ASSESSMENT COVER SHEET

Assignment Details

Course: Introduction to stastical Machine Learning

Semester/Academic Year: 2020 sem 2

Assignment title: PCA and k-means clustering

Assessment Criteria

Assessment Criteria are included in the Assignment Descriptions that are published on each course's web site.

Plagiarism and Collusion

Plagiarism: using another person's ideas, designs, words or works without appropriate acknowledgement.

Collusion: another person assisting in the production of an assessment submission without the express requirement, or consent or knowledge of the assessor.

Consequences of Plagiarism and Collusion

The penalties associated with plagiarism and collusion are designed to impose sanctions on offenders that reflect the seriousness of the University's commitment to academic integrity. Penalties may include: the requirement to revise and resubmit assessment work, receiving a result of zero for the assessment work, failing the course, expulsion and/or receiving a financial penalty.

DECLARATION

I declare that all material in this assessment is my own work except where there is clear acknowledgement and reference to the work of others. I have read the University Policy Statement on Plagiarism, Collusion and Related Forms of Cheating:

http://www.adelaide.edu.au/policies/?230

I give permission for my assessment work to be reproduced and submitted to academic staff for the purposes of assessment and to be copied, submitted and retained in a form suitable for electronic checking of plagiarism.

Vandit
2/11/20

SIGNATURE AND DATE

2020 Introduction to Introduction to Statistical Machine Learning Assignment 3: PCA and K-means

Vandit Jyotindra Gajjar
School of Computer Science, The University of Adelaide
vanditjyotindra.gajjar@student.adelaide.edu.au
Student ID: a1779153

Abstract

As time passes the Machine Learning community has seen massive growth in the field in theory as well as in applications. One of the earliest concepts for dimensionality reduction in the data was Principal Component Analysis (PCA) [1]. PCA is an unsupervised method that is fast and adaptable. This assignment/report will mainly focus on the implementation of PCA from the scratch and use K-means clustering to group the data and perform classification accordingly by using a 1-nearest nearest classifier and a linear Support Vector Machine (SVM) classifier [2]. PCA is a linear dimensionality reduction technique that can be applied for capturing the prominent information from a high-dimensional space by applying the projection to a lower-dimensional subspace. PCA is basically an Unsupervised dimensionality reduction algorithm, where one can cluster similar data points based on the feature correlation without having the supervision (in our case labels). In this assignment, we focus on the dimensionality reduction, classification, and k-means clustering task. Additionally, based on the assignment requirements we compare our custom implemented PCA and K-means algorithm with the available implementation of the technique by an open-source library scikit-learn [3] and with the SVMs [2]. The experiment was conducted on the provided dataset – MNIST subsampled dataset [4]. The dataset consists of 6000 training samples and 1000 testing samples. This report focuses on solving the assignment questions of the PCA and K-means clustering algorithms with detailed analysis. The source code is publicly available here.

A General Introduction to Principal Component Analysis (PCA)

The Principal Component Analysis (PCA) is an unsupervised algorithm for dimensionality reduction which is used in most cased where the datasets have the high dimensionality, so by transform a large-scale dataset to a smaller one with having most of the information. This transformation can be done by reducing the number of variables; however,

this process comes with an expense of the accuracy, but the important point of this dimensionality reduction technique is to make a trade-off with the accuracy for ease. When we transform a large-scale dataset to a smaller one, a small dataset can be explored and visualized easily and because of that for any Machine Learning algorithm the analysis will be much faster and easier. So basically to sum it up, PCA is easy to implement and understand – Preserve most of the information while reducing the number of variables. The steps to understand the PCA is provided below with in-depth description.

- **Standardization:** The core aim of the first step – Standardization is to standardize the range of the continuous initial variables so that each variable can participate equally in the analysis process. The main reason behind this process prior to PCA is that if there will be a huge difference between the initial variables range then because of this the large ranges variables will dominate over the small range variables which will lead to very biased results at the end. For example, A variable has ranged from 0 to 50 will dominate over the variable range from 0 to 5. So, standardization transforms the data into equivalent scales to avoid this issue. The below equation shows how the standardization can be done by subtracting the mean and divide by the standard deviation for every variable each value.

$$z = \frac{\text{varibale value - mean}}{\text{standard deviation}}$$

- **Covariance Matrix Computation:** The aim of performing this step is to check if there are any relationship presents between the variables. The main reason is that sometimes variables are highly correlated with each other in a such way that there will be redundant information captured at the end and this leads to biased results and thus the computation of the covariance matrix takes place. The covariance matrix is an m x m symmetric matrix and in which each entries covariances are associated with the possible pairs of the initial variables. For example, if we have 3 variables a, b, and c, the covariance matrix is 3 x 3 matrix form for a 3-dimensional dataset. If the sign of these entries is positive, then the variables are correlated otherwise they are inversely correlated.

$$\begin{matrix} \text{Cov}(a,a) & \text{Cov}(a,b) & \text{Cov}(a,c) \\ \text{Cov}(b,a) & \text{Cov}(b,b) & \text{Cov}(b,c) \\ \text{Cov}(c,a) & \text{Cov}(c,b) & \text{Cov}(c,c) \end{matrix}$$

- **Compute The Eigenvectors and Eigenvalues Of The Covariance Matrix:** To determine the principal components of the data, we need to compute the eigenvectors and eigenvalues from the covariance matrix. The principal components are constructed in a manner that the first principal component will have the largest possible variance in the dataset. The second component will be constructed in the same manner having uncorrelation with the first component and having next highest variance. This step continues until total P components have been calculated equal to the total variables. So basically by ranking the eigenvectors corresponding to eigenvalues from high to low value, we get the principal components in significant order. For an example we have 2-dimensional dataset having 2 variables a, b and the covariance matrix and eigenvectors and eigenvalues are as follows:

$$vec1 = \begin{matrix} 0.67 \\ 0.73 \end{matrix} \qquad \boldsymbol{\alpha}1 = 1.28$$
$$vec2 = \begin{matrix} -0.73 \\ 0.67 \end{matrix} \qquad \boldsymbol{\alpha}2 = 0.04$$

  So here because of $\boldsymbol{\alpha}1 > \boldsymbol{\alpha}2$, PC1 will be of *vec1* and PC2 will be of *vec2.*
- **Feature Vector:** Computing the eigenvectors and eigenvalues from the covariance matrix will help us to find the significant order of the principal component. In this step we discard those components who have the less significance/low eigenvalues and by following the process, we form the feature vector from the remaining matrix of vectors. For example we have the vector as follows

$$\begin{matrix} 0.67 & -0.73 \\ 0.73 & 0.67 \end{matrix}$$

  Now finally because we have to discard the eigenvector who has the less significance the final feature vector will be

$$\begin{matrix} 0.67 \\ 0.73 \end{matrix}$$

- **Recast The Data Along The Principal Components Axis:** The final step of the PCA is process is using the formed feature vector to reorient the data from the original axes to the ones which is represented by the principal components. The process can be

done by multiplying the transpose of the original data set by the transpose of the feature vector.

$$\text{Final Dataset} = \text{Feature Vector}^{\text{Transpose}} \times \text{Standardized Original Dataset}^{\text{Transpose}}$$

A General Introduction to K-means Clustering

Before diving deep into K-means clustering, clustering is the process of dividing the data into clusters/groups based on the patterns in the data. Clustering is also an unsupervised learning problem, in which we do not have any target or class to predict instead we look and analyze the data and try to make clusters based on the similarities in the dataset and form the clusters. There are certain properties of clusters which are important and are listed below:

- In the datasets, all the instance points in a cluster need to be similar to each other.
- The different cluster's data points should be different from the clusters.

There are some applications for the clustering as well which are document clustering, customer segmentation, image segmentation, recommendation systems, etc.

The K-means clustering is an algorithm that tries to maximize the distance between data points/instances in a cluster corresponding to the centroid. The core aim of the K-means algorithm is to minimize the total sum of distances between the data points/instances and their corresponding cluster centroid. The main steps to implement the K-means algorithm is described below with the steps:

We have a training dataset $x^1, x^2 \ldots\ldots, x^m$ and we need to form the clusters. We have been provided with the feature vector for each of this data points/instances but no associated labels. Our primary goal is to predict K centroids and label for each of these points. The algorithm will be as follows:

1. Initialize cluster centroids $\mu_1, \mu_2, \ldots\ldots, \mu_k \in R^n$
2. Repeat this process until convergence: {

   For every i, set

   $$c^{(i)} = \arg \min_j ||x^{(i)} - \mu_j||^2$$

For every j, set

$$\mu_j = \frac{\sum_{i=1}^{m} 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^{m} 1\{c^{(i)} = j\}}$$

The simple steps of this algorithms are as follows:

- Choose the number of K clusters to form
- Select K random points from the data points as centroids
- Assign all the data points to the closest cluster centroid
- Recompute centroid of newly formed clusters
- Repeat steps 3 and 4 until convergence

Dataset Statistics

In our assignment, the requirement is to use the provided data which is MNIST Dataset [4] but in the subsampled version. The dataset contains images for 10 digits which means 10 classes are present. The training dataset contains 6000 training samples and 1000 testing samples. The dimension of the image is 28 x 28. The data is in the form of CSV, so which makes a total of 784-pixel values have been provided between 0 and 1. The task is to implement PCA and K-means for this data and perform classification.

Implementation Details

In our experiment, we have performed the training on a workstation with the Intel i-7 core processor and accelerated by NVIDIA GTX 1060 6 GB GPU. All the experiments run in the Windows platform using Anaconda virtual environment and Google Colab Notebooks [5]. The training and testing data is 6000 and 1000 sample points consisting of 10 labels. For simplicity, we use the NumPy [6] and Pandas [7] library for mathematical operations.

Experiments

We perform several experiments on the dataset based on the assignment requirements. The first task was to implement PCA from the scratch and find the mean vector and projection matrix from the MNIST training data which we have completed by using the algorithm provided

in the lecture slides. Then after implementing the PCA from scratch, we have applied the PCA to the training and testing dataset and performed the classification with 1 – nearest neighbor classification. We have analyzed the performance change against the different reduced dimensions ranging from 256 to 10. Figure 1 showcases the error-rate vs. PCA reduced dimension from 256 to 10. We can observe that after the curve flattens when the dimensions reach 150. Figure 2 also provides the error-rate values for different dimensions.

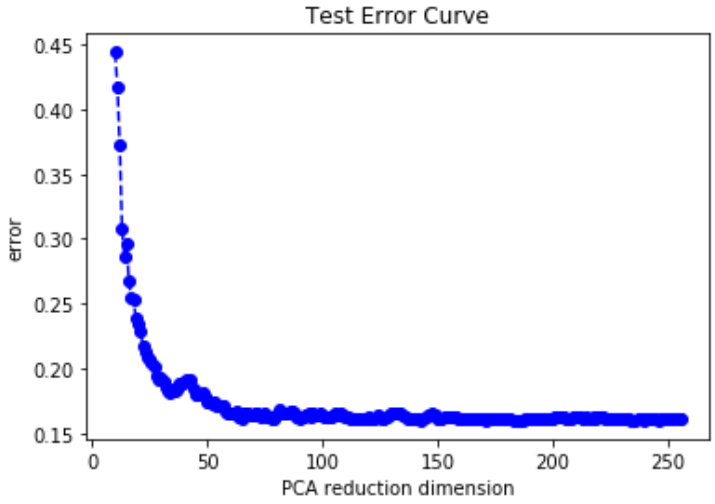

Figure 1. **Test error curve** for reduced dimensionality ranging from 256 to 10.

```
[0.444 0.417 0.372 0.308 0.287 0.296 0.267 0.254 0.253 0.239 0.235 0.229
 0.218 0.213 0.209 0.207 0.204 0.201 0.195 0.192 0.193 0.19  0.186 0.183
 0.182 0.184 0.183 0.185 0.188 0.188 0.19  0.191 0.19  0.191 0.184 0.18
 0.18  0.18  0.181 0.179 0.174 0.174 0.173 0.174 0.171 0.172 0.172 0.171
 0.167 0.165 0.165 0.165 0.165 0.167 0.163 0.162 0.166 0.165 0.165 0.164
 0.164 0.165 0.165 0.163 0.164 0.165 0.163 0.163 0.161 0.162 0.164 0.169
 0.166 0.165 0.166 0.166 0.164 0.167 0.165 0.163 0.162 0.163 0.163 0.164
 0.163 0.165 0.165 0.164 0.164 0.165 0.164 0.163 0.163 0.163 0.163 0.164
 0.165 0.164 0.165 0.164 0.163 0.163 0.162 0.162 0.162 0.162 0.162 0.162
 0.162 0.162 0.163 0.161 0.162 0.161 0.164 0.163 0.162 0.162 0.163 0.164
 0.166 0.166 0.165 0.165 0.163 0.162 0.163 0.162 0.161 0.161 0.162 0.162
 0.161 0.16  0.162 0.164 0.164 0.164 0.165 0.164 0.162 0.162 0.162 0.162
 0.162 0.163 0.163 0.163 0.163 0.162 0.162 0.162 0.161 0.161 0.161 0.161
 0.161 0.162 0.162 0.162 0.162 0.162 0.162 0.161 0.161 0.161 0.161 0.161
 0.161 0.162 0.162 0.161 0.161 0.16  0.16  0.16  0.16  0.16  0.161 0.161
 0.161 0.161 0.162 0.162 0.162 0.161 0.162 0.162 0.161 0.162 0.163 0.163
 0.163 0.163 0.163 0.163 0.162 0.162 0.162 0.162 0.163 0.163 0.163 0.162
 0.162 0.163 0.162 0.164 0.163 0.162 0.163 0.163 0.163 0.161 0.161 0.161
 0.162 0.162 0.162 0.161 0.162 0.162 0.162 0.161 0.16  0.16  0.16  0.16
 0.161 0.161 0.16  0.161 0.161 0.162 0.162 0.162 0.16  0.161 0.161 0.161
 0.162 0.162 0.162 0.162 0.162 0.162 0.162]
```

Figure 2. **Error rate values** for different dimensions from 10 to 256. Refer this vector from left to right and top to bottom for 10 to 256 dimensions.

The following task was to implement K-means clustering from the scratch. We have implemented the K-means clustering using standard libraries and applied to the MNIST training set without dimensionality reduction. Then after we have observed the results by plotting the

loss curve which can be shown in Figure 3. The plot shows the curve between the change of loss value of the K-means algorithm vs. the number of iterations. The minimum distance has been set to 50000 and the number of iterations were 40 in our case. The values have been provided in Table 1.

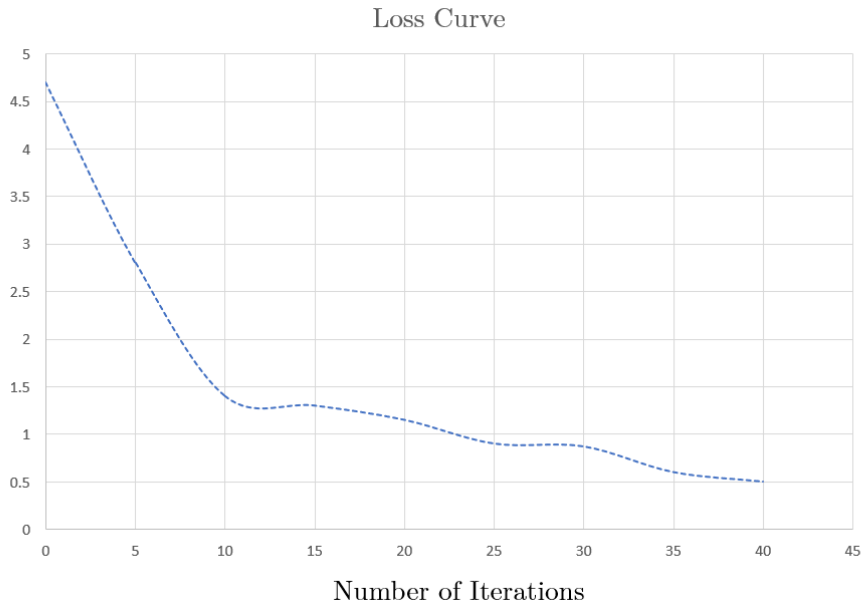

Figure 3. **Loss curve** between the change of loss value of the K-means clustering algorithm vs. the number of iterations.

Table 1. **Loss value** for the number of iterations.

Number of Iterations	**Loss**
0	4.7466
5	2.8192
10	1.4943
15	1.3029
20	1.1531
25	0.9503
30	0.8749
35	0.6193
40	0.5198

Task 4 is challenging where we have to randomly choose one of the training samples from each class as initial clustering centers where in our case that will be 10 centers in total. Now after applying the K-means algorithm to form the clusters into 10 groups, for each cluster we need to calculate the percentage of samples sharing the same digit as the initial group center.

Finally after taking the average of these percentages will be evaluation matrices for K-means clustering. We have performed this experiment with dimensionality reduced features. In Figure 4, we showcase the performance against the different reduced dimensions ranging from 256 to 10 vs an average percentage.

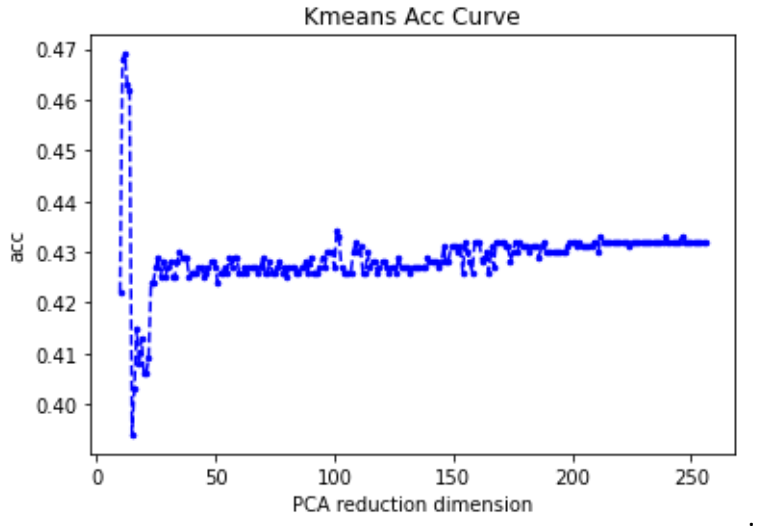

Figure 4. **Performance** against the different reduced dimensions ranging from 10 to 256

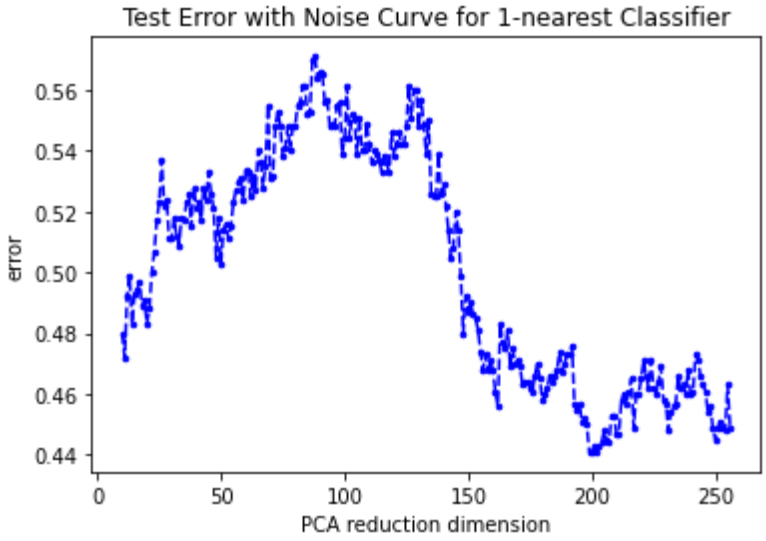

Figure 5. **Test Error** with Noise curve for 1-nearest classifier vs. PCA reduced dimensions ranging from 10 to 256.

The final task was to add the 256 noisy dimensions to the original data, so consider each sample $\mathbf{x_i}$, appending a 256-dimensional random feature vector to $\mathbf{x_i}$ to make it a 512-dimensional feature. After performing this task we performed the PCA classification by using a 1-nearest classifier and a linear SVM classifier. Figure 5 showcase the error plot of different reduced dimensions vs 1-nearest Classifier error. Figure 6 showcase the error plot of different reduced dimensions vs linear SVM classifier error. From figure 5 and figure 6, we conclude that

in the case of 1-nearest classifier with the change in PCA reduced dimensions the error-rate increases drastically, thus PCA is sensitive in the case of 1-nearest classifier and linear SVM. The core reason is that PCA finds a lower-dimensional feature representation of the data points to minimize the squared error. Thus if the features are irrelevant or noisy, then PCA will count errors with equal importance and leads to poor result when reducing the dimensions.

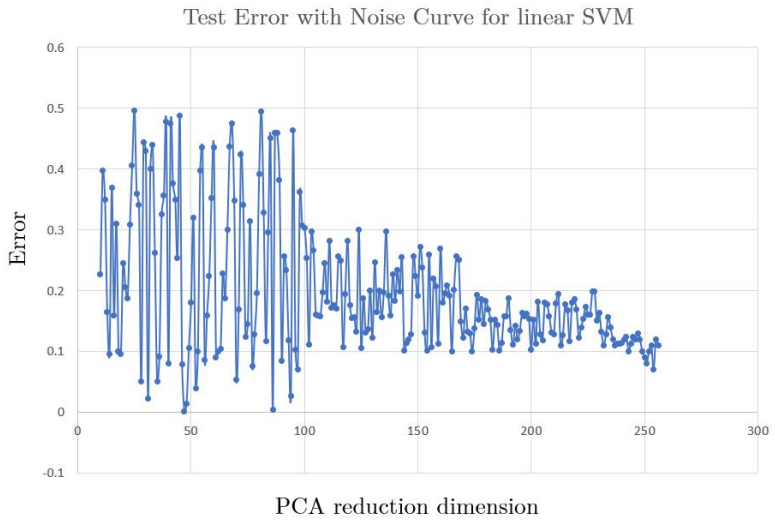

Figure 6. **Test Error** with Noise curve for linear SVM classifier vs. PCA reduced dimension ranging from 10 to 256.

References:

[1] Wall, Michael E., Andreas Rechtsteiner, and Luis M. Rocha. "Singular value decomposition and principal component analysis." A practical approach to microarray data analysis. Springer, Boston, MA, 2003. 91-109.

[2] Cortes, Corinna, and Vladimir Vapnik. "Support-vector networks." Machine learning 20.3 (1995): 273-297.

[3] Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

[4] LeCun, Yann, Corinna Cortes, and Christopher JC Burges. "The MNIST database of handwritten digits, 1998." URL http://yann. lecun. com/exdb/mnist 10.34 (1998): 14.

[5] Bisong, Ekaba. "Google Colaboratory." Building Machine Learning and Deep Learning Models on Google Cloud Platform. Apress, Berkeley, CA, 2019. 59-64.

[6] Walt, Stéfan van der, S. Chris Colbert, and Gael Varoquaux. "The NumPy array: a structure for efficient numerical computation." Computing in science & engineering 13.2 (2011): 22-30.

[7] McKinney, Wes. "pandas: a foundational Python library for data analysis and statistics." Python for High Performance and Scientific Computing 14.9 (2011).