Gurjus Singh

October 18th, 2020

MSDS 422 – Practical Machine Learning

# Assignment #5 Principal Components Analysis & Clustering

**Data preparation, exploration, visualization**

In this Data Analysis task, I had a chance to get my feet wet in the world of unsupervised machine learning algorithms. I went over mathematical concepts such Principal Component Analysis, and Eigenvalues and Vectors before running the unsupervised learning code. As before I first loaded my data set using the read\_csv function provided to us by the Pandas package. This week’s dataset was the MNIST dataset provided by Kaggle [1]. The data set consists of pixels for a set of numbers which correctly have to go in distinguishable categories [1]. I then looked at the shape of the data, and I saw an astounding number of columns which represented 784 features. I then took out the label column, which was the label to used to recognize what number the pictures were associated with.

After dropping the columns, the next step was to prepare our pixels for Principal Component Analysis. This meant we had to scale the data using standard scaling, and compute eigen vectors which explains the variation of the data [1]. After computing eigenvectors and values, I plotted the values and number of eigenvectors associated with each value.

A picture containing chart, histogram

Description automatically generated

*Plot 1-1 Eigenvalues to Values Plot*

As seen in 1-1, the plot explained how many vectors are needed to explain a high variance of the data. According to 1-1, I saw that 140 Principal Components could explain 80% of the variance, 319 could explain about 95% of the data and 725 or above could explain 100%. I used these Principal Components to test out the KMeans clustering algorithm.

After seeing how many vectors were good to explain the data, I want to see the individual eigenvectors. The picture in 1-2 showed 28 eigenvectors. I then looked at what the dataset of pixels represented in 1-3. It showed the data, which showed a few numbers could easily be mistaken and put into the wrong cluster. It was interesting to see this.

A screenshot of a cell phone

Description automatically generated

*Eigenvectors 1-2*

A close up of a keyboard

Description automatically generated

*Dataset 1-3*

The next steps of the process involved choosing the number of principal components and saw whether it was clustered correctly using K-Means. From looking at 1-1 I chose 140, 319 and 784 Principal Components respectively to test out K-Means. I then ran K-Means one more time without PCA but used the training set in Cell 1-4 below. I used metrics such as Silhouette Score, Homogeneity Score, and Completeness Score to judge how KMeans was performing.

Text

Description automatically generated

*Cell 1-4*

**Review research design and modeling methods**

As mentioned briefly in the previous section our training algorithm is called KMeans. Before using K-Means I created a 140 Principal Components. Principal Components are vectors that explain the variance. The first vector explains most of variance and the second vector explains the second most variance in the data and so on [1]. All 140 vectors are orthogonal to each other and are known as Eigenvectors [2]. Eigenvectors have associated Eigenvalues which describe the variance or “magnitude” that is associated with each Eigenvector [2]. With these eigenvectors and eigenvalues, it can help shrink the dataset or use less features also known as “Dimensionality Reduction” [3]. In order to use PCA in the code, I had to use standard scaler. After I standardized the data set, I then specifically told PCA how many vectors to make. I then fit the Principal Component Analysis to the dataset, and I got a beautiful output plot. I did this for 140 vectors, 319 vectors and 784 vectors respectively as 140 vectors explained about 80 percent of the data, 319 vectors explained 95 percent of the data and 784 vectors explained 100 percent of the data. After creating a low dimension data, I then tried to see how KMeans performed with PCA.

K-Means is a type of unsupervised learning task [1]. It is known as this as I do not have a target variable to check the predictions [1]. The way it works is one can specify how many clusters to use [1]. After specifying the K clusters to create the algorithm tries to randomly find the location to put each centroid [1]. Once it picks the location of each centroid, it then tries to calculate the Euclidean Distance of each data instance, the lowest distance is where KMeans places the data instance [1]. For this data set, I only had to drop the label variable. I did not have to split the data as this is no supervised learning algorithm like Linear Regression.

**Review results, evaluate models**

Chart, scatter chart

Description automatically generatedBefore running KMeans I had to generate the Principal Components. I then plotted the Principal Components to see what the initial lower dimension data looked like which is shown in Plots 1-5-1-7. From looking at the original data, the data is overlapped. After doing KMeans I wanted to see if the data was less overlapped. After running KMeans, I looked at the plots again, but the clusters were still overlapping. I also noticed that data instances were in different groups each time I ran the KMeans on different PCs, but I did not notice any other difference, so I chose to evaluate the KMeans on crosstabs, Silhouette Score, Completeness Score, and Homogeneity score.

*PCA PLOT – 1-5 WITH 140 PCS*

*Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated*

*PCA PLOT 1-6 WITH 319 PCAS PCA PLOT 1-7 WITH 784 PCS*

*Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated*

*K-Means 1-8 PCS 140 K-Means 1-9 319 PCS*

*Chart, scatter chart

Description automatically generated*

*1-10 KMEANS 784 PCS*

I first looked at the cross tabs for the 140, 319 and 784 PCs. For 1-11 which was for the 140 PCs I noticed that the algorithm was not performing that well and was misclassifying fours and nines as ones. I also saw it was misclassifying threes and eights as sixes. In crosstab 1-12 for 319 PCs it was performing a little bit better as it was mostly misclassifying fours and nines as sevens. It was also putting a huge number of data instances in correct clusters such as 1387 instances in cluster two which real label was also two. I was expecting the 784 PCs KMeans to perform the highest, so I then looked at the crosstab in 1-13. What I found was it still misclassified for example it misclassified threes as eights, but I did see instances that were identified in the correct cluster such as most fours were put in cluster 4 and most eights were correctly identified as well. So, this algorithm with 784 PCs did perform slightly higher.

I then looked at the metrics for all three algorithms with PCs. To explain I first looked at the homogeneity score which is metric between 0 and 1 which explains if all the cluster instances are all each of a single class label [4]. 0 meaning imperfect cluster instances and 1 meaning perfect [4]. I also looked at the silhouette score which is between 1 and -1 [5]. -1 indicates instances in wrong cluster, 1 means perfect and 0 means overlapping [5]. I lastly checked out completeness score which ranges from 0 to 1 and tells if the clusters are correct where 1 is perfect score [6].

After understanding the metrics I looked at the scores for all three PC options in 1-14-1-17 and what I found was that each algorithm stayed near 0.4 for homogeneity which meant that some of the clusters had mostly one class but it was not perfect. I also noticed that most of the Silhouette Scores were close to 0s which suggested they were all mostly overlapping. KMeans 784 PCs was performing the worst while 140 PCs was performing the best in terms of less overlap. Lastly, I looked at the Completeness Score and it looked like the 140s PCs were performing the best in terms of completeness. This was pretty surprising as I did not see 140PCs performing better than 784PCs and 340PCs respectively. Although it was higher all three algorithms were performing around the same rate in terms of choosing corrected predicted labels which means there should be an improvement made.

I lastly, did K-Means without using PCAs as seen in 1-17. What I found was that the crosstabs performed the same as using PCA. I also found that all the scores were high. But in terms of correctness it was still misclassifying at 0.49 for completeness score. It was also doing slightly better at not overlapping and was also doing better at putting single label data in a single cluster, but it still performed below I wanted it to perform.

A picture containing table

Description automatically generated

A picture containing table

Description automatically generated

*Cross tab 1-11 140 PCS Cross tab 1-12 319 PCs*

A picture containing table

Description automatically generated

*Graphical user interface, text, application

Description automatically generated*

*1-13 784 PCs 1-14 Scores for 784 PCs*

*Graphical user interface, text, application, email

Description automatically generated*

*Graphical user interface, text, application

Description automatically generated*

*1-15 Scores for 319 PCs 1-16 Scores for 140 PCs*

*A picture containing calendar

Description automatically generated*

*1-17 Direct Data KMeans without PCA*

**Implementation and programming**

The packages I had to use for this data analysis was the Sklearn, Pandas package numpy and plotly package as seen in 1-18. After importing the packages, I used the Pandas package to **read\_csv()** to import data into Python. I then took a look using **.shape.** I saw that 784 columns were in the data set which represent a huge number of features. I used **.drop** from pandas to drop label since this will be used after training as I was using a unsupervised method. I then used numpy package to find the covariance matrix using **np.cov().** This tells us how many pixels are related and how we can express our data. After this step, I could compute eigenvectors and eigenvalues to find out how much of the data can be compressed. This was used by taking the covariance matrix and performing **np.linalg.eig()** function from numpy.

After figuring this out, I then plotted a chart using the plotly package and **go.Scatter()** function to see how many vectors could explain a huge amount the data. What I found was that 80 percent of the data could be explained using 140 vectors, 319 explained 95 percent of the data and 100 percent of the data could be explained by 784 vectors. I then used the PCA to see 30 eigenvectors. I plotted the eigenvectors using a for loop and **plt.subplot() function.** After looking at the values I also looked at subplots using the training data which were numbers.

Graphical user interface, text, application

Description automatically generated

*1-18 Import Packages Code*

After initial data manipulation, I then used PCA on 140, 319 and 784 eigenvectors. I used **PCA(insert eigenvector quantity here)** to get the specified number of eigenvectors I wanted. I used .fit\_transform() function to reduce the dimensions of the original data. This was scaled data which I used **StandardScaler.fit\_transform() method** on before using PCA. This was part of sklearn package in 1-18. After reducing dimensions I then used **go.Scatter() again** and I got my first and second eigenvectors which explained most of the data and plotted them. I saw the data was overlapping with these two components.

After looking at the initial data I used **KMeans(10)** to run KMeans algorithm and then used its .**fit\_predict** method to get the clusters predicted. This was all part of sklearn package. I then plotted again using **go.Scatter().** I then used Pandas Dataframe() constructor to construct crosstab which showed the clusters and labels. I then used sklearn package again using metrics such as **homogeneity\_score(), silhouette\_score() and completeness\_score()** as way to see how KMeans performed. I also performed KMeans directly on the dataset without PCA. I also used the same metrics.

**Exposition, problem description, and management recommendations**

For the management recommendation, **I believe** the **KMeans option that performed the best was the KMeans without PCA algorithm**. From the crosstab it showed that it tried to put each number in one cluster and one cluster only, for example. It also performed well on all three metrics that I saw with a **Homogeneity Score** from 1-17 of 0.48, a **Silhouette Score** of 0.05 and a **Completeness Score of 0.49**. Overall the takeaway from these crosstabs and metric scores is KMeans is not good at accuracy in terms of putting the label in their corresponding clusters, but it is good at find distinguishing clusters, as well as finding similarities between data instances.

If one wanted to find out which KMeans performed the best on accuracy I would say the PCA with 784 PCs performed well in accuracy although the accuracy for all the KMeans options were quite low, so I was disappointed with this. Therefore, to management **I recommend** doing **KMeans without PCA** as it did a better job at clustering. In the future I would like to learn how I can improve accuracy and actually find the correct accuracy given pixels. I wonder if other approaches can be used such as Classification like Random Forests or Neural Networks would be better to use. The metrics used are mostly for figuring out if clustering is performing and I think the completeness score gives you accuracy in sense, but it was very low as it was only performing around 0.48 which means the algorithm is clustering at random.

References

[1] Geron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Incorporated.

[2]https://sebastianraschka.com/Articles/2015\_pca\_in\_3\_steps.html#:~:text=The%20eigenvectors%20and%20eigenvalues%20of,the%20eigenvalues%20determine%20their%20magnitude.

[3] https://blog.paperspace.com/dimension-reduction-with-principal-component-analysis/#:~:text=Principal%20Component%20Analysis(PCA)%20is,a%20set%20of%20orthogonal%20axes.

[4] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.homogeneity\_score.html

[5] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette\_score.html

[6] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.completeness\_score.html