Project Report

SMAI

Anshul Lahoti(20161183)

10th November, 2018

1. Introduction

In this project we were supposed to classify the CIFAR - 10 data with various reduction and classification techniques and tweak the hyper-parameters of each technique and compare the results.

The Techniques used for reducing the dimensions are:

1. Principal Component Analysis

2. Kernel Principal Component Analysis

3. Linear Discriminant Analysis

The techniques used for classification are:

1. Logistic Regression

2. Linear Support Vector Machines

3. Kernel Support Vector Machines

4. Multi-Layer Perceptron

2. Logistic Regression

Logistic Regression is a linear classification technique widely used in classification. In this project we applied logistic regression on top of all the dimension reduction techniques mentioned above(LDA, PCA, KernelPCA).

I got the most optimum result when No. of Dimensions in PCA is 64(checked for 8, 16, 32, 64). In case of Kernel-PCA it is 64(checked for 8, 16, 32, 64). In case of LDA it is 9(checked for 3, 6, 9).

Hyper-parameters that were chosen are as follows:

• In case of PCA with Logistic Regression, No. of dimensions in PCA and C values for Logistic Regression.

• In case of KernelPCA with Logistic Regression, No. of dimensions in KernelPCA and C values for Logistic Regression.

• In case of LDA with Logistic Regression, No. of dimensions in LDA and C values for Logistic Regression.

No of Dimensions: in case of all the dimension reduction techniques is just the no. of features to be kept. So, the best N features are taken. Usually it is a trade-off between no. of dimensions and time complexity.

C value: In Logistic Regression C value is the inverse of regularization strength.That means smaller C value specify stronger Regularization.

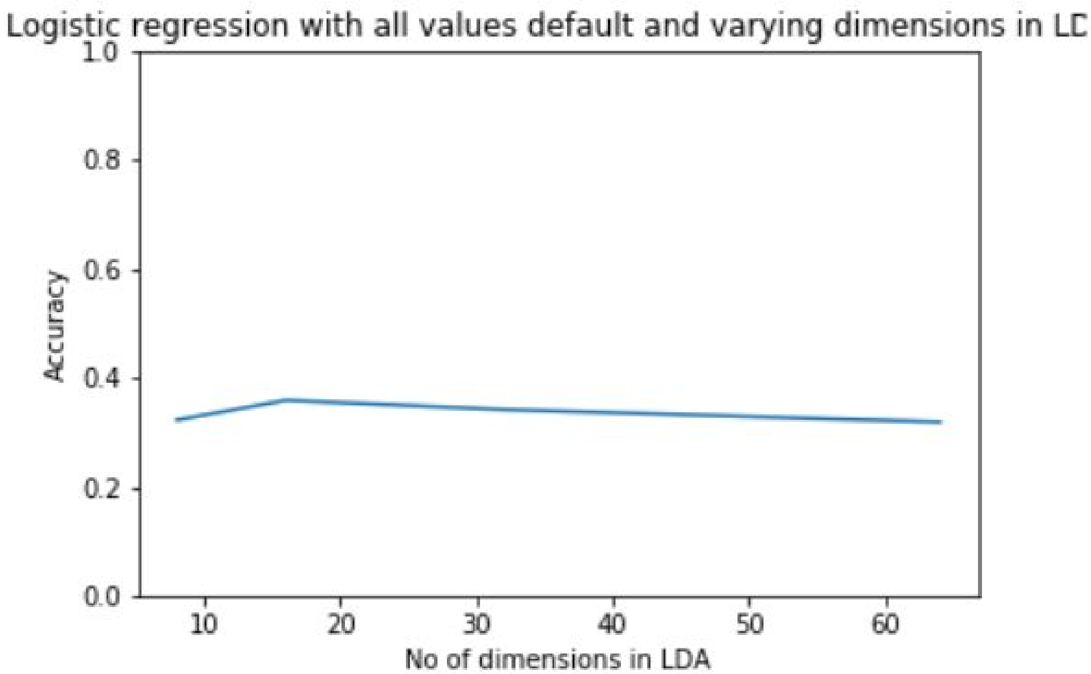

Figure 1: LDA and Log Reg

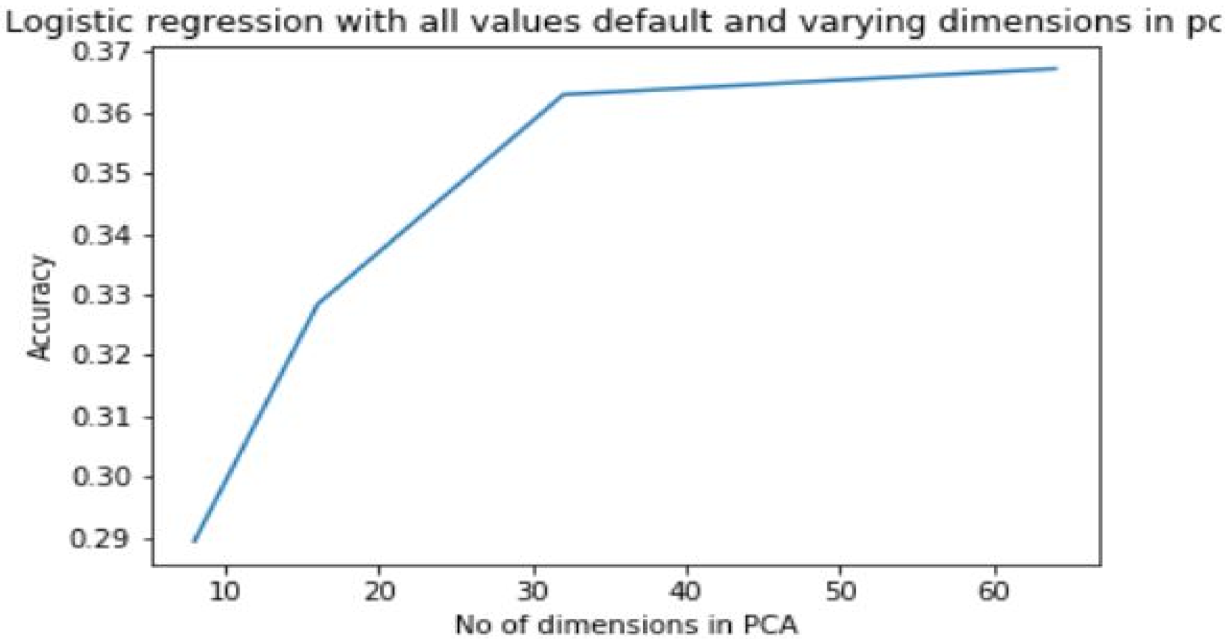

Figure 2: PCA and Log Reg

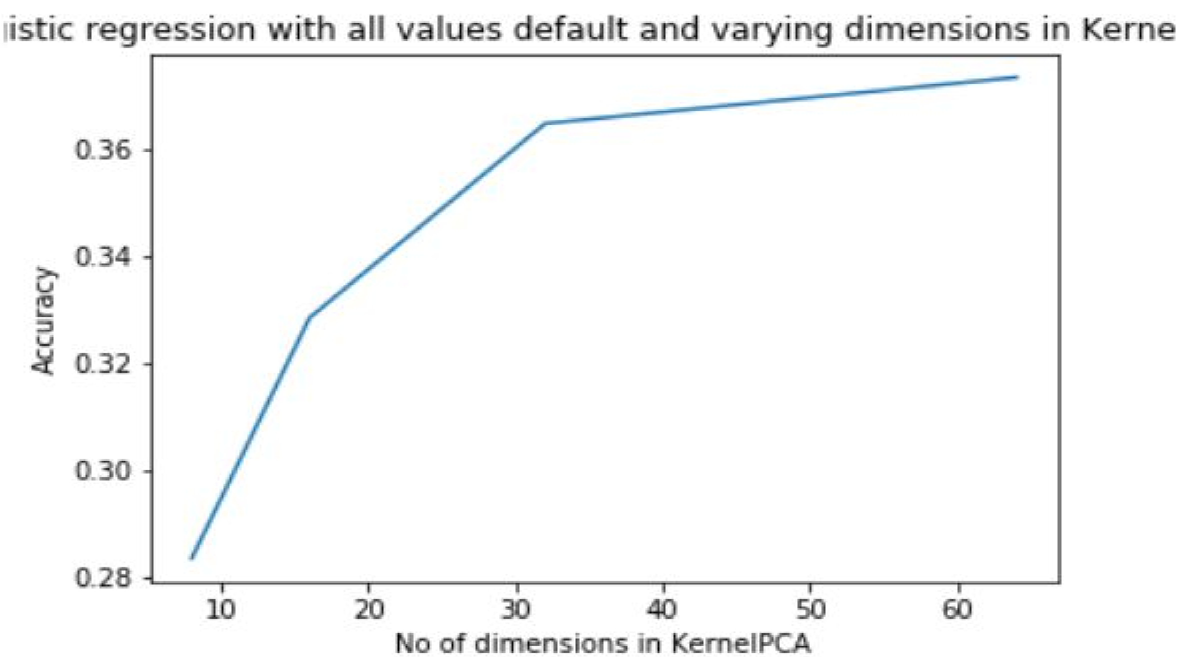

Figure 3: Kernel-PCA and Log Reg

	n components	C value	mean_train_score	mean_test_score
0	8	1	0.295372	0.289374
1	16	1	0.334999	0.328378
2	32	1	0.382498	0.362993
3	64	1	0.410811	0.367247
4	8	0.05	0.295809	0.288751
5	16	0.05	0.335811	0.329378
6	32	0.05	0.382623	0.364243
7	64	0.05	0.409748	0.367745
8	8	0.001	0.288246	0.285627
9	16	0.001	0.331124	0.326251
10	32	0.001	0.377623	0.361494
11	64	0.001	0.40731	0.364867

Table 1: PCA: n components, Log. Reg: 'C' value

	n components	C value	mean_train_score	mean_test_score
0	8	1	0.292871	0.283374
1	8	0.05	0.292996	0.282747
2	8	0.001	0.286745	0.282372
3	16	1	0.341561	0.328375
4	16	0.05	0.341937	0.328125
5	16	0.001	0.335247	0.323869
6	32	1	0.388874	0.364745
7	32	0.05	0.388936	0.364994
8	32	0.001	0.383498	0.363246
9	64	1	0.415811	0.373376
10	64	0.05	0.415187	0.374625
11	64	0.001	0.406686	0.372999

Table 2: Kernel-PCA: n components, Log. Reg: 'C' value

	n components	C value	mean_train_score	mean_test_score
0	3	1	0.239501	0.238379
1	3	0.05	0.237876	0.237753
2	3	0.001	0.230688	0.230128
3	6	1	0.279877	0.281251
4	6	0.05	0.28019	0.279625
5	6	0.001	0.275439	0.274499
6	9	1	0.318251	0.310253
7	9	0.05	0.318689	0.309752
8	9	0.001	0.315001	0.306874

Table 3: LDA: n components, Log. Reg: 'C' value

3. LinearSVM

I got the most optimum result when No. of Dimensions in PCA is 64(checked for 8, 16, 32, 64). In case of Kernel-PCA it is 64(checked for 8, 16, 32, 64). In case of Kernel-PCA it is 64(checked for 8, 16, 32, 64). In case of LDA it is 6(checked for 3, 6, 9).

Similar to Logistic Regression the hyper parameters in this case were same for the reduction techniques and C a penalty parameter of the error term was used. Higher the C value lower error tolerance, whereas lower the C value higher is the error tolerance.

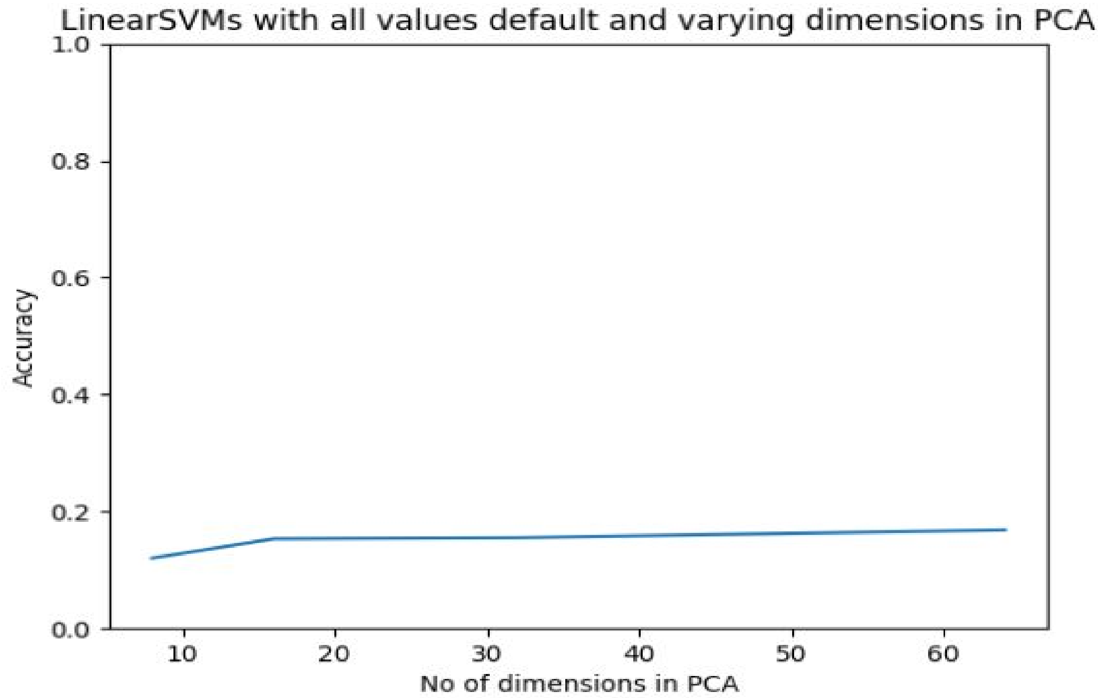

Figure 4: PCA and Linear SVM

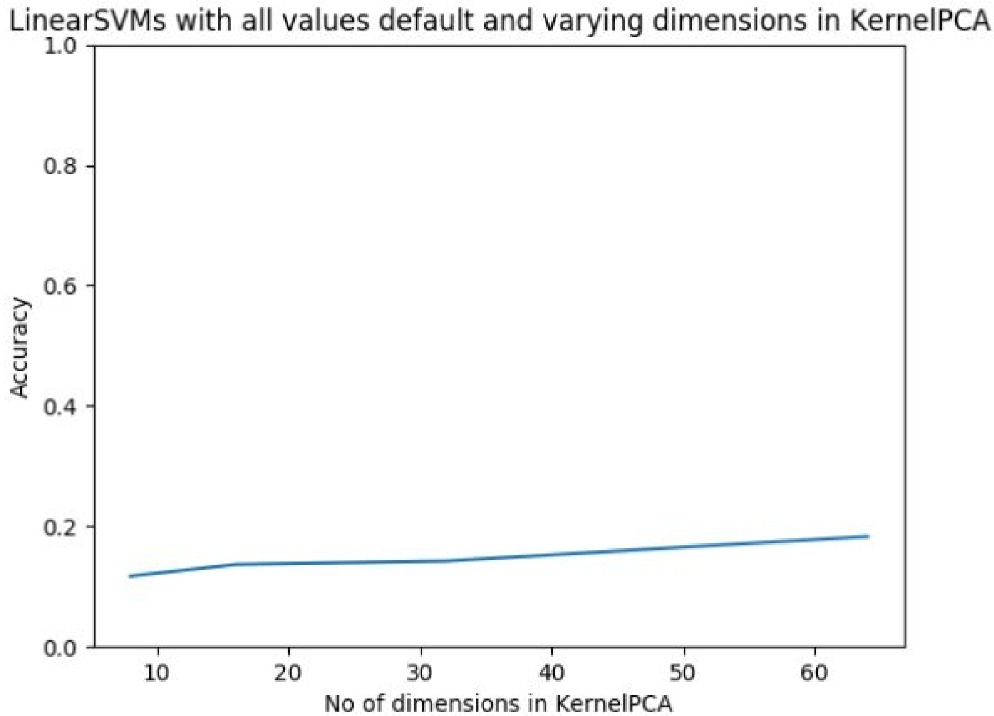

Figure 5: Kernel-PCA and Linear SVM

	parameters	mean_test_score	mean_train_score
0	{'n components': 8, 'C': 1}	0.114628	0.117438
1	{'n components': 8, 'C': 0.05}	0.10812	0.109566
2	{'n components': 8, 'C': 0.001}	0.150868	0.151319
3	{'n components': 16, 'C': 1}	0.136246	0.13669
4	{'n components': 16, 'C': 0.05}	0.143132	0.150998
5	{'n components': 16, 'C': 0.001}	0.147129	0.146936
6	{'n components': 32, 'C': 1}	0.143118	0.142441
7	{'n components': 32, 'C': 0.05}	0.162878	0.163563
8	{'n components': 32, 'C': 0.001}	0.151882	0.154995
9	{'n components': 64, 'C': 1}	0.180774	0.182925
10	{'n components': 64, 'C': 0.05}	0.137757	0.147684

Table 4: Kernel-PCA: n components, LinearSVM: 'C' value

	params	mean_test_score	mean_train_score
0	{'C': 1, 'n components': 3}	0.185996	0.558933
1	{'C': 0.05, 'n components': 3}	0.183997	0.553683
2	{'C': 0.001, 'n components': 3}	0.17975	0.48168
3	{'C': 1, 'n components': 6}	0.192261	0.786379
4	{'C': 0.05, 'n components': 6}	0.190887	0.784128
5	{'C': 0.001, 'n components': 6}	0.189513	0.746055
6	{'C': 1, 'n components': 9}	0.189751	0.955189
7	{'C': 0.05, 'n components': 9}	0.188626	0.955376
8	{'C': 0.001, 'n components': 9}	0.187878	0.955126

Table 5: LDA: n components, LinearSVM: 'C' value

	params	mean_test_score	mean_train_score
0	{'n components': 8, 'C': 1}	0.120755	0.120375
1	{'n components': 16, 'C': 1}	0.15162	0.153566
2	{'n components': 32, 'C': 1}	0.153982	0.155261
3	{'n components': 64, 'C': 1}	0.167115	0.168631
4	{'n components': 8, 'C': 0.05}	0.121261	0.131554
5	{'n components': 16, 'C': 0.05}	0.154261	0.154308
6	{'n components': 32, 'C': 0.05}	0.142985	0.146197
7	{'n components': 64, 'C': 0.05}	0.165617	0.173127
8	{'n components': 8, 'C': 0.001}	0.130903	0.131737
9	{'n components': 16, 'C': 0.001}	0.115385	0.112561
10	{'n components': 32, 'C': 0.001}	0.163005	0.168124
11	{'n components': 64, 'C': 0.001}	0.158909	0.172921

Table 6: PCA: n components, LinearSVM: 'C' value

4. SVM

Support Vector Machines are similar to LinearSVM except the fact that instead of linear kernel, rbf kernel is used. Using rbf kernel is like increasing the no. of dimensions of the feature vector using which we are able to classify using non-linear decision boundaries as well.

I got the most optimum result when No. of Dimensions in PCA is 32(checked for 8, 32, 64). In case of Kernel-PCA it is 64(checked for 8, 16, 32, 64, 128). In case of LDA it is 9(checked for 3, 6, 9).

Hyper-parameters that were tweaked are:

1. No. of dimensions for LDA, PCA and KernelPCA

2. C value of SVM (the penalty for error term as explained in LinearSVM)

3. Gamma is the kernel coefficient.

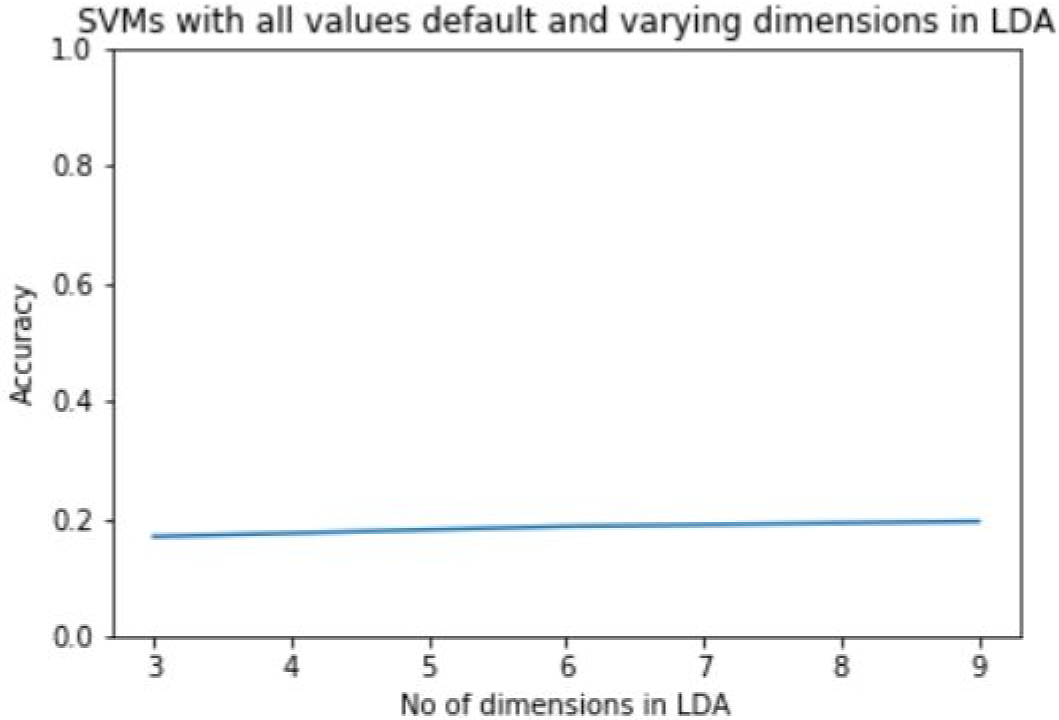

Figure 6: LDA and SVM

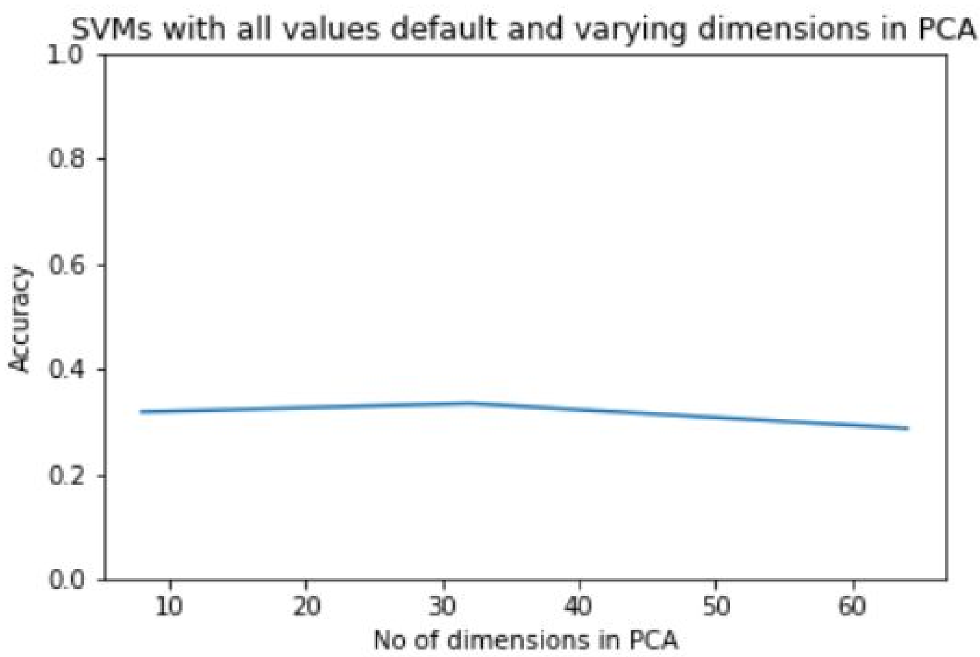

Figure 7: PCA and SVM

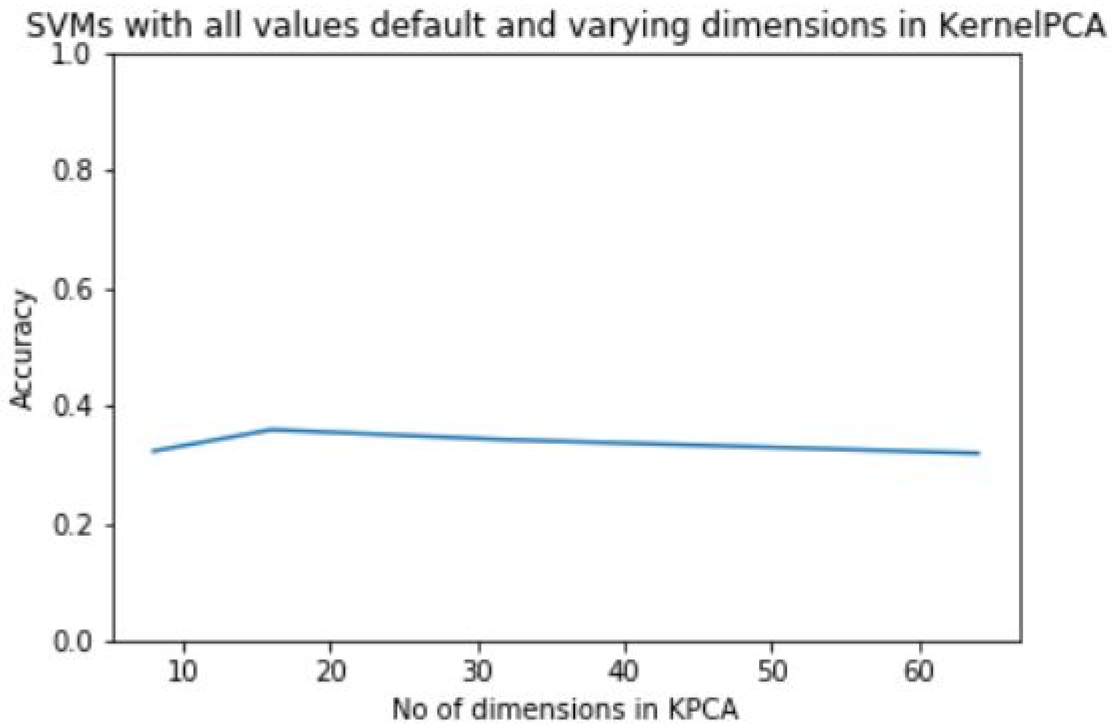

Figure 8: Kernel-PCA and SVM

	n components	C value	Gamma	mean_train_score	mean_test_score
0	3	1	0.0001	0.518081	0.169368
1	3	1	1E-06	0.103	0.103
2	3	1	1E-05	0.103	0.103
3	3	1	1E-07	0.103	0.103
4	3	0.05	0.0001	0.103	0.103
5	3	0.05	1E-06	0.103	0.103
6	3	0.05	1E-05	0.103	0.103
7	3	0.05	1E-07	0.103	0.103
8	3	0.001	0.0001	0.103	0.103
9	3	0.001	1E-06	0.103	0.103
10	3	0.001	1E-05	0.103	0.103
11	3	0.001	1E-07	0.103	0.103
12	6	1	0.0001	0.686752	0.186622
13	6	1	1E-06	0.103	0.103
14	6	1	1E-05	0.103	0.10325
15	6	1	1E-07	0.103	0.103
16	6	0.05	0.0001	0.103	0.103
17	6	0.05	1E-06	0.103	0.103
18	6	0.05	1E-05	0.103	0.103
19	6	0.05	1E-07	0.103	0.103
20	6	0.001	0.0001	0.103	0.103
21	6	0.001	1E-06	0.103	0.103
22	6	0.001	1E-05	0.103	0.103
23	6	0.001	1E-07	0.103	0.103
24	9	1	0.0001	0.956749	0.190502
25	9	1	1E-06	0.103	0.103
26	9	1	1E-05	0.103	0.103125
27	9	1	1E-07	0.103	0.103
28	9	0.05	0.0001	0.103	0.103
29	9	0.05	1E-06	0.103	0.103
30	9	0.05	1E-05	0.103	0.103
31	9	0.05	1E-07	0.103	0.103
32	9	0.001	0.0001	0.103	0.103
33	9	0.001	1E-06	0.103	0.103
34	9	0.001	1E-05	0.103	0.103
35	9	0.001	1E-07	0.103	0.103

	n components	C value	Gamma	mean_train_score	mean_test_score
0	8	1	0.0001	1	0.105625
1	8	1	1E-06	0.768187	0.322371
2	8	1	1E-05	1	0.13775

	n components	C value	Gamma	mean_train_score	mean_test_score
3	8	1	1E-07	0.391624	0.342124
4	8	0.05	0.0001	0.104875	0.104875
5	8	0.05	1E-06	0.166813	0.156372
6	8	0.05	1E-05	0.104875	0.104875
7	8	0.05	1E-07	0.306562	0.29325
8	8	0.001	0.0001	0.104875	0.104875
9	8	0.001	1E-06	0.104875	0.104875
10	8	0.001	1E-05	0.104875	0.104875
11	8	0.001	1E-07	0.104875	0.104875
12	16	1	0.0001	1	0.105125
13	16	1	1E-06	0.959999	0.358496
14	16	1	1E-05	1	0.110375
15	16	1	1E-07	0.49575	0.389748
16	16	0.05	0.0001	0.104875	0.104875
17	16	0.05	1E-06	0.119062	0.11675
18	16	0.05	1E-05	0.104875	0.104875
19	16	0.05	1E-07	0.336187	0.319999
20	16	0.001	0.0001	0.104875	0.104875
21	16	0.001	1E-06	0.104875	0.104875
22	16	0.001	1E-05	0.104875	0.104875
23	16	0.001	1E-07	0.104875	0.104875
24	32	1	0.0001	1	0.104875
25	32	1	1E-06	0.994438	0.341497
26	32	1	1E-05	1	0.10675
27	32	1	1E-07	0.601497	0.429375
28	32	0.05	0.0001	0.104875	0.104875
29	32	0.05	1E-06	0.104937	0.105
30	32	0.05	1E-05	0.104875	0.104875
31	32	0.05	1E-07	0.346249	0.326374
32	32	0.001	0.0001	0.104875	0.104875
33	32	0.001	1E-06	0.104875	0.104875
34	32	0.001	1E-05	0.104875	0.104875
35	32	0.001	1E-07	0.104875	0.104875
36	64	1	0.0001	1	0.104875
37	64	1	1E-06	0.998	0.318627
38	64	1	1E-05	1	0.105625
39	64	1	1E-07	0.6645	0.435748
40	64	0.05	0.0001	0.104875	0.104875
41	64	0.05	1E-06	0.104875	0.104875
42	64	0.05	1E-05	0.104875	0.104875
43	64	0.05	1E-07	0.348061	0.326248
44	64	0.001	0.0001	0.104875	0.104875
45	64	0.001	1E-06	0.104875	0.104875

	n components	C value	Gamma	mean_train_score	mean_test_score
46	64	0.001	1E-05	0.104875	0.104875
47	64	0.001	1E-07	0.104875	0.104875
48	128	1	0.0001	1	0.104875
49	128	1	1E-06	0.9995	0.252626
50	128	1	1E-05	1	0.105375
51	128	1	1E-07	0.716374	0.442871
52	128	0.05	0.0001	0.104875	0.104875
53	128	0.05	1E-06	0.104875	0.104875
54	128	0.05	1E-05	0.104875	0.104875
55	128	0.05	1E-07	0.350373	0.325498
56	128	0.001	0.0001	0.104875	0.104875
57	128	0.001	1E-06	0.104875	0.104875
58	128	0.001	1E-05	0.104875	0.104875
59	128	0.001	1E-07	0.104875	0.104875

Table 9: Kernel-PCA: n components, SVM: ’C’ value, Gamma

	n components	C value	Gamma	mean_train_score	mean_test_score
0	8	1	0.0001	1	0.1045
1	8	1	1E-06	0.770314	0.317628
2	8	1	1E-05	1	0.146126
3	8	0.05	0.0001	0.10375	0.10375
4	8	0.05	1E-06	0.154374	0.144382
5	8	0.05	1E-05	0.10375	0.10375
6	8	0.001	0.0001	0.10375	0.10375
7	8	0.001	1E-06	0.10375	0.10375
8	8	0.001	1E-05	0.10375	0.10375
9	32	1	0.0001	1	0.10375
10	32	1	1E-06	0.993938	0.333625
11	32	1	1E-05	1	0.105125
12	32	0.05	0.0001	0.10375	0.10375
13	32	0.05	1E-06	0.10375	0.10375
14	32	0.05	1E-05	0.10375	0.10375
15	32	0.001	0.0001	0.10375	0.10375
16	32	0.001	1E-06	0.10375	0.10375
17	32	0.001	1E-05	0.10375	0.10375
18	64	1	0.0001	1	0.10375
19	64	1	1E-06	0.998438	0.286475
20	64	1	1E-05	1	0.1045
21	64	0.05	0.0001	0.10375	0.10375
22	64	0.05	1E-06	0.10375	0.10375
23	64	0.05	1E-05	0.10375	0.10375
24	64	0.001	0.0001	0.10375	0.10375
25	64	0.001	1E-06	0.10375	0.10375
26	64	0.001	1E-05	0.10375	0.10375

Table 10: PCA: n components, SVM: 'C' value, Gamma

5. MLP

Multi-Layer Perceptron is used to classify non-linear functions. It uses different kinds of activation functions and various other parameters to fit the data. The most widely used activation functions are ReLu and tanh. So here, I have tweaked this activation function and have calculated the corresponding accuracies on top of that.

I got the most optimum result when No. of Dimensions in PCA is 64 (checked for 8, 32, 64). In case of Kernel-PCA it is 64(checked for 8, 16, 32, 64, 128). In case of LDA it is 6(checked for 3, 6, 9)

Hyper-parameters that were tweaked are:

1. No. of dimensions for LDA, PCA and KernelPCA

2. Activation function of the nodes. (ReLu and tanh)

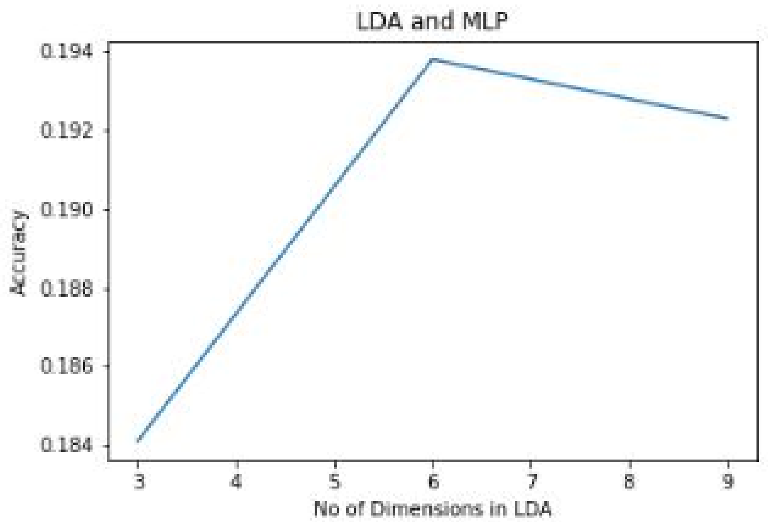

Figure 9: LDA and MLP

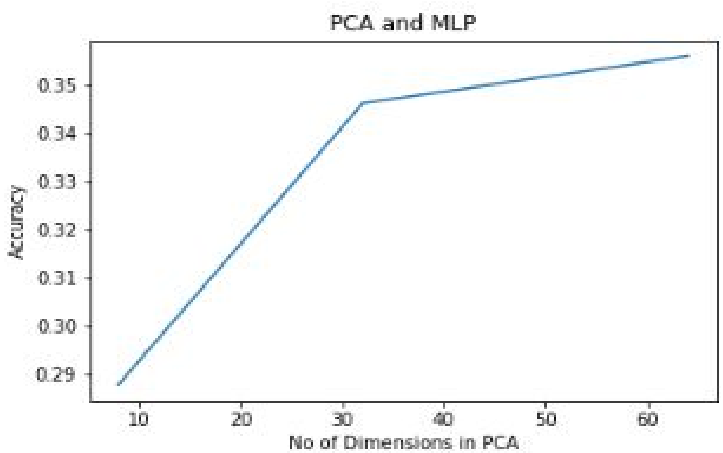

Figure 10: PCA and MLP

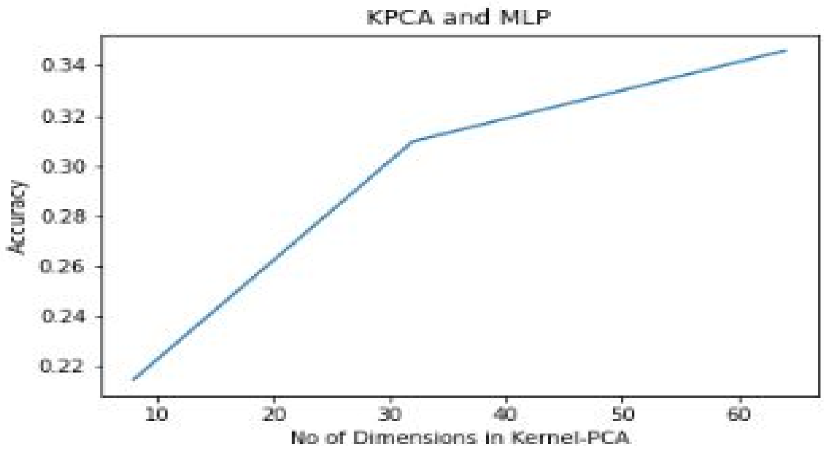

Figure 11: PCA and MLP

	n components	activation	mean_train_score	mean_test_score
0	3	tanh	0.608871	0.184125
1	3	relu	0.615932	0.185873
2	6	tanh	0.828124	0.193873
3	6	relu	0.838061	0.194744
4	9	tanh	0.975	0.192372
5	9	relu	0.9784	0.1893

Table 11: LDA: n components, MLP: activation function

	n components	activation	mean_train_score	mean_test_score
0	8	tanh	0.682	0.2863
1	8	relu	0.63832	0.288
2	32	tanh	0.76353	0.325
3	32	relu	0.778243	0.3462
4	64	tanh	0.5231	0.341
5	64	relu	0.55612	0.356

Table 12: PCA: n components, MLP: activation function

	n components	activation	mean_train_score	mean_test_score
0	8	tanh	0.5612	0.216
1	8	relu	0.5834	0.258
2	32	tanh	0.673	0.3020
3	32	relu	0.634	0.3226
4	64	tanh	0.5523	0.3568
5	64	relu	0.5842	0.358

Table 13: Kernel-PCA: n components, MLP: activation function

6. Overfitting

A classifier can be considered as a reasonable one only if it performs well on the samples that it has never seen. Otherwise if it is performing really good on the training samples and doesn't perform well on unseen samples then we say that overfitting has happened. Usually the classifiers that try to classify all the training samples tend to overfit. e.g. In the table of LDA along with LinearSVM when the no of components are 9 we can see that it performs really well(more than 95%) on training set whereas due to overfitting the accuracy on testing is really poor. One of the main reasons behind the hyper parameter setting is to avoid overfitting and make model as robust as possible by increasing its accuracy on validation set.

7. Results and Conclusion

So as the tables above suggests we can see that not all the classifiers perform well. Some of them performs quite well on the training set but failed to perform on test set due to overfitting. This overfitting can be tackled by tweaking the hyper parameters and using a validation set. Also we can say that some classifiers performs really well with some specific dimension reduction techniques, some might perform well on the raw data itself. So choosing a right choice of techniques is really important. So after selecting the best reduction-classification technique and setting the parameters the results on test set were obtained as follows:

Classifier	Dimension Reduction	F_Score
Logistic Regression	Kernel-PCA	0.4302
LinearSVM	Kernel-PCA	0.2136
SVM	PCA	0.4518
Multi-Layer Perceptron	PCA	0.397

Table 14: Final Scores

8. Problem Faced

1. First and major problem was computation time for various combinations of classifiers and dimensionality reduction techniques. And highest time was taken when computed for Raw data.
2. To get optimal solution (for each combination) we have to change hyperparameters for various classifiers and dimensionality reduction techniques. So, it was also taking time.