Assignment 3

Quang Hien Tran - 1802936

November 4, 2020

1 Brief description

1.1 Principal Component Analysis

Principle Component Analysis (PCA) is a statistical technique used for dimensionality reduction without losing much-informative data. By combining linearly primary variables, PCA creates a new dataset that has a lower dimension without losing important information. PCA emphasizes variance and covariance. The principal components are the eigenvectors of the covariance matrix of the dataset. Each eigenvector has a corresponding eigenvalue, which indicates the variance of the eigenvector. In other words, the eigenvalue which has a larger eigenvalue will store more informative data (variance).

Steps to calculate PCA

Step 1: Normalizing the data

Because of the technique using a variance, different scales will be a problem, so standardizing the data is very important.

Step 2: Calculating the covariance matrix of the dataset

Step 3: Calculating eigenvalues and eigenvectors of the covariance matrix

Step 4: Calculating new dataset

After having eigenvalues and eigenvectors, you need to choose the eigenvectors which have larger eigenvalues to depend on how many dimensions you want to reduce. The new data set is computed by multiplying the original data and selected eigenvectors.

1.2 K-means clustering

K-means clustering is a simple unsupervised machine learning algorithm. The main idea of K-means clustering is grouping data points into a cluster by similarities. Here, it is the distance to centroids. The data points of one class will cluster together and a data point will be assigned to a cluster if the distance from the data point to the centroid of the cluster is shortest.

K-means clustering algorithms can be demonstrated:

Step 1: Initializing

Specify the number K of clusters and randomly select K centroids from the dataset

Step 2: Loops:

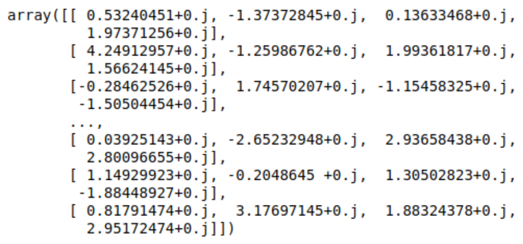
1. assign each point to the closest centroids by distance
2. update new centroids by computing mean of clusters till convergence

2 Implementation

2.1 Task 1

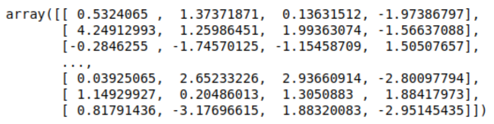
In this task, I will implement PCA from scratch.

Figure 1: The result of function with n_components = 4



```
array([[ 0.53240451+0.j, -1.37372845+0.j,  0.13633468+0.j,
          1.97371256+0.j],
       [ 4.24912957+0.j, -1.25986762+0.j,  1.99361817+0.j,
          1.56624145+0.j],
       [-0.28462526+0.j,  1.74570207+0.j, -1.15458325+0.j,
         -1.50504454+0.j],
       ...,
       [ 0.03925143+0.j, -2.65232948+0.j,  2.93658438+0.j,
          2.80096655+0.j],
       [ 1.14929923+0.j, -0.2048645 +0.j,  1.30502823+0.j,
         -1.88448927+0.j],
       [ 0.81791474+0.j,  3.17697145+0.j,  1.88324378+0.j,
          2.95172474+0.j]])
```

Figure 2: The result of library with n_components = 4



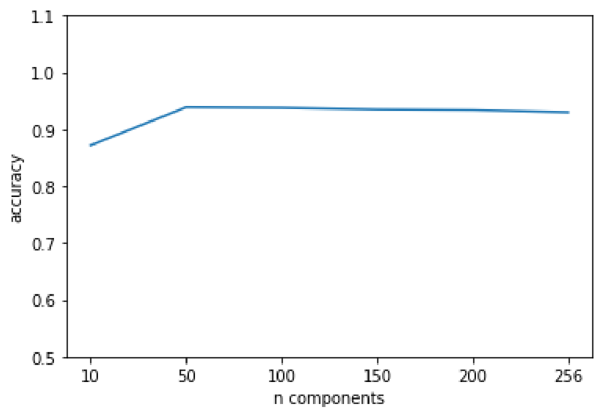
```
array([[ 0.5324065 ,  1.37371871,  0.13631512, -1.97386797],
       [ 4.24912993,  1.25986451,  1.99363074, -1.56637088],
       [-0.2846255 , -1.74570125, -1.15458709,  1.50507657],
       ...,
       [ 0.03925065,  2.65233226,  2.93660914, -2.80097794],
       [ 1.14929927,  0.20486013,  1.3050883 ,  1.88417973],
       [ 0.81791436, -3.17696615,  1.88320083, -2.95145435]])
```

The result of my function in Figure 1 is very close to the one by third party library in Figure 2.

2.2 Task 2

In this task, I will apply the PCA function to both the training and testing set and use a 1-nearest neighbor classifier to evaluate the performance when reducing dimensions. The reduced dimensions are [10,50,100,150,200,256]. The result is shown in Figure 3.

Figure 3: The result of library with n_components = 4



The performance of the classifier increase from n=10 to n=50 and does not change much in the rest of the line. The explaination of this is when applying PCA with n lower than 50, the informative data is lost apporiately, so it effects the performance of model. When increasing number of components over 50, the rest of components does not distribute much, so it does not change the performance of model significantly.

Figure 4: The data distribution of n_components

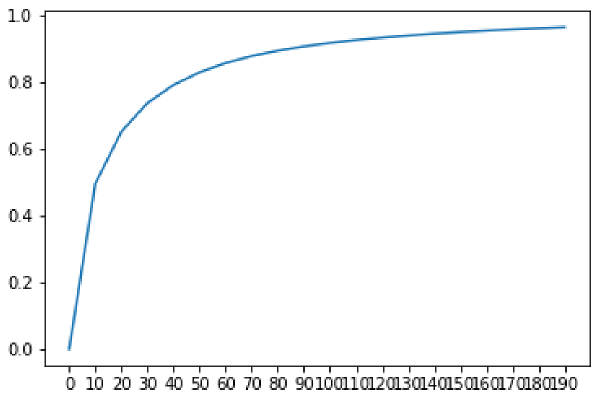
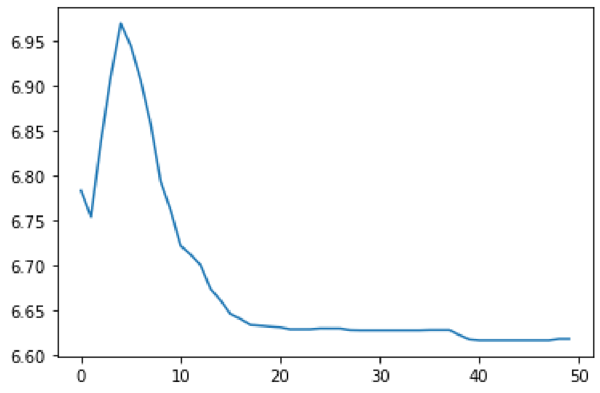


Figure 4 proves that the PCA with n=10 lost much information. When increasing the number of components. the loss decease but does not change much from n=100.

2.3 Task 3

In this task, I will implement K-means algorithms to classify MNIST without dimensionality reduction.

Figure 5: Loss curse through iterations



In Figure 4, the loss increases then drops dramatically and converges around iteration = 50. The loss increases in the beginning, I think, depends a lot on initial centroids. After all, the loss decrease proves that the function works.

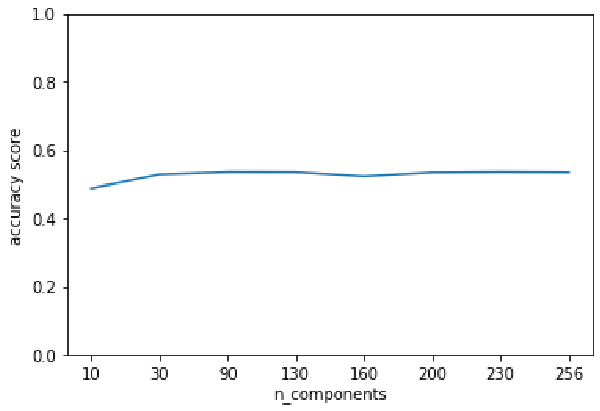
2.4 Task 4

In this task, I will evaluate the effect of PCA on the performance of K-means algorithms. The initial centroids will be fixed through this task.

The accuracy score without applying PCA is: 0.5372

The performance of K-means classifier in Figure 6 does not change through different n components. The explaination of this is PCA trying to secure variance and covariance, so relative distances between score does not change; thus, PCA will not affect K-means classifier. The only result, I think, is reducing computations.

Figure 6: Performance of model with different n_components



2.5 Task 5

In this task, I will evaluate the effect of PCA on a noisy dataset. The original dataset will be added 256-dimension random feature.

PCA	Noise	Accuracy Score 1-nearest neighbor	Accuracy Score SVM
		0.928	0.921
	X	0.568	0.874
X	X	0.634	0.826

Table 1: Performance with different options

Without using PCA and noise, both classifiers have a good performance (90%). When adding noisy dimensions to the dataset, the performance of the 1-nearest neighbor drops significantly while the score of SVM does not affect much. When applying PCA to the noisy dataset, the performance of 1-nearest neighbor improves and the one of SVM still does not change much.

The reason for the performance drop of the 1-nearest neighbor is that this classifier is sensitive to noise. Calculating distance from a data point to its nearest neighbors highly depends on local points, especially K=1 in this case. The problem might be improved when increasing K.

The result of SVM shows that this classifier is robust to noise. Adding the noisy features, in this case, does not affect its performance. The reason is the SVM cannot use additional features (just random numbers) to distinguish the classes, so the final hyperplane tends to separate the original features and parallel the noisy features.

PCA can help reduce noise. Noise affects the dataset and all eigenvalues/ eigenvectors . In PCA, we just use top n eigen values and leave the rest of it, so theoretically it can reduce the effect of noise.