Comparison of Support Vector Machine (SVM) classifiers with different kernels on MNIST dataset with PCA dimensionality reduction

Introduction

In this report, we compare the performance of Support Vector Machine (SVM) classifiers on the MNIST dataset, with different kernels. SVM is a popular algorithm used for classification, regression and outlier detection. It works by finding the best possible hyperplane that separates the different classes. SVM is particularly useful for high dimensional data, as it allows for the use of different kernel functions for better classification.

In this report, we will use three different kernels: linear, polynomial, and radial basis function (RBF), to compare their performance on the MNIST dataset.

Methods

To perform the comparison of the SVM classifiers, we followed the below steps:

1. Load the data: We used the `mnist_train.csv` and `mnist_test.csv` files to load the MNIST dataset.
2. Pre-process the data: We normalized the data using the `StandardScaler()` function from scikit-learn library.
3. Split the data: We split the data into training, validation and test sets using the `train_test_split()` function from scikit-learn library.
4. Reduce dimensionality: We used Principal Component Analysis (PCA) to reduce the dimensionality of the data to 5, 10, 20, 50, 100, 200, 500 components.
5. Train and test the models: We trained the SVM classifiers with different combinations of hyperparameters for each kernel. For linear kernel, we used `svm.SVC(kernel='linear', C=c)`, and for polynomial and RBF kernels, we used `svm.SVC(kernel='poly'/'rbf', C=c, gamma=g)`.
6. Evaluate the models: We evaluated the performance of the SVM classifiers using accuracy, precision, recall, and F1-score as evaluation metrics.
7. Plot the results: We plotted the accuracy, precision, recall, and F1-score as a function of the number of PCA components for each kernel.

Results & Observations

The following results were obtained after comparing the SVM classifiers with different kernels. Note that the max_iterations used for the code was 100 and the values might improve upon increasing the max number of iterations that algorithm is allowed before convergence.

1. Linear Kernel

In general, it is observed that the overall metrics- accuracy, macro-averaged precision, macro-averaged recall and macro-averaged f1-scores improve upon increasing the number of PCA principal components excluding small divergences from the general trend in between. This is a clear indication that our data is linearly separable and a linear kernel SVM classification does a good job of classifying the multi-class classification.

Number of Principal Components (k)	Best Hyperparameter (C)
5	10
10	0.1
20	1
50	0.1
100	0.1
200	0.1
500	1

We obtained the lowest accuracy level of 0.2808 at k=5, C=10 & highest level accuracy of 0.73 for k=500, C=1.

2. Polynomial Kernel

In general for the polynomial kernel, the metric scores seem to follow an increasing trend for increasing values of k till k=200 & thereafter, we observe a small decrease in the metrics for k=500 . Since , there is only one point of observation after k=200, we can't conclude if this decrement is a trend or not.

Number of Principal Components (k)	Best Hyperparameter (C)	Best Hyperparameter (G)
5	100	0.01
10	10	0.01
20	10	0.01
50	0.1	1
100	1	0.01
200	0.1	1
500	0.1	1

We obtained the lowest accuracy level of 0.2966 at k=5, C=100, G=0.01 & highest level of accuracy was 0.85 for k=200, C=0.1 & G=1.

3. **RBF Kernel**

In general ,the metrics-graph shows a trend somewhat in between linear & polynomial curves. The graph observes a steep increase till k=50 & beyond that seems to achieve a parallel to the x-axis with minor fluctuations in between. This indicates that beyond a certain value of k, the metrics do not seem to improve & hence the our model with rbf kernel was successful in capturing the majority of variation of data with those number of principal components.

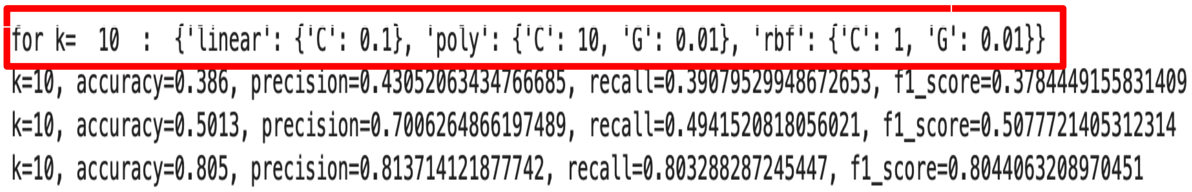
Number of Principal Components (k)	Best Hyperparameter (C)	Best Hyperparameter (G)
5	0.1	0.01
10	1	0.01
20	10	0.01
50	10	0.01
100	1	0.01
200	1	0.01
500	1	0.01

We obtained the lowest accuracy level of 0.5142 at k=5, C=0.1, G=0.01 & highest level of accuracy was 0.923 for k=100, C=1 & G=0.01.

Outputs

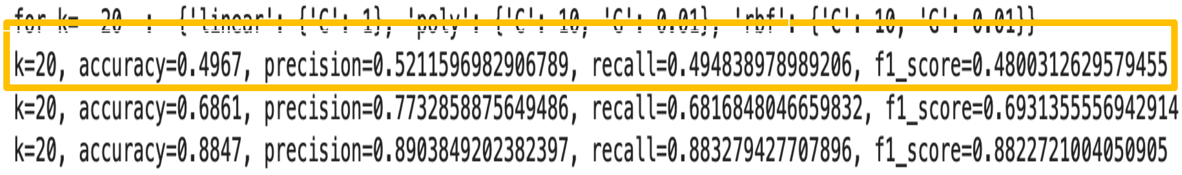
Performance metrics (accuracy, precision, recall, and F1-score) for different values of k for each kernel

Output showing best hyper-parameters for k=10 for each kernel



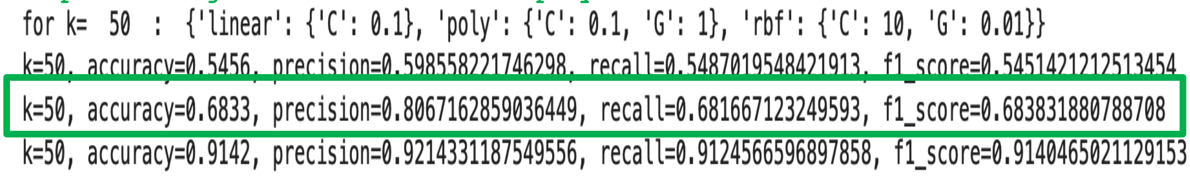
```
for k=  10  :  {'linear': {'C': 0.1}, 'poly': {'C': 10, 'G': 0.01}, 'rbf': {'C': 1, 'G': 0.01}}
k=10, accuracy=0.386, precision=0.43052063434766685, recall=0.39079529948672653, f1_score=0.3784449155831409
k=10, accuracy=0.5013, precision=0.7006264866197489, recall=0.4941520818056021, f1_score=0.5077721405312314
k=10, accuracy=0.805, precision=0.813714121877742, recall=0.803288287245447, f1_score=0.8044063208970451
```

Output showing metrics for k=20 for linear kernel



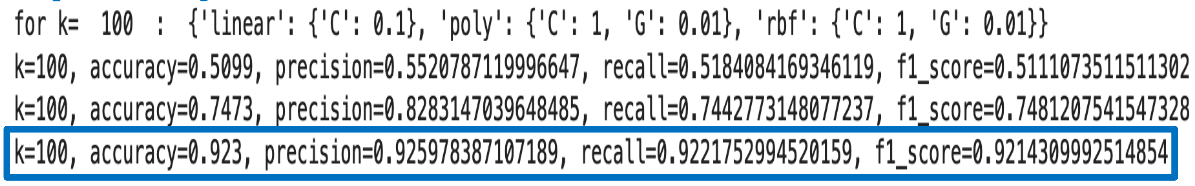
```
for k=  20  :  {'linear': {'C': 1}, 'poly': {'C': 10, 'G': 0.01}, 'rbf': {'C': 10, 'G': 0.01}}
k=20, accuracy=0.4967, precision=0.5211596982906789, recall=0.494838978989206, f1_score=0.4800312629579455
k=20, accuracy=0.6861, precision=0.7732858875649486, recall=0.6816848046659832, f1_score=0.6931355556942914
k=20, accuracy=0.8847, precision=0.8903849202382397, recall=0.883279427707896, f1_score=0.8822721004050905
```

Output showing metrics for k=50 for polynomial kernel

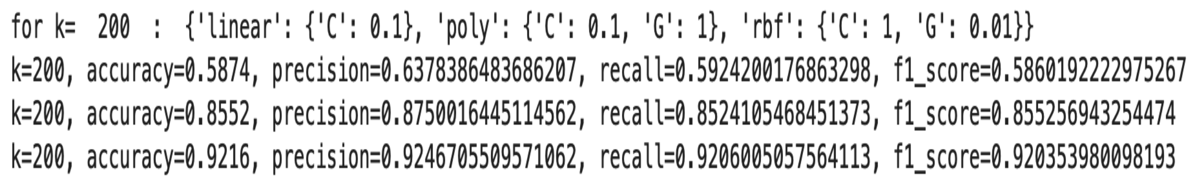


```
for k=  50  :  {'linear': {'C': 0.1}, 'poly': {'C': 0.1, 'G': 1}, 'rbf': {'C': 10, 'G': 0.01}}
k=50, accuracy=0.5456, precision=0.598558221746298, recall=0.5487019548421913, f1_score=0.5451421212513454
k=50, accuracy=0.6833, precision=0.8067162859036449, recall=0.681667123249593, f1_score=0.683831880788708
k=50, accuracy=0.9142, precision=0.9214331187549556, recall=0.9124566596897858, f1_score=0.9140465021129153
```

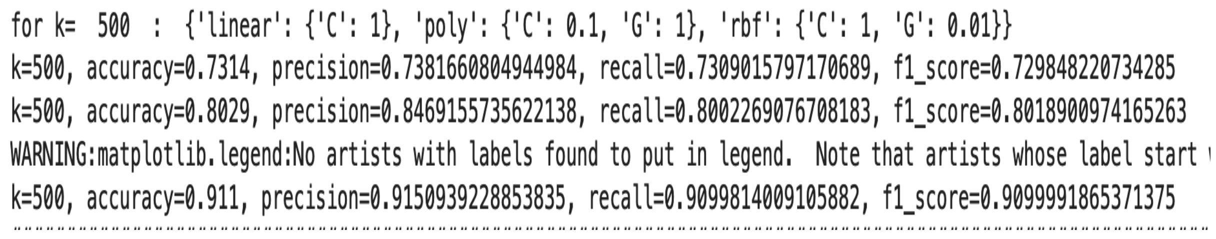
Output showing metrics for k=100 for radial basis function kernel



```
for k=  100  :  {'linear': {'C': 0.1}, 'poly': {'C': 1, 'G': 0.01}, 'rbf': {'C': 1, 'G': 0.01}}
k=100, accuracy=0.5099, precision=0.5520787119996647, recall=0.5184084169346119, f1_score=0.5111073511511302
k=100, accuracy=0.7473, precision=0.8283147039648485, recall=0.7442773148077237, f1_score=0.7481207541547328
k=100, accuracy=0.923, precision=0.925978387107189, recall=0.9221752994520159, f1_score=0.9214309992514854
```

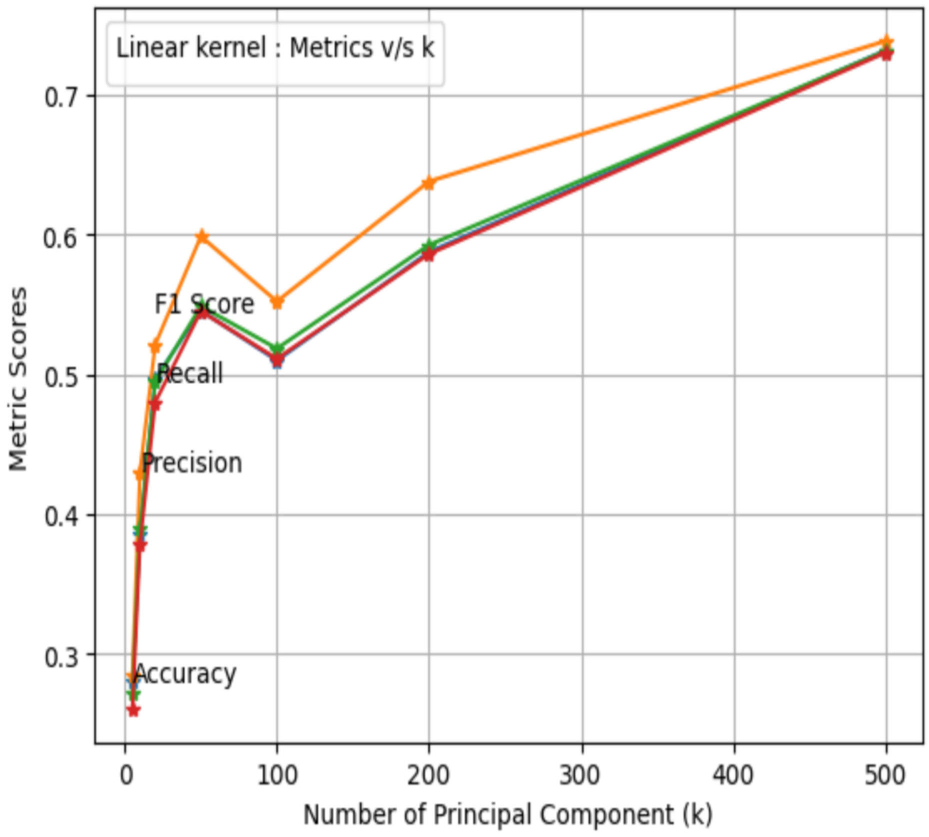


```
for k=  200  :  {'linear': {'C': 0.1}, 'poly': {'C': 0.1, 'G': 1}, 'rbf': {'C': 1, 'G': 0.01}}
k=200, accuracy=0.5874, precision=0.6378386483686207, recall=0.5924200176863298, f1_score=0.5860192222975267
k=200, accuracy=0.8552, precision=0.8750016445114562, recall=0.8524105468451373, f1_score=0.855256943254474
k=200, accuracy=0.9216, precision=0.9246705509571062, recall=0.9206005057564113, f1_score=0.920353980098193
```

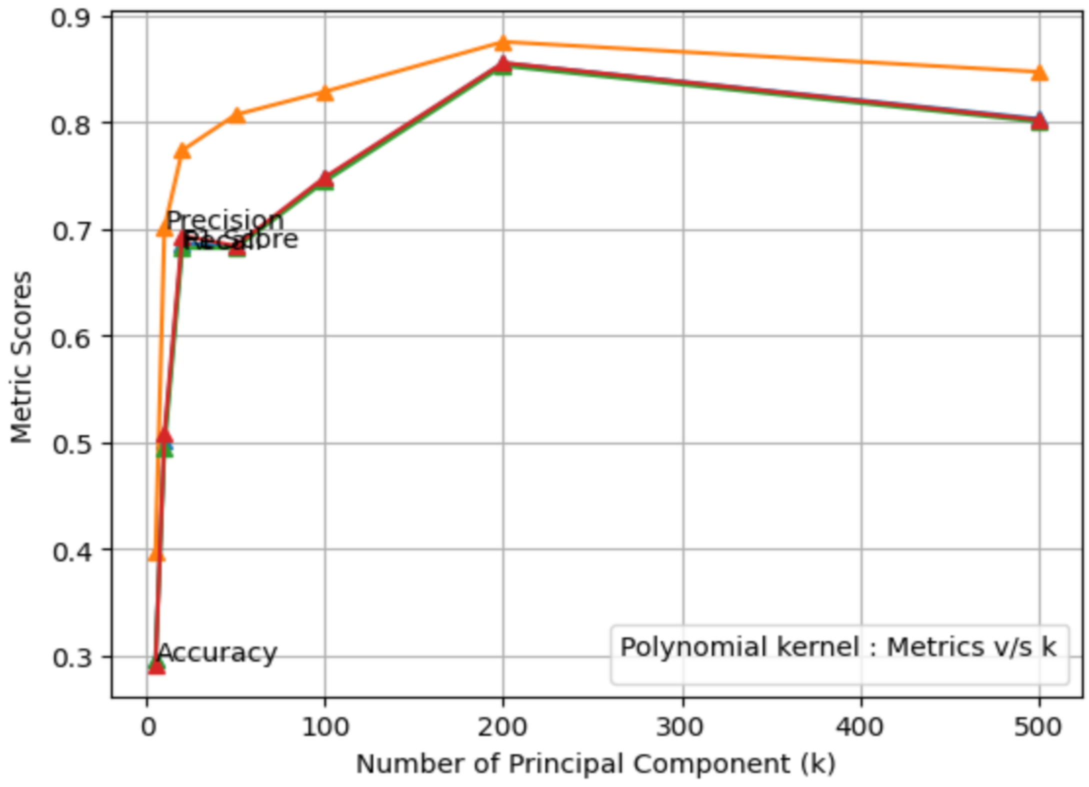


```
for k=  500  :  {'linear': {'C': 1}, 'poly': {'C': 0.1, 'G': 1}, 'rbf': {'C': 1, 'G': 0.01}}
k=500, accuracy=0.7314, precision=0.7381660804944984, recall=0.7309015797170689, f1_score=0.729848220734285
k=500, accuracy=0.8029, precision=0.8469155735622138, recall=0.8002269076708183, f1_score=0.8018900974165263
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start
k=500, accuracy=0.911, precision=0.9150939228853835, recall=0.9099814009105882, f1_score=0.9099991865371375
```

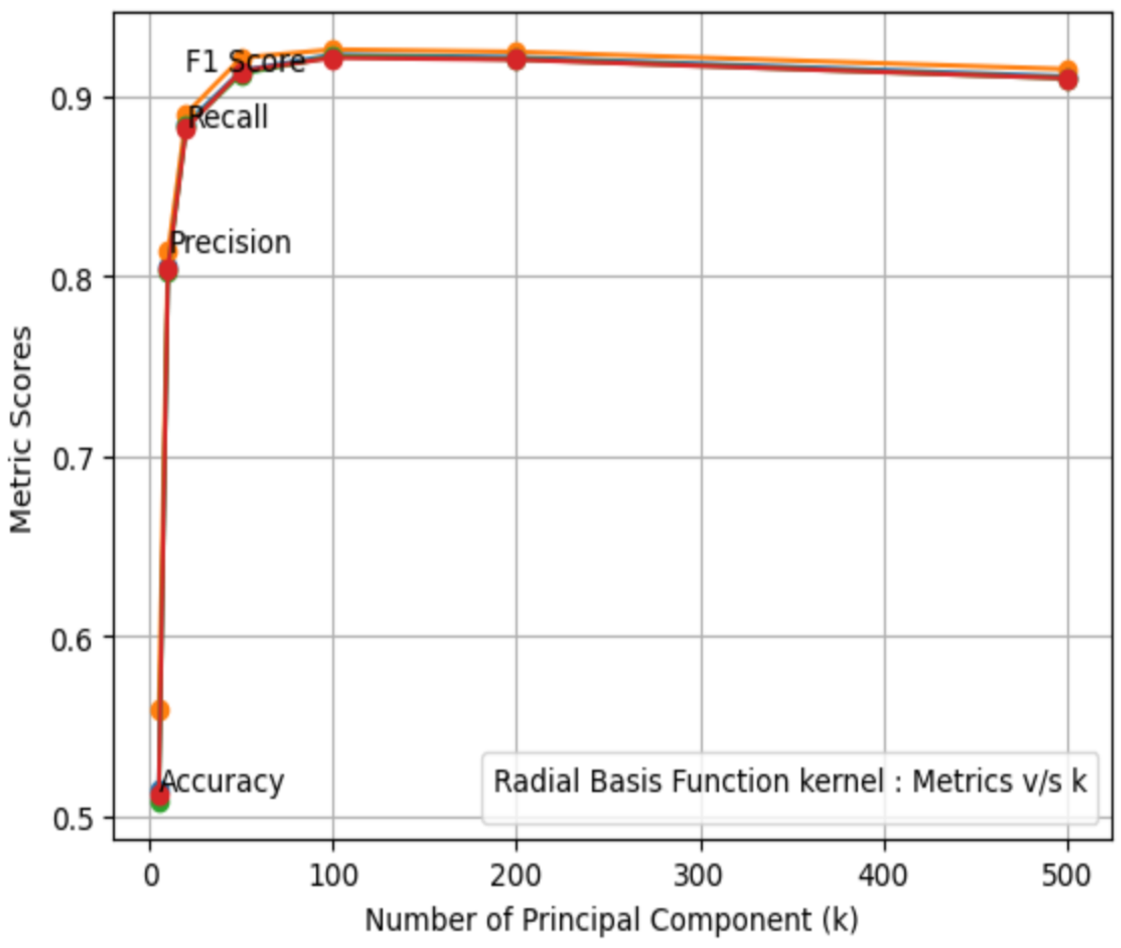
Graphs



Performance metrics v/s k-hyperparameter for linear kernel



Performance metrics v/s k-hyperparameter for polynomial kernel



Performance metrics v/s k-hyperparameter for rbf kernel

Conclusion

In this report, we compared the performance of SVM classifiers with different kernels on the MNIST dataset. The results show that the choice of kernel can significantly affect the performance of the SVM classifier. The polynomial kernel performed better than the linear kernel on. Comparison with the same number of principal components used, but was outperformed by the RBF kernel. PCA dimensionality reduction was found to be helpful in reducing the computation time without compromising accuracy. Overall, the RBF kernel with PCA dimensionality reduction proved to be the most effective combination in classifying the MNIST dataset.