E0-270 Machine Learning - Assignment 1

Submitted by: Shingala Jaydeep Jaysukhbhai (SR: 21076)
Department of Computer Science and Automation, IISc, Bangalore.
jaydeeps@iisc.ac.in

1 Problem Statement

Implementation of Principal Component Analysis and Support Vector Machines on MNIST Dataset for multiclass classification from scratch using python without using any extra in-built libraries.

2 Introduction

MNIST Dataset consists of 60,000 training images and 10,000 test images. Each image is 28 X 28 grayscale image, which is handwritten digits between 0 and 9 inclusive. our job is to correctly classify those images to true class between 0 and 9, inclusive of where it belongs to.

3 Methedology

Here, below given is a complete methodology used to build this models step by step.

3.1 Task 1: Principle Component Analysis

PCA (Principal Component Analysis) is a widely used technique for dimensionality reduction. It can be applied to the MNIST dataset to reduce the number of features and speed up the training process of models. The MNIST dataset contains 28x28 grayscale images of handwritten digits, which are represented as 784-dimensional feature vectors.

PCA aims to find a lower-dimensional representation of the data that captures as much of the variance as possible. To do this, it computes the principal components of the data, which are the directions inthe feature space along which the data varies the most.

The first principal component is the direction that explains the most variance in the data. The second principal component is the direction that explains the most variance after the first principal component is removed from the data, and so on. The number of principal components to keep is a hyperparameter that can be tuned based on the wanted trade-off between dimensionality reduction and information loss. however here as different K values that are a number of principal components to be used are already given, will use them.

First, we need to find mean of the entire training dataset and then subtract that mean from given dataset. This process will convert dataset into zero centered dataset. now we need to find covarience matrix of the transpose of the mean centered dataset. Now we findd top k eigen values of the covarience matrix and corresponding top k eigen vectors.

Now, we multiply dataset with transpose of matrix made from top k eigen vectors. this transforms the current dataspace into k dimentional dataspace, with minimum information loss possible.

$$X = X - Mean(X)$$
$$Covariencematrix = X^T X$$
$$eigenspace = topkeigenvectors(covariencematrix)$$
$$transformedX = X * eigenspace^T$$

We can now train a machine learning model on the transformed data and evaluate its performance on the test data. The reduced-dimensionality data should speed up the training process and lead to better generalization performance.

3.2 Task 2: Multi-class SVM

Step 1: SVM formulation for multiclass classification
SVM is a binary classification algorithm, meaning it can only classify between two classes. To extend SVM to multiclass classification, we use a technique called one-vs-all (OvA) classification. In this technique, we train multiple binary SVM classifiers, each one distinguishing between one class and the rest of the classes.

Here we have 10 classes (digits 0-9) in the MNIST dataset. We will train 10 binary SVM classifiers, each one distinguishing between one digit and the rest of the digits. For example, the first classifierwill distinguish between the digit 0 and the rest of the digits, the second classifier will distinguish between the digit 1 and the rest of the digits, and so on.

Step 2: Training the SVM classifiers For each binary classifier, we will train the SVM using the training data. We will use the default hyperparameters for the SupportVectorModel class, which include the regularization parameter C and the loss function. Here, those hyperparameters are number of iterations, C , and learning rate. Now, as we are using stochastic gradient descent for update rulewe will take any sample at random from entire training dataset in each iteration and predict with current weight and bias b. Let, x be the random sample at t^th iteration then:
if $sign(y * (W^T x + b)) > 0$ then, that means this sample is correctly classified. so there is no need of updating weight and bias.
else we need to update the weight vector and bias based on this misclassification and update rule that i have used is as given below.

$$minimize : 1/2 * ||w||^2 + C * sum((0, 1 - y_i * (w * x_i + b)))$$
$$\text{subject to: } y_i * (w * x_i + b) >= 1 \text{ for all } i$$

where $||w||$ is the Euclidean norm of the weight vector, C is the regularization parameter that balances between the margin maximization and the classification error minimization, y_i is the label of the ith example (-1 or 1), and x_i is the feature vector of the ith iteration.

The first term in the objective function, $1/2 * ||w||^2$, encourages a large margin, while the second term penalizes misclassifications. The constraint ensures that all examples are correctly classified with a margin of at least 1.

Step 3: Making predictions To make predictions for a new test sample, we will pass the test sample through each of the binary classifiers and choose the class with the highest confidence score. The confidence score for a class is the output of the SVM classifier for that class. Specifically, for the ith classifier, the confidence score for class j is given by:

$$score_{i,j} = w_i * x + b_i$$

where w_i is the weight vector learned by the ith classifier, x is the input image vector, and b_i is the bias term learned by the ith classifier. The predicted class for the input image is the class with the highest confidence score across all classifiers.

Step 4: Model evaluation We will evaluate the performance of our multiclass SVM model using the test data. We will compute the accuracy of the model, which is the percentage of test images that are correctly classified.

Accuracy, Precision, Recall, and F1-Score are performance metrics used to evaluate the performance of a classification model.

Accuracy: Accuracy is the percentage of correctly predicted instances out of the total instances. It is a measure of how well the model is able to predict both positive and negative instances.

Formula: Accuracy = (TP + TN) / (TP + TN + FP + FN)

Where:

TP = True positives (instances predicted as positive and actually positive) TN = True negatives (instances predicted as negative and actually negative) FP = False positives (instances predicted as positive but actually negative) FN = False negatives (instances predicted as negative but actually positive) Precision: Precision is the percentage of correctly predicted positive instances out of the total predicted positive instances. It is a measure of how well the model is able to predict positive instances.

Formula: Precision = TP / (TP + FP)

Recall: Recall is the percentage of correctly predicted positive instances out of the total actual positive instances. It is a measure of how well the model is able to identify all positive instances.

Formula: Recall = TP / (TP + FN)

F1-Score: F1-Score is a weighted harmonic mean of precision and recall. It is a measure of the balance between precision and recall.

Formula: F1-Score = 2 * (Precision * Recall) / (Precision + Recall)

All of these metrics range from 0 to 1, where 1 indicates perfect performance and 0 indicates no performance. The choice of which metric to use depends on the problem at hand. For example, if false positives are particularly costly, precision may be the most important metric to optimize. If it's important to identify all positive instances, recall may be the most important metric to optimize.F1-Score is often used when you want to balance both precision and recall.

4 Key Observations

some of the key findings while developing and analysing the model are as follows.

4.1 On hyperparameters

1. In a soft-margin SVM, the parameter 'C' is a regularization hyperparameter that controls the trade-off between maximizing the margin and minimizing the classification error on the training data.

The 'C' parameter determines the degree to which the SVM allows for misclassifications on the training data. A smaller value of C allows for more misclassifications (i.e., a wider margin), while a larger value of C penalizes misclassifications more heavily (i.e., a smaller margin).

In other words, as the value of 'C' increases, the SVM becomes more sensitive to misclassifications, and as a result, it may overfit the training data by creating a complex decision boundary that perfectly separates the training data, but generalizes poorly to new, unseen data. Conversely, if the value of 'C' is too small, the SVM may underfit the training data by creating a simple decision boundary that does not capture the underlying patterns in the data.

Therefore, selecting the appropriate value of 'C' is crucial for achieving good performance on both the training and testing data. Here, after testing model on multiple 'C', i found that keeping 'C' around 10 gives very good performance and higher accuracy.

2. As we are randomly sampling one sample at a time from training dataset, after trying different values of it, i found that around after 2,00,000 iterations the model converges good. This might be becuase we have 60,000 training samples, after this many iterations every sample is randomly taken for almost 2 times into consideration.

3. Learning rate: keeping Learning rate as 0.0001 in this setting is giving good accuracy.

With a higher learning rate, the model can update its parameters more quickly, which can be beneficial for faster convergence and finding a good solution. However, when the learning rate is too high, the model can overshoot the optimal solution and start oscillating around it, or even diverge and increase the loss indefinitely.

This happens because a high learning rate can cause the updates to be too large, and the model may jump across the optimal solution in each iteration. This can lead to instability and poor performance, as the loss function may increase instead of decreasing.

Moreover, with a higher learning rate, the optimization process can become less smooth, which can make it difficult for the algorithm to find the optimal solution. The gradients may fluctuate more, and the update steps may be less consistent, which can lead to slower convergence or getting stuck in suboptimal solutions. In our case with stochastic gradient descent, the comparatively lower learning rate is beneficial.

4.2 For different values of K

Here, the value of K is number of principle components and as they increase we are getting batter and batter accuracy. but after some value of k for any type of setting of hyperparameters the accuracy is not changing notably. even in most of cases after k=100, the accuracy is either saturated that is most tends to constant or decreases but does not increase.

This might be becuase of after a certain number of principal components (K), the remaining components contain very little information about the underlying structure of the data.

After this point, adding more principal components may lead to overfitting or capturing noise in the data, which can lead to a decrease in accuracy. Moreover, the additional components may also introduce more complexity to the model, which can make it harder to generalize to new data.

4.3 Different settings Vs. Accuracy of model

To tune those 3 hyperparameters i tried training model on different settings of parameters and got some insights as below. Number of iterations = 1,50,000
C = 1
Learning rate = 0.001

	5	10	20	50	100	200	500
Accuracy (in percentage)	55	68	76	82	83	82	82
precision	0.63	0.72	0.79	0.83	0.84	0.84	0.86
recall	0.54	0.67	0.75	0.81	0.82	0.81	0.84
f1-score	0.51	0.63	0.74	0.81	0.81	0.81	0.81

One key observation to make here was that as i was decreasing learning rate the model was getting batter learning and hence resulted in higher accuracy. with the same model when i increased learning rate from 0.0001 to 0.1 or 0.5 the model performed woerst ever resulting in around 10 percent accuracy overall.

5 Final Results

After trying different settings, and tuning model hyperparameters, given below is the best accuracy, precision, recall and f1-scores for my trained model.

Number of iterations = 2,00,000
C = 10
Learning rate = 0.0001

	5	10	20	50	100	200	500
Accuracy (in percentage)	52	66	75	84	91	90	90
precision	0.52	0.68	0.74	0.83	0.92	0.89	0.86
recall	0.54	0.65	0.78	0.81	0.94	0.87	0.84
f1-score	0.48	0.61	0.72	0.81	0.90	0.88	0.81

here, column contains different values of K that is principle components and rows contains Different measures of testing. so the highest accuracy that was observed was with k=100 and accuracy was around 85 percentage.

Given below are the different graphs.

5.1 Graph 1: Different Number of principle components K vs. Accuracy

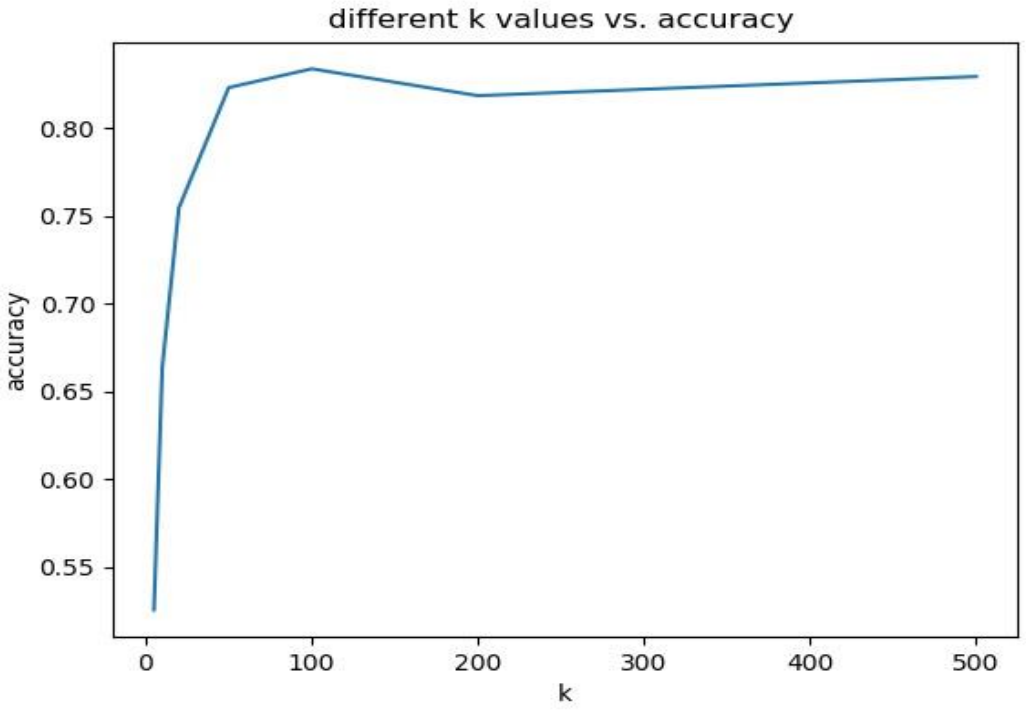

Figure 1: We can observe that as value of number of principle components taken to train increases the accuracy also increases but after some time it is saturated and there no more significant change in accuracy with change in value of k

5.2 Graph 2: Different Number of principle components K vs. Precision

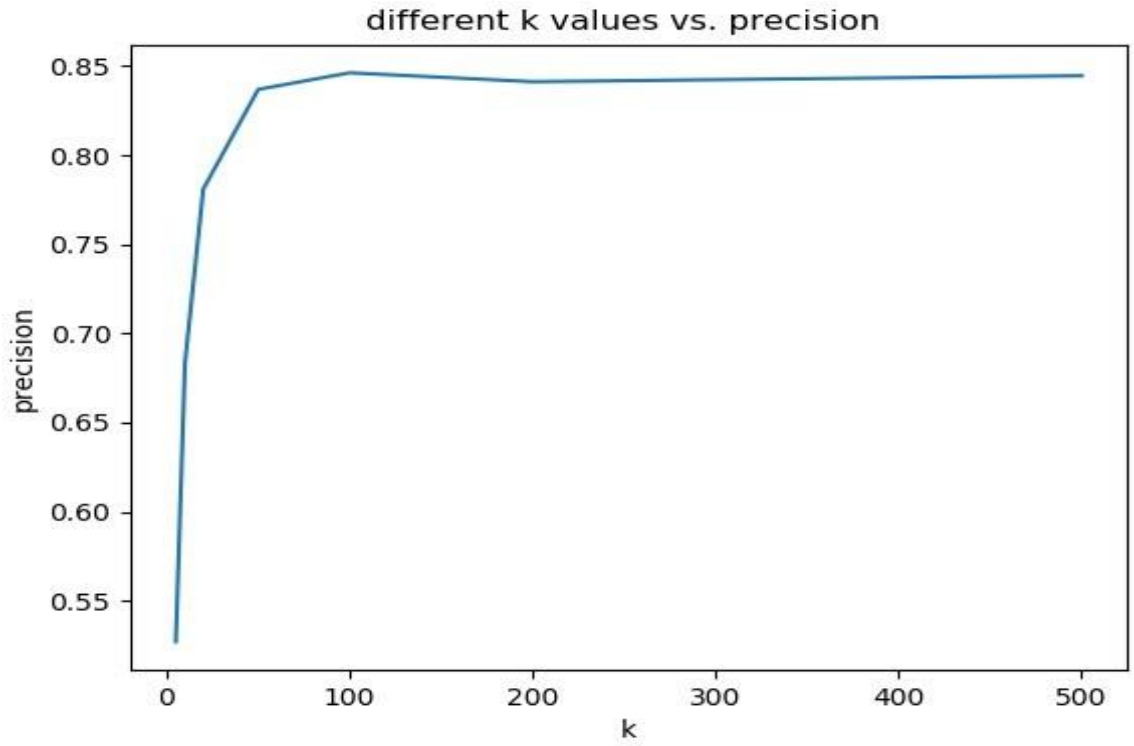

Figure 2: As seen as value of K increases the precision is increasing, it means that the proportion of true positive predictions among all positive predictions is increasing. In other words, the model is becoming more accurate in correctly identifying positive instances.

5.3 Graph 3: Different Number of principle components K vs. Recall

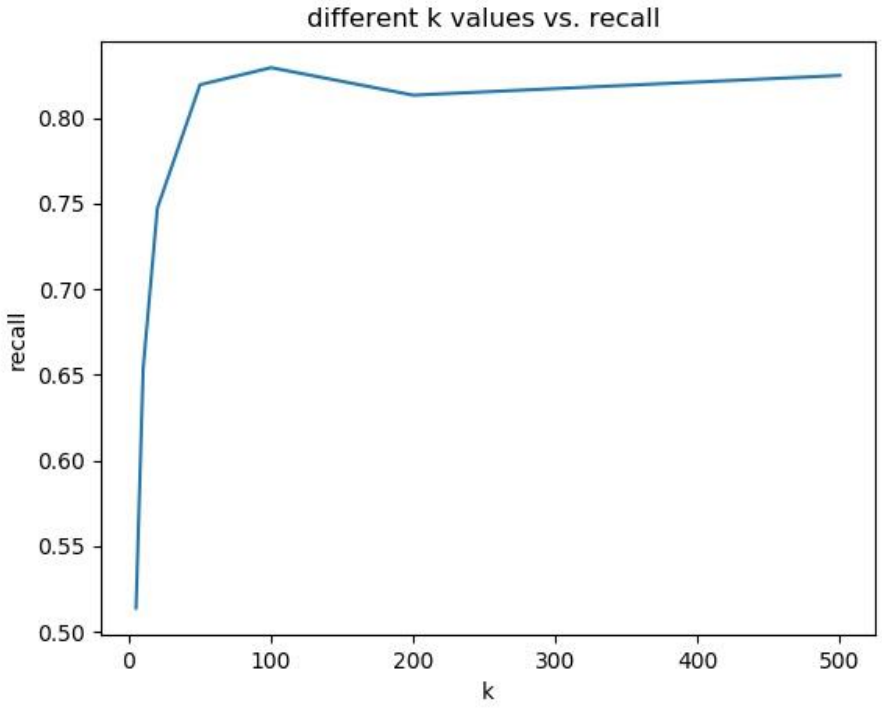

Figure 3: As we can observe above, as value of k increases, the recall is also increasing, it means that the proportion of true positive predictions among all actual positive instances is increasing. In other words, the model is becoming more accurate in identifying all positive instances, even if it means making some false positive predictions.

5.4 Graph 4: Different Number of principle components K vs. F1-Score

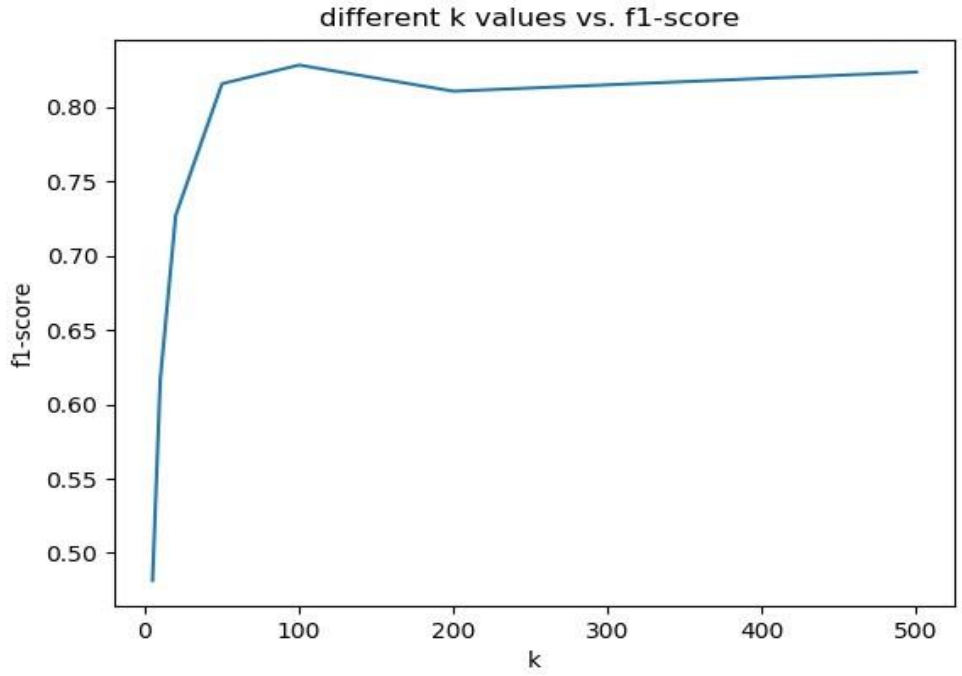

Figure 4: As seen above as value of K increases the value of f1-score also increases, it means that the

model is becoming better at balancing precision and recall.

6. Conclusion:

we can conclude that implementing PCA and multi-class SVM models on the MNIST dataset can be an effective way to classify handwritten digits.

PCA can be used to reduce the dimensionality of the input data, making it easier to process and analyze. By extracting the most important features from the original data, PCA can help to improve the accuracy of machine learning models, while also reducing the computational complexity and memory requirements.

Once the PCA is performed, multi-class SVM models can be trained on the reduced-dimensionality data to classify the handwritten digits. Linear kernel functions can be used to transform the data into a higher-dimensional feature space, where the linear relationships between the original features are more apparent. This makes it easier to find a hyperplane that separates the data into different classes.

We also discussed that the choice of hyperparameters, such as the number of principal components to keep, the regularization parameter, and the learning rate, can have a significant impact on the performance of the model. It is important to carefully tune these hyperparameters to ensure optimal performance.

Overall, implementing PCA and multi-class SVM models on the MNIST dataset can be a useful technique for image classification tasks, and can provide insights into the underlying structure of the data.