SMAI Mini Project Report

Kunal Garg - 20161067

November 2018

1 Introduction

1.1 Overview

In this project we perform classification task on CIFAR-10 dataset and report the f1-score as well as the accuracy on the test dataset after tuning hyperparameters for different classifiers and dimensionality reduction techniques. We first preform Dimensionality Reduction of data using 3 different methods:

1. PCA
2. LDA
3. KPCA (Nonlinear Representation)

Then we have analyzed accuracy on each of these represenation using four different classifiers:

1. Kernel SVM with RBF Kernel
2. Decision Tree
3. MLP
4. Logistic Regression

1.2 Implementation Details

The code for the project is divided into 12 ipython notebooks(3 for each of the classifier). The results being displayed as part of report can be found in the ipython notebooks. In code implementation, I first performed (80:20) split of train data into train and cross validation data respectively. Then, for each of the classifier, I found the optimal number of dimensions in which data should be reduced for each of dimensional reduction method. This is done by calculating accuracy by varying dimensions without specifying any hyperparameters in classfiers. The variation in dimensions is as follows:

- PCA : [1,2,4,8,16,32,64,128,256,512,1024]
- LDA : [1,2,4,8,16,32,64,128,256]
- KPCA : [1,2,4,8,16,32,64,128,256,512,1024]

Image preprocessing : In case of PCA and KPCA, image preprocessing is done using MinMaxScalar function of sklearn which transforms features by scaling each feature to range of (-1,1). Scaling is critical, when performing PCA as PCA tries to get the features with maximum variance and the variance is high for high magnitude features which skews the PCA towards high magnitude features. We don't need feature scaling in LDA as it is designed to handle this.

Hyperparameter tuning : After finding the optimal dimension for each of dimensional reduction method and corresponding classifier, I reduce the train data into the corresponding dimension and tune hyperparameters of the classifier by varying them and calculating accuracy on the cross validation data. The best hyperparameters for each classifier-representation combination is used to evaluate the test data. There are 50k training and 10k test images in the CIFAR-10 dataset, but I took only one data batch from the set of 5 data batches which has 10k images. After train-val split, we have 8k training images and 2k cross validation images. The values of f1-score and accuracy were coming out to be almost same, so I have majorly mentioned only accuracy in the report. The results mentioned in the report can be improved reasonably if we train our classifers on all 5 data batches.

1.3 Dimensionality Reduction Techniques: PCA, LDA and KPCA

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.

Linear discriminant analysis(LDA) is a statistical procedure used to find a new feature space using a linear combination of features to project the data so that as to maximize classes separability. LDA explicitly attempts to model the difference between the classes of data. PCA on the other hand does not take into account any difference in class.

Kernel Principal component analysis (KPCA) is an extension of conventional PCA to a high dimensional feature space using the "kernel trick". It can extract upto n (number of samples) non-linear principal components without expensive computations.

2 Kernel SVM Classifier

In SVMs, decision function depends on some subset of the training data, called the support vectors and as it uses a subset of training points in the decision function it is also memory efficient. SVM is really effective in high dimensional spaces. Kernel SVM implements the "one-against-one" approach for multi-class classification. If n_class is the number of classes, then $n\_class * (n\_class - 1)/2$ classifiers are constructed and each one trains data from two classes.
Some of the parameters in Kernel SVM classifier include C, γ, kernel, degree. In this project I have used RBF Kernel : $\exp(-\gamma\|x - x'\|^2)$. Though there are many hyperparameters in sklearn library function svm.SVC() but I varied only 2 hyperparameters γ(kernel coefficient) and C(penalty parameter of the error term).

2.1 Dimensionality Reduction using PCA

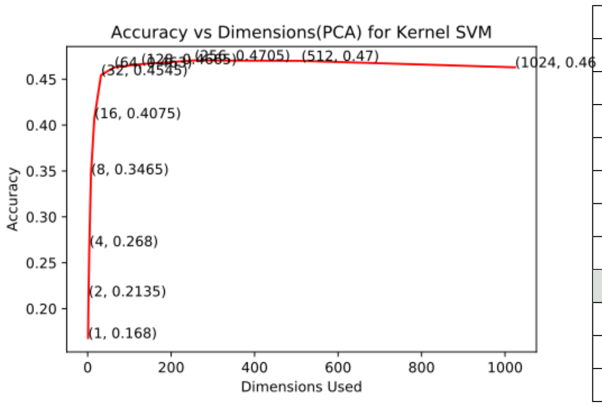

Figure 1: Kernel SVM PCA

	Dimensions	Accuracy
1	1	0.168
2	2	0.2135
3	4	0.268
4	8	0.3465
5	16	0.4075
6	32	0.4545
7	64	0.463
8	128	0.4665
9	256	0.4705
10	512	0.47
11	1024	0.463

Table 1: Number of components taken vs Accuracy(PCA)

We can see from the Figure 1 as well as Table 1 that after a certain number of dimensions of PCA, there is not much increase in accuracy on testing on validation data. This happens because 95% of features are captured by PCA in low dimensional space itself. I have taken 128 as the optimal dimensions. This will be fixed and we will tune hyperparameters based on this. But theoretically we should change dimensions and hyperparameters simultaneously and calculate the accuracies.

In Table 2 we can see both the test and train accuracies obtained by varying C and γ. The best accuracy is obtained at $C = 5 \ and \ \gamma = 0.001$. We can also notice in rows 14-15 and 19-20, the model is **overfitting** as the accuracy on train data is almost 100% but on validation data, its very less(just 10%). So, if we use large γ, we say that there's higher chance of overfitting.

	C	Gamma(γ)	Test Accuracy	Train Accuracy
1	0.5	0.0005	0.410375	0.462625
2	0.5	0.001	0.427375	0.523844
3	0.5	0.01	0.20775	0.749907
4	0.5	0.1	0.10525	0.106062
5	0.5	0.5	0.10525	0.10525
6	1	0.0005	0.430125	0.512094
7	1	0.001	0.442625	0.609063
8	1	0.01	0.35575	0.989813
9	1	0.1	0.106125	1
10	1	0.5	0.10525	1
11	5	0.0005	0.450625	0.687656
12	5	0.001	0.459	0.864281
13	5	0.01	0.366625	1
14	5	0.1	0.1065	1
15	5	0.5	0.10525	1
16	10	0.0005	0.451375	0.78575
17	10	0.001	0.4565	0.940343
18	10	0.01	0.366625	1
19	10	0.1	0.1065	1
20	10	0.5	0.10525	1

Table 2: Variation of validation test accuracy and train accuracy for different hyperparameters(PCA)

2.2 Dimensionality Reduction using LDA

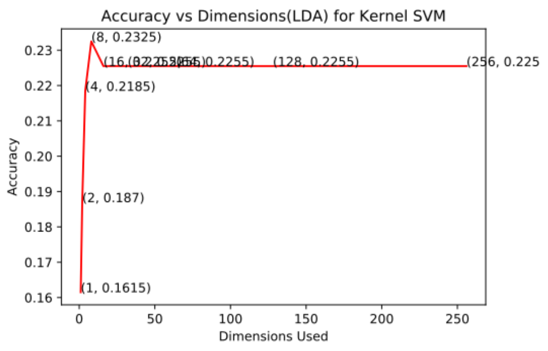

Figure 2: Kernel SVM LDA

	Dimensions	Accuracy
1	1	0.1615
2	2	0.187
3	4	0.2185
4	8	0.2325
5	16	0.2255
6	32	0.2255
7	64	0.2255
8	128	0.2255
9	256	0.2255

Table 3: Number of components taken vs Accuracy(LDA)

From the Figure 2 as well as Table 3 we notice that after dimensions of LDA increases more than 8 there is bit decrease in accuracy and there is no change in accuracy after than. This may have happened because in our case there are 10 categories to classify a sample and LDA is based on principle of class separability. I have taken 8 as the optimal dimensions. As seen in PCA, this will be fixed and we will tune hyperparameters based on this. But theoretically we should change dimensions and hyperparameters simulatenously and calculate the accuracies.

In Table 4 we can see both the test and train accuracies obtained by varying C and γ by fixing principle components to 8 and doing dimensional reduction by LDA. The best accuracy is obtained at C = *10 and* γ = *0.01*. We can again a bit of overfitting in row 15 and row 20 of Table 4 as the accuracy on train data is almost 100% but on validation data, its less than row 18. So, we conclude that if we use large γ, there's higher chance of model getting overfit.

	C	Gamma(γ)	Test Accuracy	Train Accuracy
1	0.5	0.0005	0.239	0.823
2	0.5	0.001	0.2375	0.82075
3	0.5	0.01	0.2365	0.8245
4	0.5	0.1	0.239	0.836875
5	0.5	0.5	0.2005	0.895875
6	1	0.0005	0.2375	0.820625
7	1	0.001	0.239	0.821625
8	1	0.01	0.2375	0.825
9	1	0.1	0.2345	0.842625
10	1	0.5	0.228	0.9265
11	5	0.0005	0.2335	0.822625
12	5	0.001	0.236	0.82425
13	5	0.01	0.2375	0.83025
14	5	0.1	0.2355	0.8675
15	5	0.5	0.212	0.99175
16	10	0.0005	0.235	0.82425
17	10	0.001	0.2335	0.824
18	10	0.01	0.242	0.8325
19	10	0.1	0.234	0.886875
20	10	0.5	0.2135	0.998

Table 4: Variation of validation test accuracy and train accuracy for different hyperparameters(LDA)

2.3 Dimensionality Reduction using KPCA

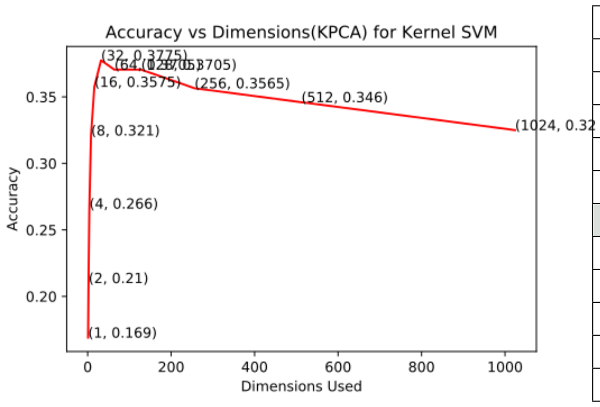

Figure 3: Kernel SVM KPCA

	Dimensions	Accuracy
1	1	0.169
2	2	0.21
3	4	0.266
4	8	0.321
5	16	0.3575
6	32	0.3775
7	64	0.3705
8	128	0.3705
9	256	0.3565
10	512	0.346
11	1024	0.325

Table 5: Number of components taken vs Accuracy(KPCA)

We can see from Figure 3 and Table 5 that the optimal accuracy for KPCA dimensionality reduction is when the number of dimensions are 32. Its a peak as we can see from the figure.

Table 6 shows that the best accuracy on test data is obtained when $C = 10$ *and* $\gamma = 0.5$. We can observe that there is no overfitting in KPCA(42% accuracy on validation and 50% on train data), but there was overfitting for both PCA and LDA when we used there hyperparameter values. Hence, KPCA(nonlinear data representation) helps to prevent overfitting of data.

	C	Gamma(γ)	Test Accuracy	Train Accuracy
1	0.5	0.0005	0.1015	0.103625
2	0.5	0.001	0.1015	0.103625
3	0.5	0.01	0.171	0.15575
4	0.5	0.1	0.329	0.335625
5	0.5	0.5	0.3755	0.395625
6	1	0.0005	0.1015	0.103625
7	1	0.001	0.1015	0.103625
8	1	0.01	0.224	0.2255
9	1	0.1	0.3505	0.362125
10	1	0.5	0.3885	0.419
11	5	0.0005	0.1015	0.103625
12	5	0.001	0.171	0.155625
13	5	0.01	0.3325	0.336
14	5	0.1	0.383	0.40775
15	5	0.5	0.407	0.47225
16	10	0.0005	0.1705	0.155625
17	10	0.001	0.2245	0.226875
18	10	0.01	0.35	0.3625
19	10	0.1	0.3835	0.415375
20	10	0.5	0.423	0.501

Table 6: Variation of validation test accuracy and train accuracy for different hyperparameters(KPCA)

2.4 Conclusion

Thus, we notice that there are entirely different set of hyperparameter values which give best accuracy with different dimensionality reduction techniques. Also, higher is the value of γ, higher is the chance of model getting overfit in case of PCA and LDA(linear dimensionality reduction techniques) but no such this happens in case of KPCA (non-linear dimensionality reduction technique). I increased the value of C and gamma further and noticed that Kernel SVM classifier trained on KPCA reduced data performed even better than previously obtained results.

	Dimensions	C	Gamma	Validation Set Accuracy	Test Set Accuracy
PCA	128	5	0.001	0.463	0.4678
LDA	8	10	0.01	0.242	0.2374
KPCA	32	100	1	0.4475	0.4556

3 Decision Tree Classifier

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. DTs are simple to understand and to interpret and it very easy validate a model using statistical tests. But the problem in this classifier is Decision trees can create over-complex trees an can often lead to overfitting.
Some of the parameters in Decision Tree Classifier include criterion(function to measure the quality of a split), max_depth(maximum depth of the tree), min_samples_split(the minimum number of samples required to split an internal node) etc. The default values of parameters controlling the size of the trees(e.g max_depth) lead to fully grown and unpruned trees which can potentially be very large on some data sets which results in overfitting. Though there are many hyperparameters in sklearn library function tree.DecisionTreeClassifier() but I varied only 2 hyperparameters *criteria* and *max_depth* for this project.

3.1 Dimensionality Reduction using PCA

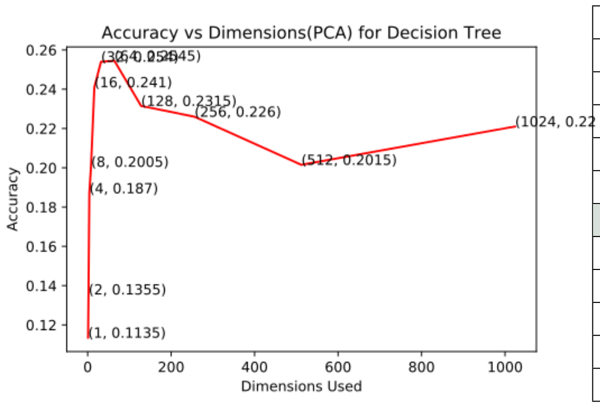

Figure 4: Decision Tree PCA

	Dimensions	Accuracy
1	1	0.1135
2	2	0.136
3	4	0.187
4	8	0.2
5	16	0.241
6	32	0.254
7	64	0.254
8	128	0.232
9	256	0.226
10	512	0.202
11	1024	0.221

Table 7: Number of components taken vs Accuracy(PCA)

From Figure 4 and Table 7 we notice that the accuracy is almost same for PCA components 32 and 64. So, I took a middle value between them and used 40 as the optimal dimension for PCA reduction. The accuracy is decreasing as we move from 64 to 512, which may be because model is getting overfit on the training data.

	Criteria	Max depth	Test Accuracy	Train Accuracy
1	gini	3	0.208	0.226625
2	gini	6	0.2785	0.323375
3	gini	12	0.2555	0.696125
4	gini	30	0.221	1
5	gini	60	0.224	1
6	gini	120	0.2295	1
7	entropy	3	0.2185	0.232625
8	entropy	6	0.26	0.30975
9	entropy	12	0.249	0.747625
10	entropy	30	0.236	1
11	entropy	60	0.234	1
12	entropy	120	0.237	1

Table 8: Variation of validation test accuracy and train accuracy for different hyperparameters(PCA)

In Table 8 we can see both the test and train accuracies obtained by varying Criteria and Max depth of the DT classifier. The best accuracy is obtained at *Criteria = "gini" and Max depth = 6.* We can also notice in rows 4-6 and 10-12, the model is overfitting as the accuracy on train data is almost 100% but on validation data, it is less than the best we have. We can observe that accuracy at Max depth = 6(Early Stopping) has higher accuracy than when Max depth is 12. Also if we further explore the tree the model gets overfit.

3.2 Dimensionality Reduction using LDA

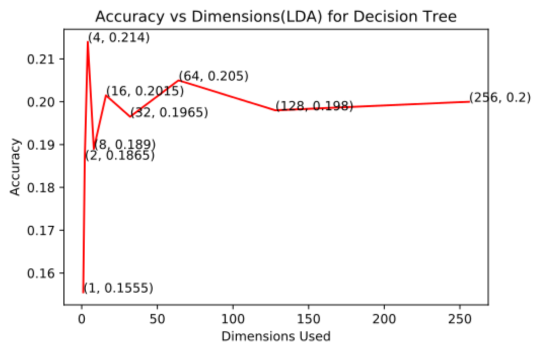

Figure 5: Decision Tree LDA

	Dimensions	Accuracy
1	1	0.1555
2	2	0.186
3	4	0.214
4	8	0.189
5	16	0.202
6	32	0.196
7	64	0.205
8	128	0.198
9	256	0.2

Table 9: Number of components taken vs Accuracy(LDA)

We can from see Figure 5 and Table 9 that the optimal accuracy for LDA dimensionality reduction is when the number of dimensions are 4. I tried varying LDA components from 5-7 also, but the it gives max accuracy when dimension is 4. Also, we can notice that after dimensions of LDA increases more than 8 there is not much change in accuracy. Although there a slight decrease in accuracy, same as what was happening in PCA due to overfitting of the model.

	Criteria	Max depth	Test Accuracy	Train Accuracy
1	gini	3	0.2015	0.44175
2	gini	6	0.2075	0.540625
3	gini	12	0.207	0.7185
4	gini	30	0.196	0.999875
5	gini	60	0.1985	1
6	gini	120	0.1965	1
7	entropy	3	0.2005	0.45375
8	entropy	6	0.209	0.543625
9	entropy	12	0.1965	0.771375
10	entropy	30	0.1965	1
11	entropy	60	0.199	1
12	entropy	120	0.194	1

Table 10: Variation of validation test accuracy and train accuracy for different hyperparameters(LDA)

Table 10 shows that the best accuracy on test data is obtained when *Criteria = "entropy" and Max depth = 6* which is 20.9% but the accuracy when hyperparameters, criteria and max depth are "gini" and $\gamma = 0.5$ is also very close to max accuracy. When we increase the max depth of the tree from 6 to 12, there is not much increase in accuracy on test data but accuracy on train data increased by over 20%, which means that model is starting to get overfit. We can clearly see this if we look at rows 4-6 and rows 10-12, i.e when the max depth is >= 30, model is overfitting as the accuracy on train data is almost 100% but on validation data, it is less than the best we have.

3.3 Dimensionality Reduction using KPCA

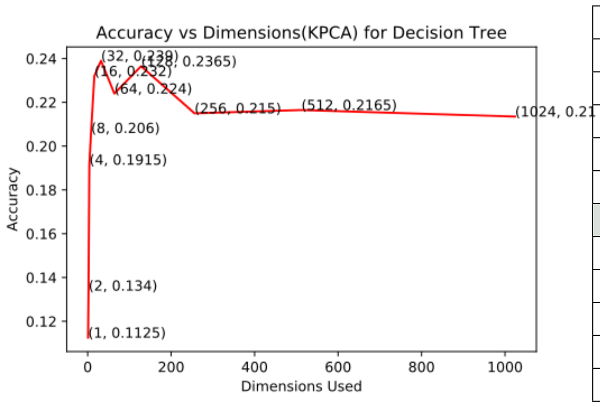

Figure 6: Decision Tree KPCA

	Dimensions	**Accuracy**
1	1	0.1125
2	2	0.134
3	4	0.1915
4	8	0.206
5	16	0.232
6	32	0.239
7	64	0.224
8	128	0.2365
9	256	0.215
10	512	0.2165
11	1024	0.2135

Table 11: Number of components taken vs Accuracy(KPCA)

We can from see Figure 6 and Table 11 that the optimal accuracy for KPCA dimensionality reduction is when the number of dimensions are 32.Its a peak as we can see from the figure. On increasing the dimensions, as we can see from the table that the accuracy is getting reduced, which may happened due to overfitting of the model.

	Criteria	**Max depth**	**Test Accuracy**	**Train Accuracy**
1	gini	3	0.23	0.225625
2	gini	6	0.2755	0.3165
3	gini	12	0.2735	0.690375
4	gini	30	0.2575	1
5	gini	60	0.264	1
6	gini	120	0.267	1
7	entropy	3	0.2295	0.234125
8	entropy	6	0.28	0.31025
9	entropy	12	0.2715	0.7645
10	entropy	30	0.2565	1
11	entropy	60	0.2565	1
12	entropy	120	0.252	1

Table 12: Variation of validation test accuracy and train accuracy for different hyperparameters(KPCA)

Table 12 shows that the best accuracy on test data is obtained when *Criteria = "entropy" and Max depth = 6* which is 28% but the accuracy when hyperparameters, criteria is "gini" and max depth is 6 is also very close to max accuracy. Now, as we increase the max depth of the tree from 6 to 12, there is a decrease in accuracy on test data but accuracy on train data increased by over 40%, which means that model is starting to get overfit. We can clearly see this if we look at rows 4-6 and rows 10-12, i.e when the max depth is >= 30, model is overfitting as the accuracy on train data is almost 100% but on validation data, it is less than the best we have.

3.4 Conclusion

In Decision Tree classifier, we noticed that all the dimensionality reduction techniques give almost same hyperparameter values which give best the accuracy unlike the Kernel SVM classifier. Also, the classifier starts to overfit if we increase the max depth of the tree from 6 to 12 in all the cases.

	Dimensions	**Criteria**	**Max Depth**	**Validation Set Accuracy**	**Test Set Accuracy**
PCA	40	gini	6	0.2785	0.2755
LDA	4	entropy	6	0.209	0.2147
KPCA	32	entropy	6	0.28	0.2624

4 Multi-layer Perceptron(MLP) Classifier

Multi-layer Perceptron (MLP) is a supervised learning algorithm that learns a function $f(\cdot) : R^m \rightarrow R^o$ by training on a dataset, where m is the number of dimensions for input and o is the number of dimensions for output. It can learn non-linear models for either classification or regression and is different from logistic regression, in that between the input and the output layer, there can be one or more non-linear layers, called hidden layers. But MLP is sensitive to feature scaling and MLP with hidden layers have a non-convex loss function where there exists more than one local minimum.

Some of the parameters in MLP Classifier include hidden_layer_sizes, activation, solver, learning_rate, max_iter etc. Though there are many hyperparameters in sklearn library function but I varied only 4 hyperparameters solver, learning rate, max_iter and hidden_layer_sizes. criteria and max depth for this project. The default values of these hyperparameters is "adam", "constant", "200" and (100,) respectively.

4.1 Dimensionality Reduction using PCA

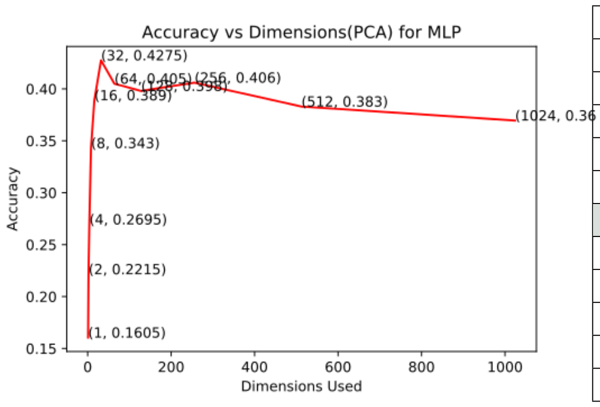

Figure 7: MLP PCA

	Dimensions	Accuracy
1	1	0.1605
2	2	0.2215
3	4	0.2695
4	8	0.343
5	16	0.389
6	32	0.4275
7	64	0.405
8	128	0.398
9	256	0.406
10	512	0.383
11	1024	0.3695

Table 13: Number of components taken vs Accuracy(PCA)

We can see from Figure 7 and Table 13 that the optimal accuracy for PCA dimensionality reduction is when the number of dimensions are 32 and after that as we increase the number of dimensions, accuracy on test set starts decreasing.

Table 14 shows that the best accuracy on test data is obtained when *Solver = "adam", Learning Rate = adaptive, Max iterations = 200, Hidden Layer Size = (100,20).* If we look at the row 3 of the table, it has a little bit more accuracy than row 19(which we chose as the optimal) but if we look at the corresponding train accuracy, row 2 has 4% more accuracy than row 19 which means model may have slight overfit in that case.

Another interesting thing in Table 14 is if we look at rows 5, 11, 17 and 23, we notice that the classifier made using these hyperparameters has clearly overfit the train data(as accuracy on train data is more than 98% in all the cases and accuracy on test data is lower than our optimal accuracy). All these rows have max iterations hyperparameter set to 500. This means that classifier has overfit because we are performing a large number of iterations for convergence. If keep all other hyperparameters same and just change the max iteration hyperparameter to 200 then, we can see that model is performing better on test data and has not quite overfit.

	Solver	Learning Rate	Max iter	Hidden Layer Size	Test Accuracy	Train Accuracy
1	lbfgs	constant	200	(100, 20)	0.3995	0.700625
2	lbfgs	constant	200	(100, 50)	0.3735	0.84725
3	lbfgs	constant	200	(100, 50, 10)	0.4085	0.76125
4	lbfgs	constant	500	(100, 20)	0.3895	0.76225
5	lbfgs	constant	500	(100, 50)	0.3525	0.989375
6	lbfgs	constant	500	(100, 50, 10)	0.3655	0.92475
7	lbfgs	adaptive	200	(100, 20)	0.409	0.66325
8	lbfgs	adaptive	200	(100, 50)	0.378	0.8655
9	lbfgs	adaptive	200	(100, 50, 10)	0.3845	0.759125
10	lbfgs	adaptive	500	(100, 20)	0.377	0.750875
11	lbfgs	adaptive	500	(100, 50)	0.3585	0.998
12	lbfgs	adaptive	500	(100, 50, 10)	0.3555	0.872
13	adam	constant	200	(100, 20)	0.4	0.726
14	adam	constant	200	(100, 50)	0.378	0.877125
15	adam	constant	200	(100, 50, 10)	0.3785	0.866875
16	adam	constant	500	(100, 20)	0.3805	0.789
17	adam	constant	500	(100, 50)	0.3785	0.964375
18	adam	constant	500	(100, 50, 10)	0.362	0.9035
19	adam	adaptive	200	(100, 20)	0.4075	0.721875
20	adam	adaptive	200	(100, 50)	0.3725	0.87875
21	adam	adaptive	200	(100, 50, 10)	0.364	0.848125
22	adam	adaptive	500	(100, 20)	0.39	0.752625
23	adam	adaptive	500	(100, 50)	0.36	0.979625
24	adam	adaptive	500	(100, 50, 10)	0.3655	0.92325

Table 14: Variation of validation test accuracy and train accuracy for different hyperparameters(PCA)

4.2 Dimensionality Reduction using LDA

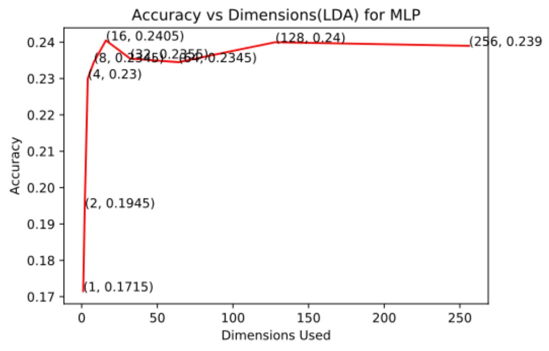

Figure 8: MLP LDA

	Dimensions	Accuracy
1	1	0.1715
2	2	0.1945
3	4	0.23
4	8	0.2345
5	16	0.2405
6	32	0.2355
7	64	0.2345
8	128	0.24
9	256	0.239

Table 15: Number of components taken vs Accuracy(LDA)

We can see from Figure 8 and Table 15 that the optimal accuracy for LDA dimensionality reduction is when the number of dimensions are 9. This is because dimension of LDA is also limited by number of classes and accuracy doesn't vary much after dimensions are more than 9.

In Table 16, though the best accuracy is shown by row 9 but if we look at the train accuracy then it is pretty higher than the other rows which have been highlighted which means the model might have been overfit. I think the best suitable values for hyperparameters in this case would be *Solver = "adam", Learning Rate = adaptive, Max iterations = 200, Hidden Layer Size = (100,20)* or *Solver = "adam", Learning Rate = constant, Max iterations = 200, Hidden Layer Size = (100,20)* as both of them have same test accuracy and much lower train accuracy.

	Solver	Learning Rate	Max iter	Hidden Layer Size	Test Accuracy	Train Accuracy
1	lbfgs	constant	200	(100, 20)	0.2505	0.916875
2	lbfgs	constant	200	(100, 50)	0.239	0.950125
3	lbfgs	constant	200	(100, 50, 10)	0.253	0.917875
4	lbfgs	constant	500	(100, 20)	0.241	0.969625
5	lbfgs	constant	500	(100, 50)	0.242	1
6	lbfgs	constant	500	(100, 50, 10)	0.2335	1
7	lbfgs	adaptive	200	(100, 20)	0.245	0.9195
8	lbfgs	adaptive	200	(100, 50)	0.2475	0.96325
9	lbfgs	adaptive	200	(100, 50, 10)	0.2625	0.926625
10	lbfgs	adaptive	500	(100, 20)	0.2515	0.96075
11	lbfgs	adaptive	500	(100, 50)	0.247	1
12	lbfgs	adaptive	500	(100, 50, 10)	0.2405	0.998125
13	adam	constant	200	(100, 20)	0.2615	0.88675
14	adam	constant	200	(100, 50)	0.258	0.909125
15	adam	constant	200	(100, 50, 10)	0.256	0.910875
16	adam	constant	500	(100, 20)	0.253	0.9175
17	adam	constant	500	(100, 50)	0.2465	0.9695
18	adam	constant	500	(100, 50, 10)	0.241	0.983125
19	adam	adaptive	200	(100, 20)	0.2615	0.88375
20	adam	adaptive	200	(100, 50)	0.2545	0.907625
21	adam	adaptive	200	(100, 50, 10)	0.2615	0.914875
22	adam	adaptive	500	(100, 20)	0.2515	0.924125
23	adam	adaptive	500	(100, 50)	0.249	0.9695
24	adam	adaptive	500	(100, 50, 10)	0.248	0.9705

Table 16: Variation of validation test accuracy and train accuracy for different hyperparameters(LDA)

4.3 Dimensionality Reduction using KPCA

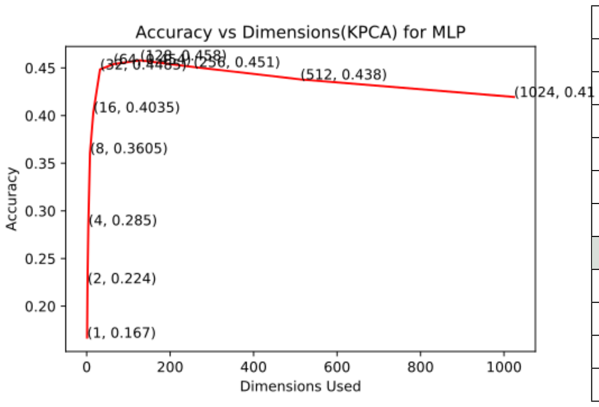

Figure 9: MLP KPCA

	Dimensions	Accuracy
1	1	0.167
2	2	0.224
3	4	0.285
4	8	0.3605
5	16	0.4035
6	32	0.4485
7	64	0.454
8	128	0.458
9	256	0.451
10	512	0.438
11	1024	0.4195

Table 17: Number of components taken vs Accuracy(KPCA)

We can see from Figure 9 and Table 17 that the optimal accuracy for KPCA dimensionality reduction is when the number of dimensions are 64. Accuracy hasn't changed much when we increased the dimension from 64 to 128 and then has decreased on increasing dimensions.

Table 18 shows that the best accuracy on test data is obtained when values of hyperparameters are as of row 13 with *Solver = "adam", Learning Rate = constant, Max iterations = 200, Hidden Layer Size = (100,20).* The interesting to see is that the hyperparameter values in row 2 which were causing overfit in case of PCA and LDA(linear dimensionality reduction techniques) gave very good accuracy for KPCA(non-linear dimensionality reduction technique). Hence, the set of hyperparameter values which give poor accuracy in case of linear dimension reduction techniques may give very good accuracy in non-linear dimension reduction techniques.

	Solver	Learning Rate	Max iter	Hidden Layer Size	Test Accuracy	Train Accuracy
1	lbfgs	constant	200	(100, 20)	0.4175	0.48775
2	lbfgs	constant	200	(100, 50)	0.4455	0.50375
3	lbfgs	constant	200	(100, 50, 10)	0.3935	0.44525
4	lbfgs	constant	500	(100, 20)	0.4105	0.617625
5	lbfgs	constant	500	(100, 50)	0.418	0.709875
6	lbfgs	constant	500	(100, 50, 10)	0.4165	0.574
7	lbfgs	adaptive	200	(100, 20)	0.4185	0.470875
8	lbfgs	adaptive	200	(100, 50)	0.4295	0.507
9	lbfgs	adaptive	200	(100, 50, 10)	0.3945	0.465125
10	lbfgs	adaptive	500	(100, 20)	0.418	0.607375
11	lbfgs	adaptive	500	(100, 50)	0.4245	0.62725
12	lbfgs	adaptive	500	(100, 50, 10)	0.3995	0.54
13	adam	constant	200	(100, 20)	0.453	0.6585
14	adam	constant	200	(100, 50)	0.4505	0.745
15	adam	constant	200	(100, 50, 10)	0.415	0.73225
16	adam	constant	500	(100, 20)	0.43	0.812125
17	adam	constant	500	(100, 50)	0.404	0.88675
18	adam	constant	500	(100, 50, 10)	0.391	0.8325
19	adam	adaptive	200	(100, 20)	0.4435	0.665875
20	adam	adaptive	200	(100, 50)	0.4415	0.7295
21	adam	adaptive	200	(100, 50, 10)	0.4075	0.685375
22	adam	adaptive	500	(100, 20)	0.4195	0.7835
23	adam	adaptive	500	(100, 50)	0.404	0.876125
24	adam	adaptive	500	(100, 50, 10)	0.388	0.88975

Table 18: Variation of validation test accuracy and train accuracy for different hyperparameters(KPCA)

4.4 Conclusion

In MLP classifier, we noticed that all the three dimentionality reduction techniques gave maximum accuracy for almost same set of hyperparameter of the classifier but the optimal dimension in which data should be reduced differed a lot in all the three techniques. LDA almost gives same accuracy in both ”constant” and ”adaptive” Learning Rates. Also we noticed that a set of hyperparameter values may give poor accuracy for linear dimensionality reduction techniques but same set of values may give really good results for non-linear dimensionality reduction techniques.

	Dimensions	Solver	Learning Rate	Max iter	HL Size	Val Set Accuracy	Test Set Accuracy
PCA	32	adam	constant	200	(100,20)	0.4145	0.4083
LDA	9	adam	constant	200	(100,20)	0.263	0.2354
LDA	9	adam	adaptive	200	(100,20)	0.2605	0.2357
KPCA	70	adam	adaptive	200	(100,20)	0.435	0.4512

5 Logistic Regression Classifier

Logistic Regression Classifier uses one-vs-rest (OvR) scheme if the ’multi_class’ option is set to ’ovr’, and uses the cross- entropy loss if the ’multi_class’ option is set to ’multinomial’. By default it uses ’ovr’.

Some of the parameters in Logistic Regression classifier include solver, penality, max iterations, C, class weight, multi_class. Though there are many hyperparameters in sklearn library function but I varied only 4 hyperparameters solver, penality, max iterations, C(Inverse of regularization strength, smaller values specify strong regularization strength). The default values of these parameters are ”liblinear”, ”l2”, 100, 1 respectively.

5.1 Dimensionality Reduction using PCA

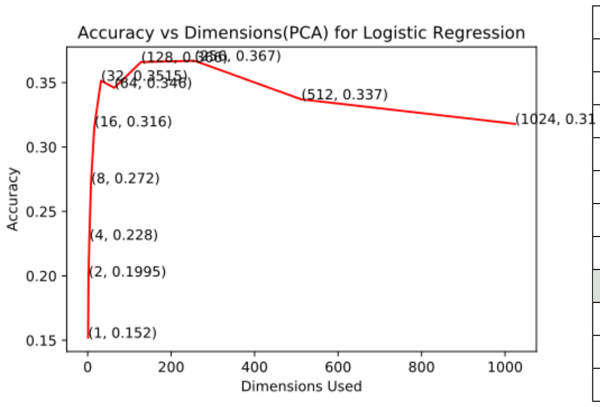

Figure 10: Logistic Regression with PCA

	Dimensions	**Accuracy**
1	1	0.152
2	2	0.2
3	4	0.228
4	8	0.272
5	16	0.316
6	32	0.352
7	64	0.346
8	128	0.366
9	256	0.367
10	512	0.337
11	1024	0.318

Table 19: Number of components taken vs Accuracy(PCA)

We can see from Figure 10 and Table 19 that the optimal accuracy for PCA dimensionality reduction is when the number of dimensions are 128. There isn't much difference in accuracy between dimensions 128 and 256, but after that as we increase the number of dimensions, accuracy on test set starts decreasing.

	Solver	**Penality**	**Max iterations**	**C**	**Test Accuracy**	**Train Accuracy**
1	liblinear	l1	200	0.5	0.3785	0.429375
2	liblinear	l1	500	0.5	0.3785	0.429375
3	liblinear	l1	200	1	0.3785	0.429375
4	liblinear	l1	500	1	0.3785	0.429375
5	liblinear	l1	200	2.5	0.3785	0.429375
6	liblinear	l1	500	2.5	0.3785	0.429375
7	liblinear	l2	200	0.5	0.375	0.42875
8	liblinear	l2	500	0.5	0.375	0.42875
9	liblinear	l2	200	1	0.375	0.42875
10	liblinear	l2	500	1	0.375	0.42875
11	liblinear	l2	200	2.5	0.375	0.42875
12	liblinear	l2	500	2.5	0.375	0.42875
13	lbfgs	l2	200	0.5	0.3715	0.43375
14	lbfgs	l2	500	0.5	0.3715	0.43375
15	lbfgs	l2	200	1	0.3715	0.43375
16	lbfgs	l2	500	1	0.3715	0.43375
17	lbfgs	l2	200	2.5	0.3715	0.43375
18	lbfgs	l2	500	2.5	0.3715	0.43375

Table 20: Variation of validation test accuracy and train accuracy for different hyperparameters(PCA)

In Table 20, test and train accuracy has been calculated by varying some hyperparameters. We notice that all the rows of table from 1-6, 7-12, 13-18 have the same test and train accuracy. i.e if we fix the solver and penality, there isn't any effect of other hyperparameters, which means that model would have converged before reaching 200 iterations. I only varied C only from 0.5 to 2.5, may be if we vary C by some large values then accuracy may change. I picked up *Solver = "liblinear", Penality = "l1", Max iteration = 200, C = 1* as the best hyperparameter values.

5.2 Dimensionality Reduction using LDA

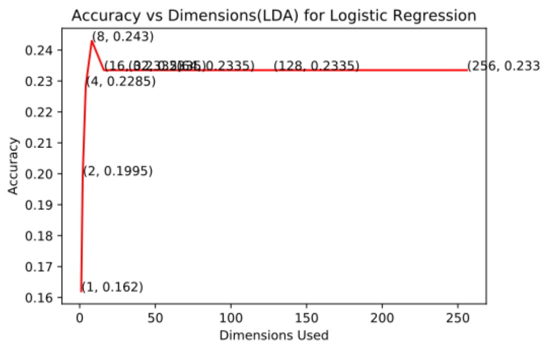

Figure 11: Logistic Regression with LDA

	Dimensions	Accuracy
1	1	0.162
2	2	0.2
3	4	0.228
4	8	0.243
5	16	0.234
6	32	0.234
7	64	0.234
8	128	0.234
9	256	0.234

Table 21: Number of components taken vs Accuracy(LDA)

We can see from Figure 11 and Table 21 that the optimal accuracy for LDA dimensionality reduction is when the number of dimensions are 8. There's a peak at 8 but as we increase the number of dimensions, accuracy on test set decreases a bit from 8 to 16 and remains constant after that.

	Solver	Penality	Max iterations	C	Test Accuracy	Train Accuracy
1	liblinear	l1	200	0.5	0.2245	0.824625
2	liblinear	l1	500	0.5	0.2245	0.824625
3	liblinear	l1	200	1	0.2245	0.824625
4	liblinear	l1	500	1	0.2245	0.824625
5	liblinear	l1	200	2.5	0.2245	0.824625
6	liblinear	l1	500	2.5	0.2245	0.824625
7	liblinear	l2	200	0.5	0.2245	0.825125
8	liblinear	l2	500	0.5	0.2245	0.825125
9	liblinear	l2	200	1	0.2245	0.825125
10	liblinear	l2	500	1	0.2245	0.825125
11	liblinear	l2	200	2.5	0.2245	0.825125
12	liblinear	l2	500	2.5	0.2245	0.825125
13	lbfgs	l2	200	0.5	0.229	0.827
14	lbfgs	l2	500	0.5	0.229	0.827
15	lbfgs	l2	200	1	0.229	0.827
16	lbfgs	l2	500	1	0.229	0.827
17	lbfgs	l2	200	2.5	0.229	0.827
18	lbfgs	l2	500	2.5	0.229	0.827

Table 22: Variation of validation test accuracy and train accuracy for different hyperparameters(LDA)

In Table 22, test and train accuracy has been calculated by varying some hyperparameters. We notice that all the rows of table from 1-12, 13-18 have the same test and train accuracy. Also both sets of rows have almost same accuracies. I picked up *Solver = "liblinear", Penality = "l1", Max iteration = 200, C = 1* as the best hyperparameter values.

5.3 Dimensionality Reduction using KPCA

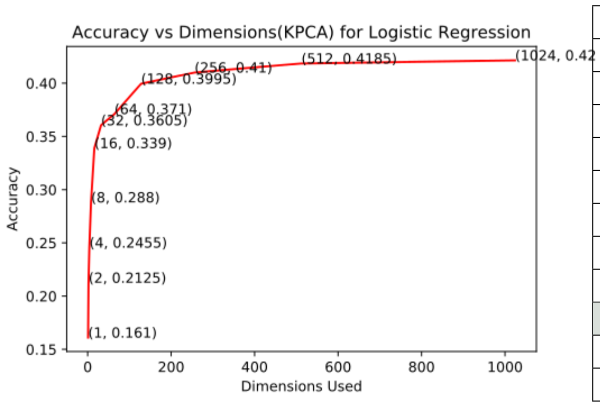

Figure 12: Logistic Regression KPCA

	Dimensions	Accuracy
1	1	0.161
2	2	0.2125
3	4	0.2455
4	8	0.288
5	16	0.339
6	32	0.3605
7	64	0.371
8	128	0.3995
9	256	0.41
10	512	0.4185
11	1024	0.4215

Table 23: Number of components taken vs Accuracy(KPCA)

We can see from Figure 12 and Table 23 that the optimal accuracy for KPCA dimensionality reduction is when the number of dimensions are 1024. But the accuracy hasn't changed much when we increased the dimension from 256. So we can use 256 as the optimal dimension.

	Solver	Penality	Max iterations	C	Test Accuracy	Train Accuracy
1	liblinear	l1	200	0.5	0.4285	0.440625
2	liblinear	l1	500	0.5	0.4285	0.440625
3	liblinear	l1	200	1	0.4285	0.440625
4	liblinear	l1	500	1	0.4285	0.44075
5	liblinear	l1	200	2.5	0.4285	0.440625
6	liblinear	l1	500	2.5	0.4285	0.440625
7	liblinear	l2	200	0.5	0.4195	0.44575
8	liblinear	l2	500	0.5	0.4195	0.44575
9	liblinear	l2	200	1	0.4195	0.44575
10	liblinear	l2	500	1	0.4195	0.44575
11	liblinear	l2	200	2.5	0.4195	0.44575
12	liblinear	l2	500	2.5	0.4195	0.44575
13	lbfgs	l2	200	0.5	0.424	0.451125
14	lbfgs	l2	500	0.5	0.424	0.451125
15	lbfgs	l2	200	1	0.424	0.451125
16	lbfgs	l2	500	1	0.424	0.451125
17	lbfgs	l2	200	2.5	0.424	0.451125
18	lbfgs	l2	500	2.5	0.424	0.451125

Table 24: Variation of validation test accuracy and train accuracy for different hyperparameters(KPCA)

In Table 24, test and train accuracy has been calculated by varying some hyperparameters. The situation is same as it was in PCA and LDA, so I picked up *Solver = "liblinear", Penality = "l1", Max iteration = 200, C = 1* as the best hyperparameter values.

5.4 Conclusion

In Logistic Regression classifier, we noticed that all the three dimensionality reduction techniques gave maximum accuracy for almost same set of hyperparameter of the classifier.

	Dimensions	Solver	Penality	Max iter	C	Val Set Accuracy	Test Set Accuracy
PCA	128	liblinear	l1	200	1	0.3785	0.3781
LDA	8	liblinear	l1	200	1	0.2245	0.2294
KPCA	256	liblinear	l1	200	1	0.4285	0.4128

6 Final Results

	Classifier	**Dim Reduction**	**Test Set Accuracy**	**Test Set F1 score**
1	Kernel SVM	PCA	0.4678	0.4678
2	Kernel SVM	LDA	0.2374	0.2374
3	Kernel SVM	KPCA	0.4556	0.4556
4	Decision Tree	PCA	0.2755	0.2755
5	Decision Tree	LDA	0.2147	0.2147
6	Decision Tree	KPCA	0.2624	0.2624
7	MLP	PCA	0.4083	0.4083
8	MLP	LDA	0.2357	0.2357
9	MLP	KPCA	0.4512	0.4512
10	Logistic Regression	PCA	0.3781	0.3781
11	Logistic Regression	LDA	0.2294	0.2294
12	Logistic Regression	KPCA	0.4128	0.4128

After analyzing 4 different classifiers and 3 different dimensionality reduction techniques, we can to conclusion that Kernel SVM with PCA provides the best results and accuracy of Kernel SVM with KPCA is just a little less than it if we supply the tuned hyperparameter values.

In **Kernel SVM classifier**, we analyzed that higher we keep the value of γ, the more is the chance of classifier getting overfit in case of linear dimensionality reduction techniques but if we do the same-thing for non-linear dimensionality reduction, then we see that the accuracy actually increases rather classifier getting overfit.

In **Decision Tree classifier**, we analyzed that early stopping(by reducing the depth of the tree), helps us to avoid overfitting of the model. As if we explore the full tree, classifier almost giving 100% accuracy on the train set.

In **MLP classifier**, there were some rows in the table which correspond to almost 100% accuracy on train data and the common thing in all of them is that the value max iteration which is quite large, due to which classifier starts getting overfit. But, the interesting observation was that there was much less overfitting in case of KPCA as compared to PCA and LDA.

In **Logistic Regression classifier**, dimensionality reduction techniques gave maximum accuracy for almost same set of hyperparameter values. But the classifier did not overfit even if we increased the max iterations to be performed.

So, if we compare all the classifiers, Logistic Regression has the least chance to get overfit on training data even if we increase the max number of iterations to be performed. Kernel SVM is really good classifier, but the value of γ and C should be carefully selected in order to achieve really good accuracy. MLP also performs great if we use KPCA reduction. In every classifier KPCA's accuracy is either more than that of PCA or almost equal to it. LDA has least accuracy in all the 3 as in LDA the number of output components in general is capped at $NumberofClasses - 1$ which is 9 in this case. Hence, in general performance looks like **KPCA** $>$ **PCA** $>$ **LDA**. Since, the data is not linearly separable, introducing a non linearity in dimensionality reduction probably improves performance and also helps to avoid overfitting.