E0-270 : Assignment 1

Dhaval Chavda - 20973

March 2023

1 Principal Component Analysis (PCA)

1.1 Task 1

In the first task, we normalise pixel values between -1 and 1. It will help in many areas like

1. comparing variables that have different units or scales, normalising the data can ensure that each variable is given equal weight in the analysis.
2. Normalizing the data also helps to avoid the problem of outliers, which can skew the results of an analysis. By scaling the data to a specific range, outliers are brought within the same range as the rest of the data, making them less likely to have an undue influence on the results.

1.1.1 Method

To normalise data between -1 and 1, I use **Min-Max Normalization** method. In this method, first, I scale the data between 0 and 1. After that, I shifted it between -1 and 1.

Here, I wrote steps for normalization.

1. Find the minimum and maximum values of the data you want to normalise.
2. Subtract the minimum value from all the data points to shift the minimum value to 0.
3. Divide all the data points by the difference between the maximum and minimum values to scale the data to the range of 0 to 1.
4. Multiply all the data points by 2 and subtract 1 to shift the range to -1 to 1.

Formula for this normalization

$$NormalizeData = 2 * ((data - MinData)/(MaxData - MinData)) - 1$$

1.2 Task 2

Implement PCA to reduce the dimensionality of the images from 784 to a lower number of principal components (k = 5, 10, 20, 50, 100, 200, 500).

1.2.1 Method

Implementation Steps :

1. Compute the covariance matrix.
$$CovarMat = np.cov(X.T)$$
2. Compute the eigenvectors and eigenvalues.
$$EigenValues, EigenVectors = np.linalg.eig(coVar_mat)$$
3. Select the principal components according to k.
$$topKEigenvectors = EigenVectors[:, : k]$$
4. Transform the data.
$$TransformedData = np.matmul(self.components.T, X.T).T$$

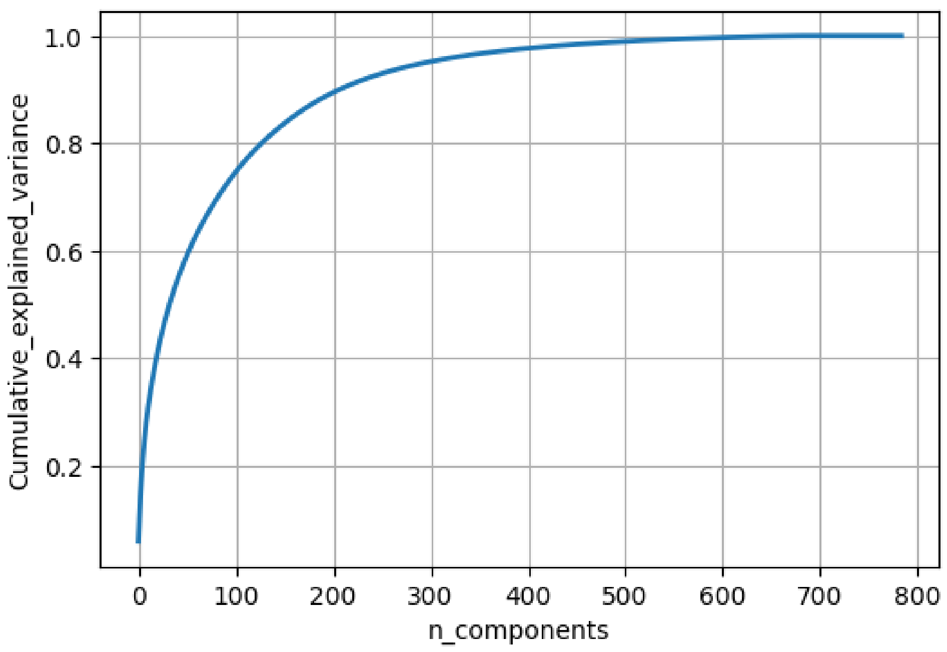

Figure 1: Variance Preserved

1.2.2 Analysis of Results

Here, We reduce the dimensionality of data. So, all the informations(Variance) of data can not be preserved. We find the most useful K features to transform data into K dimensions.
Particularly for the MNIST dataset, there are 784 dimensions and we reduce it into different dimensions like (k= 5, 10, 20, 50, 100, 200, 500). We can calculate how much information is preserved using this given formula:

$$infoPreserved = \frac{\lambda_1 + \lambda_2 + \lambda_3 + ... + \lambda_k}{\sum\limits_{i=0}^{i=d} \lambda_i} \tag{1}$$

where k = No. of Principal Components
d = Dim. of dataset

PCA is a powerful tool for reducing the complexity of data and extracting important features, which can lead to more accurate and efficient machine learning models.

Graph on how much variance can be retained using PCA

In this graph, the x-axis represents the number of principal components, and the y-axis represents the percentage of variance explained by those components. As we can see, the first 100 principal components explain around 70% of the variance in the MNIST dataset, while the first 300 principal components explain around 90% of the variance. This suggests that we can represent the MNIST dataset with a relatively small number of principal components while still retaining a significant amount of information.

2 Support Vector Machine

Since there are 10 classes in MNIST, you have to perform 1-vs-rest classification for each class separately (thereby training 10 SVM models, one for each class). For multi-class prediction, choose the class corresponding to the model that has the highest value of $w^T x + b$

2.1 Task 1

Implement soft SVM from scratch (linear kernel) using stochiastic gradient descent. To implement soft SVM, I use the dual unconstrained problem of SVM. Equation of dual of SVM given below :

$$\min_{w,b} f(w,b) = \frac{||w||^2}{2} + c * \sum_{n=1}^{N} max(0, 1 - y_n(w^T x_n + b))$$

Since this is an unconstrained problem, it can be solved directly using a Stochastic Gradient Descent (SGD). Since the second term is discontinuous, the gradient might not be defined everywhere. When the gradient is not defined, I took it to be zero

2.1.1 Method

1. Set the hyperparameter (learning rate, number of iterations and C)
2. choose random data point and Check for the below condition

   $$y_i * (w^T x_i + b) < 1$$

3. If the above condition is true then set the gradient as :

   $$gradient\_w = w - (C * x_i * y_i)$$

   $$gradiant\_b = -(c * y_i)$$

   otherwiese set gradiant as :

   $$gradiant\_w = gradiant\_b = 0$$

4. Update w and b as :

   $$w = w - learning\_rate * gradiant\_w$$

   $$b = b - learning\_rate * gradiant\_b$$

5. Repeat this for random data points for a number of iterations.

2.2 Task 2, 3 and 4

Use the above-trained models to create a multi-class classifier. Here we use the one vs Rest Strategy for prediction. So, We first train a model for each class as one vs Rest and then find one class with the highest probability.

2.2.1 Method

1. initialize the list of size 10 for each class in MNIST dataset.
2. Train model for each class using the above SVM function using SGD. We use the one Vs rest method to train model
   - first choose one class and make 1 in the label for that class data point and make -1 for all other classes.
   - do this for all classes
3. Predict each class and choose one class with the highest probability.

2.2.2 Results

Here, First, I try with different hyperparameters and try to tune my model. Accuracy during that process is given below :

L Rate	**Iter**	**C**	**5**	**10**	**20**	**50**	**100**	**200**	**500**
0.001	100000	10	56.49	72.34	82.21	83.78	85.36	86.37	82.88
0.001	200000	10	55.2	74	81.57	86.43	82.09	84.88	85.39
0.0001	100000	10	53..9	73.18	84.53	87.59	89.82	89.44	90
0.0001	200000	10	58.2	74.99	85.09	88.24	89.53	90.2	88.34
0.0001	300000	10	59.13	73.73	85.59	88.52	89.54	89.95	90.78
0.0001	400000	10	58.16	74.32	85.08	89.33	89.85	90.66	90.38
0.0001	500000	10	58.77	73.98	84.58	88.3	89.9	90.94	90.25
0.0001	500000	7	59.37	73.51	85.44	88.89	89.86	90.51	90.44

Table 1: HyperParameter Tuning

After doing all this try, I set hyperparameters as follows :

Learning Rate = 0.0001
Iterations = 300000
C = 10

Using above parameters I got as follows :

K	**Accuracy**	**Precision**	**Recall**	**F1 Score**
5	57.4	58.16	56.43	57.28
10	75.32	74.58	74.79	74.69
20	84.83	84.98	84.54	84.76
50	88.89	88.84	88.73	88.79
100	90.06	89.99	89.95	89.97
200	90.47	90.46	90.3	90.38
500	90.29	90.48	90.11	90.31

Table 2: Iterations = 300000, Learning Rate = 0.0001, C = 10

2.2.3 Graphs

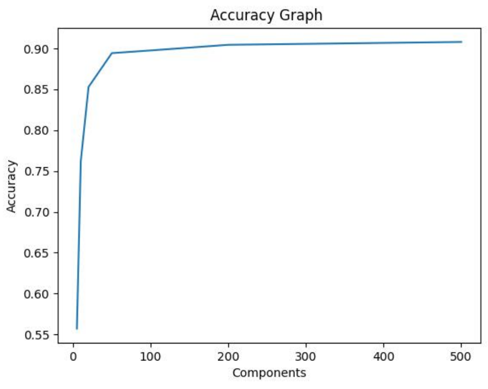

Figure 2: Accuracy

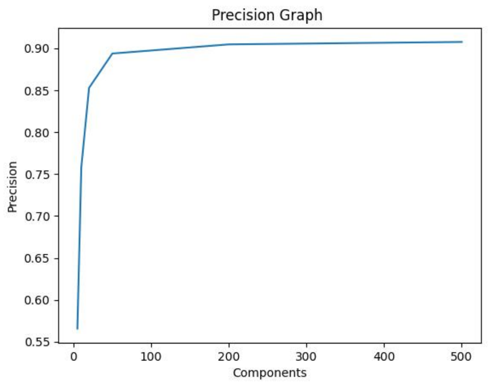

Figure 3: Precision

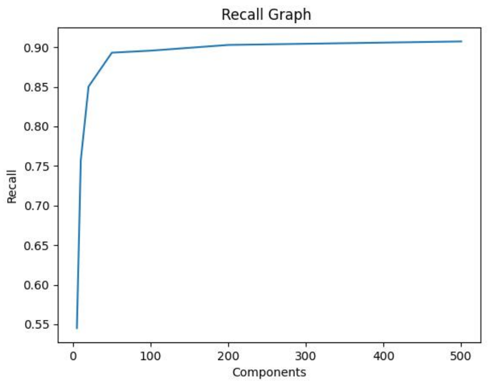

Figure 4: Recall

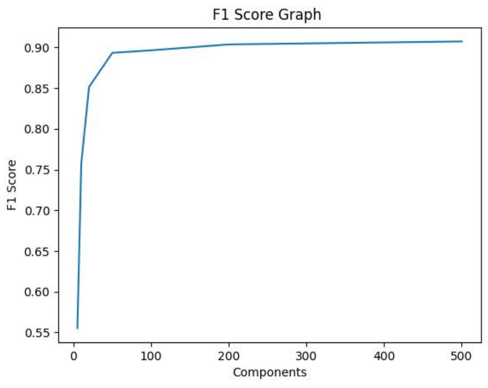

Figure 5: F1 Score

2.2.4 Analysis

At the end of the experiments, We can say that PCA is based on the eigenvalue decomposition of the covariance matrix of the data. The first principal component captures the direction of maximum variation in the data, and subsequent components capture the remaining variance in orthogonal directions. So, it identify important variables in the data and reduces the dimensionality of the data without losing much information. Here, We see that In the MNIST dataset out of 784 dimensions if we reduce to only 300 dimensions, then also we can preserve 90% of the variance of the dataset. After PCA, We apply SVM. We first tune the model and then we can see that I achieved around 88-90% accuracy.