
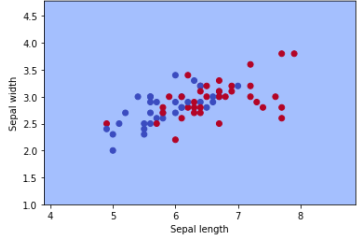
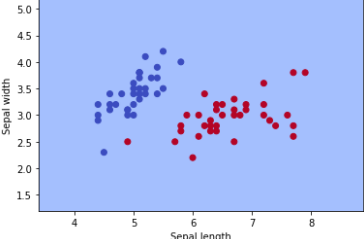
C = 0.001	Class-0 vs Class-1	
	Class-1 vs Class-2	
	Class-0 vs Class-2	

Table 6: Plots for different cases: kernel function = rbf and C = 0.001

The contour plots are most complex in case of C = 1000. Hence, the complexity of SVM classifier is dependent on the value of penalty parameter. There are no contour plots in table 6, hence performance is worst in case of C = 0.001. The corresponding classification reports in table 1-3, justifies the complexity of the plots.

4 Summary

In the report, we observed various dimensionality reduction techniques and analyzed their performance. Starting from the eigenfaces, we observed practical application of dimensionality reduction. By plotting the ghost images, we also observed the loss of information during dimensionality reduction. In the second section, we deployed PCA, LDA and tSNE, on fisher-iris dataset. Through plots and classification report, we observed the change in data. Finally, in the third section, we used SVM for prediction on fisher-iris dataset. Initially, linear kernel was used. Then we observed the effect of penalty parameter on accuracy of the predicted values. Finally, in the last part, we compared, how kernel and penalty parameter, simultaneously, affect the performance of our model.

5 References

- [1] Google Collab Section 1:
https://colab.research.google.com/drive/10mlfSD1EdijZr87tvbM_x_fmrg0V-2a?usp=sharing
- [2] Google Collab Section 2:
<https://colab.research.google.com/drive/1ynWTV94HxIjMizX84hcR6KB7pKAQNH9W?usp=sharing>
- [3] Google Collab Section 3:
<https://colab.research.google.com/drive/1KPVNksrawyUpuG9HUiG2eE1RQJVONUDJ?usp=sharing>
- [4] Webpage: <https://towardsdatascience.com/a-guide-to-svm-parameter-tuning-8bfe6b8a452c>
- [5] Webpage: <https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60>
- [6] Webpage: <https://towardsdatascience.com/end-to-end-topic-modeling-in-python-latent-dirichlet-allocation-lda-35ce4ed6b3e0>
- [7] Webpage: <https://www.kdnuggets.com/2018/08/introduction-t-sne-python.html>
- [8] Webpage: <https://towardsdatascience.com/eigenfaces-face-classification-in-python-7b8d2af3d3ea>
- [9] Webpage: https://scikit-learn.org/0.18/auto_examples/svm/plot_iris.html