On Dimensionality Reduction Techniques

Michael Montgomery
Valdosta State University
Faculty Advisor: Dr. Jin Wang
memontgomery@valdosta.edu

Fall 2019

ABSTRACT

Dimensionality reduction is an important procedure when dealing with high dimensional data, but what is dimensionality reduction, why is it important, and how do we perform it? The aim of this paper is to describe the importance of dimensionality reduction, and how it is achieved through filtering and two techniques, Principal Component Analysis and Linear Discriminant Analysis. This paper, also, aims to explore some of the applications for these techniques through code implementations.

Keywords

Dimensionality Reduction; Principal Component Analysis; Linear Discriminant Analysis; High Correlation Filter; Low Variance Filter; Missing Value Ratio; K-Nearest Neighbor

1. INTRODUCTION

Multivariate data is common in real-world situations, but multivariate data can cause some issues when too many variables come into consideration. So, in order to handle data with high dimensions with some sort of ease the dimensions of the data need to be reduced. For this reason, a multitude of techniques have been developed to ensure that meaningful data is transformed into lower dimensions. Another way to think of dimensionality reduction is to take, let's say, a three- or four-dimensional object and project that object into two dimensions or any number of lower dimensions than the original object. For example, a sphere, a 3-dimesional object can be projected into a 2-dimensional space, and that object would look like a circle. Also, we can view a four-dimensional cube, commonly known as a tesseract, in a three-dimensional space due to an orthogonal projection into that space.

2. Why is dimensionality reduction important?

Data with high dimensionality is common in real-world situations, so there is a need to develop ways of handling data at higher dimensions. There are several reasons for reducing dimensions. One such reason is that the reduction of dimensions helps reduce the amount of storage needed to store the data. Another is that dimensionality reduction will reduce the amount of time needed for computation. Another reason is that when dealing with higher dimension data it becomes increasingly difficult to visualize and interpret data. So, by reducing dimensions we reduce the ambiguity of the data set, and we create an environment friendly for some mathematical techniques that become problematic at higher dimensions.

High dimensionality data, typically, has redundant dimensions, so that means that this attribute can be exploited and that some variables can be eliminated by explaining them as a combination of other variables. Dimensionality reduction exploits structure and correlation for that data with high dimensionality typically possesses an intrinsic lower-dimensional structure [4].

3. Feature Selection as a way of reducing dimensionality

Feature selection, in terms of filtering data, is an efficient way to eliminate redundant data from a data set. There are a multitude of ways to pick and choose the data that needs to stay and the data that needs to go. This paper explores a few methods such as High Correlation filters, Low Variance Filters, and Missing Value Ratio.

3.1 High Correlation Filters

High Correlation Filters evaluates the correlation between two columns in a dataset to determine if one variable or the other should be removed altogether. One popular method to determine the correlation between the two variables is the Pearson's Correlation. Pearson's Correlation was developed by Karl Pearson in the early twentieth century

The Pearson's Correlation measures the strength and direction of linear association between two continuous variables and is defined as:

$$r = \frac{cov(X,Y)}{std(X)*std(Y)}$$

Where X and Y are the two variables in question. Pearson's Correlation coefficient is defined as r, and there some observations that can be made from the value that is assigned to r. The value of r is always between [-1,1], or $-1 \leq r \leq 1$, where the closer to -1, including having the value of, r is the more negatively correlated the X and Y are. The closer to 1, including having the value of, r is the more positively correlated X and Y are. If r is 0 then there is no linear correlation between X and Y. Furthermore, the Coefficient of Determination can be calculated as the square of r, or r^2. When the Coefficient of Determination is represented as a percentage it represents the percent of variation in the dependent variable that can be explained by the independent variable. [1][2]

High Correlation filters typically goes through data columns and use these methods to determine if the two columns, X and Y, are linearly correlated. Typically, there is a threshold that is adhered to determine if one of the columns are to be thrown out of the dataset. The threshold is determined by the user applying the filter, so if the determined threshold is eighty percent then anything above eighty percent correlation will be removed. This method effectively removes data with high correlation for that if two columns have a high correlation then the columns can be explained by the other.

3.1.1 *Code Implementation*

```
library('MASS')
# Assign value of r
r = 0.80
# simulate random data
```

```
#https://www.rdocumentation.org/packages/
rockchalk/versions/1.8.144/topics/mvrnorm
data = mvrnorm(n=500,
               mu=c(0,0),
               Sigma=matrix(c(1,r,r,1),
               nrow = 2),
               empirical = TRUE)
# Assign the X and Y variables
X = data[,1]
Y = data[,2]

plot(X,Y, main="Correlation: r = 0.80")


correlation <- function(X,Y){
  return((cov(X,Y)/(sd(X)*sd(Y))))
}
correlation(X,Y)
# outputs 0.80
```

The code above uses the R programming language to simulates a set of random variables with a Pearson's Correlation, *r*, value set to 0.80. Graph 3.1, located in the appendix, shows a positive correlation between two data columns, while graph 3.2 shows no two data columns with no linear correlation, and graph 3.3 shows two data columns with negative data columns. The filtering process will fall into O(n^2) time complexity to compare each column to every other column in the dataset.

3.2 Low Variance Filters

Low Variance Filters evaluate the amount of variance a column has in a dataset. Like in the High Correlation Filters, there is typically a specified threshold in which the data column can have before it is removed. The idea behind the low variance filter is that if a column has little to no variance then it doesn't sufficiently contribute meaningful information to the dataset overall. Therefore, the data column can be removed or ignored.

Variance can be defined as:

$$\sigma = \frac{\sum(x-\mu)^2}{N}$$

Where μ is the mean of the elements in the column, N is the number of elements in the column and x is each individual element. The variance is the summation of the mean subtracted from each element squared divided by the number of elements.

3.2.1 Code Implementation

```
from sklearn.feature_selection import
VarianceThreshold
x = [[100,3,2,5], [100,3,22,4],
[100,3,78,85], [100,3,43,65],
[100,3.5,85,52]]

low_variance_filter =
VarianceThreshold(threshold=0.1)
low_variance_filter.fit_transform(x)


array([[ 2.,  5.],
       [22.,  4.],
       [78., 85.],
       [43., 65.],
       [85., 52.]])
```

The code above uses python package called sklearn's variance filter class to implement a low variance filter. The low variance filter takes in a specified percentage in which the data's columns cannot be below that ratio or it is removed from the dataset. Above we run the filter on a 5X4 matrix to determine if the variance in the column is below ten percent and if the column is below ten percent then it is removed from the dataset.

3.3 Missing Value Ratio

Missing Value Ratio filters are useful in removing columns from a dataset. The idea behind the missing value ratio is that in real world data there will most likely be columns with missing values, so if a threshold for acceptance is specified then certain columns can be removed or ignored. The missing value ratio can be defined as:

$$Missing\ Value\ Ratio\ (\%) = \frac{Number\ of\ Rows\ Missing}{Total\ Number\ of\ Rows} \times 100$$

The filter is used on each column of the data set to determine the percentage of the column that is missing. If the column is missing above the specified amount, then that column will be removed or ignored.

3.3.1 Code Implementation

```
import numpy as np

x = [[None, 215,232,354,None,654],
     [None, 215,None,354,456,654],
     [None, 215,232,354,None,654],
     [None, 215,None,354,456,654],
     [None, 215,None,354,456,654],
     [None, 215,232,354,456,None]]
count = 0
ratio = 0.5
elim_columns = []
for i in range(len(x)):
  for j in range(len(x)):
    if(x[j][i] == None):
      count+=1

  if(count >= (len(x)*ratio)):
    elim_columns.append(int(i))
  count = 0

for i in range(len(elim_columns)):
  [k.pop(i) for k in x]
print(x)

[[215, 354, None, 654],
[215, 354, 456, 654],
[215, 354, None, 654],
[215, 354, 456, 654],
[215, 354, 456, 654],
[215, 354, 456, None]]
```

The above code implements a missing value ratio filter using base python. The filter eliminates columns that has less than the specified ratio of values in each column. In the code above the specified ratio amount is fifty percent, so columns one and three were eliminated from the dataset. This is filter is another way to eliminate features from a dataset for that these columns would not convey much useful information.

4. Principal Component Analysis (PCA) & Linear Discriminant Analysis (LDA)

4.1 PCA

Principal Component Analysis has been a known technique for over one hundred years. This technique was proposed by Pearson in 1901, and later extended and improved upon by Hoteling in 1933. This technique is one of the widely used techniques for reducing the dimensions of high-dimensional data and lossy data compression. Principal Component Analysis has application in a multitude of fields, notably, Computer Vision is a major field of interest in applying Principal Component Analysis for uses in image compression and pattern recognition, i.e. facial recognition software.

Principal Component Analysis finds a projection of some set of data points, *x*, such that the new set of *x*'s are as similar as possible to the original data set but has a lower intrinsic dimensionality than the original [7]. Principal Component Analysis is a way of finding patterns in data and then expressing that data as a combination of the most important features in that dataset.

4.1.1 Covariance Matrix

Typically, the first step in Principal Component Analysis is to standardize, center, the data. This ensures consistency in the data, and it does not affect the covariance matrix.

$$\mathrm{cov} = [\sigma(x,x)\ \sigma(x,y)\ \sigma(x,z)\ \sigma(x,y)\ \sigma(y.y)\ \sigma(y,z)\ \sigma(x,z)\ \sigma(y,z)\ \sigma(z,z)\ ]$$

The covariance matrix has interesting properties. The main diagonal of the covariance matrix is simply the variance of the different variables in the dataset, and every other position in the matrix is the covariance of each variable to each other because of this property the covariance matrix is symmetrical. Finding the covariance between each variable is important for that the covariance will determine if the two variables in question are positively or negatively correlated and if that relationship is weak or strong depending on if the variable is positive or negative and how large the value is itself.[5][6]

4.1.2 Eigenvectors and Eigenvalues

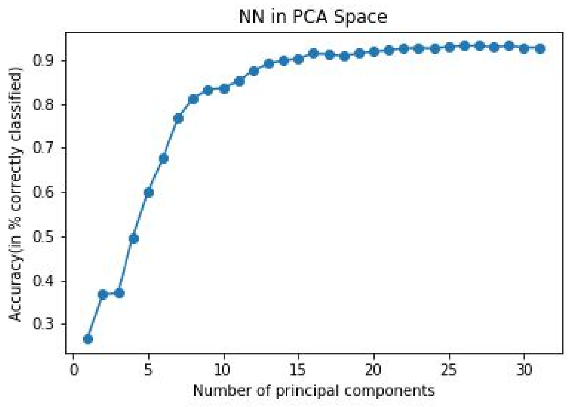
The eigenvectors and eigenvalues of the covariance matrix can be found, and they provide useful information. Eigenvectors and eigenvalues are the fundamental backbone of Principal Component Analysis. Eigenvectors can be thought of as a summary, or direction, of the dataset (matrix), and the eigenvalues can be thought of as a magnitude, or how important a certain vector is to the dataset.

Finding the eigenvectors and eigenvalues of the dataset's covariance matrix explains some useful information. By finding the eigenvectors and values of the covariance matrix we are able to determine how important certain features are within the dataset. Principal Component Analysis relies on finding the importance of a certain number of features, and this is done by finding the eigenvalues of each column, or feature. Columns with higher eigen values represent features with higher variance, and therefore most likely higher importance in respect to the feature having higher importance in describing the dataset. These columns are referred to as the principal components of the dataset [5]. While the eigenvectors describe the how the columns are swaying the data in particular direction.

Principal Component Analysis's algorithm relies on ordering the columns by in descending order to ensure that the first few columns correspond to the columns with the highest eigenvalues. Then the algorithm chooses the top *N* columns, where *N* is specified by the user. These columns are what used to represent the dataset in smaller dimensions than the original. Therefore, reducing the dimensions of the dataset and/or compressing the image when used in computer vision. [5][6]

4.1.3 Code Implementation

```
test_size = 1000
train_size = 10000
number_of_pcs = 32
xtest = xtest_arr[:test_size,:]
xtrain= xtrain_arr[:train_size,:]
ytrain = y[:,:train_size][0]

x_zero = xtrain - np.mean(xtrain,axis=0)
xt_zero = xtest - np.mean(xtest,axis=0)

[u,s,v] = np.linalg.svd(x_zero)
pcs = v.T



for components in
range(1,number_of_pcs):
  x_projected = np.matmul(x_zero, pcs[:,
:components])
  test_projected = np.matmul(xt_zero,
pcs[:, :components])
  count = 0
  for i in range(test_size):
    dist = []
    for j in range(train_size):

dist.append(euclidean_distance(x_projecte
d[j,:],test_projected[i,:]))
    dist = np.array(dist)
    minimum = np.argmin(dist)

    if(ytrain[minimum] == y_test[0][i]):
      count = count + 1


comps.update({components:count/test_size}
)
```



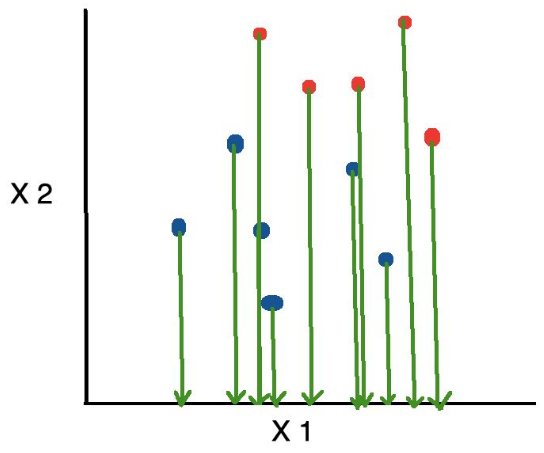
The code above displays an implementation of Principal Component Analysis written in Python, an interpreted general programming language that has many packages for statistical analysis much like R. The code uses Singular Value Decomposition. Singular Value Decomposition and finding the eigenvalues of the covariance matrix achieves practically the same outcome, but Singular Value Decomposition is more general.

The code above is an implementation of Nearest Neighbor in Principal Component space. The dataset used is the MNIST database of handwritten digits, and the Nearest Neighbor technique was tested on one thousand samples over a sample of ten thousand training samples. Each picture of handwritten digit is a 28X28 grid of pixels, represented by RGB value. Each picture is reshaped into a 784-column row. So, each picture now has 784 dimensions of pixel. The graph above was implemented using matplotlib, a python package for creating graphs, etc. Using the graph, we observe that if we run Principal Component Analysis on the pictures, we can efficiently describe the pictures with roughly fifteen principal components. Therefore, we can reduce the dimensions of each picture safely to roughly fifteen.

4.2 LDA

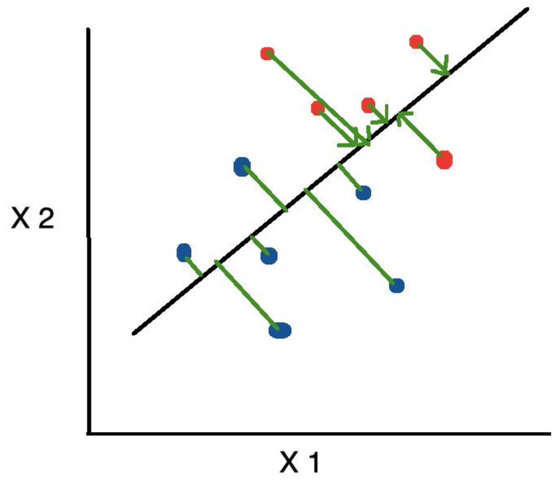
Linear Discriminant Analysis, like Principal Component Analysis, aims to reduce dimensionality in large dimensional data, but it achieves this by preserving as much class discriminatory information as possible because of this it is mostly associated with high-dimensional data with a large amount of classifications. Linear Discriminant Analysis aims to reduce dimensionality by projecting the data into a lower-dimensional subspace like Principal Component Analysis.

Linear Discriminant Analysis aims to obtain a transformation of the features into a lower subspace. For example, lets observe a two-dimensional dataset. The objective would be to transform the data into a one-dimensional space, but the objective is also to keep as much separation of the classes as possible. So, in order to do this, in most cases, a new axis would need to be defined in order to maximize the separability of the classes.



[3]

The above picture shows a projection of two-dimensional data onto the *X1* axis. This projection does not provide much useful information with respect to the two different classes. However, if we define a new axis and project the data onto it, as seen below, then we can preserve useful classification information, but how do we do that?



[3]

4.2.1 *Mean Vectors and Scatter Matrices*

In order to find a good projection vector, we need to calculate the mean vectors of each class in the dataset. The mean vectors consist of the mean column values of all entries of a particular class. So, all entries of class one will provide the mean data columns for the mean vector of class one and so forth. Once the mean vectors are found then a scatter matrix needs to be found. The scatter matrix is found for each class within itself and a scatter matrix is found between each class. Alternatively, we can compute the covariance matrices for these instead. The within class scatter matrices compute the variance between each feature within the class, and the between class variance does the same but comparing the covariance between features of different classes.[3][4][8]

4.2.2 Eigenvectors and Eigenvalues

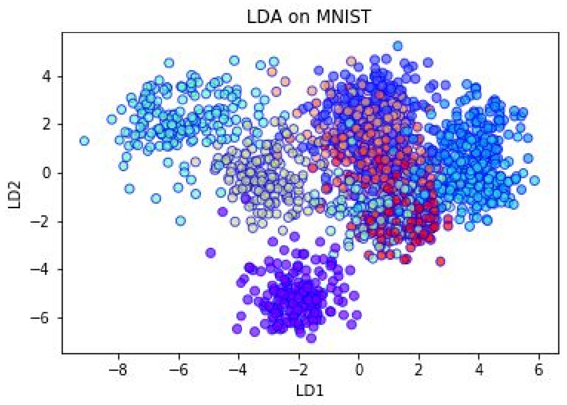
Linear Discriminant Analysis, much like Principal Component Analysis, relies on eigenvectors and eigenvalues to determine the linear discriminants for the new feature subspace. Remembering that eigenvectors describe direction of a particular column, or feature, and the eigenvalues describe the magnitude of the vector, then we can find the eigenvalues and eigenvectors for the scatter matrices. Once these are found we can reconstruct the data into a smaller subspace using the largest eigenvalues.

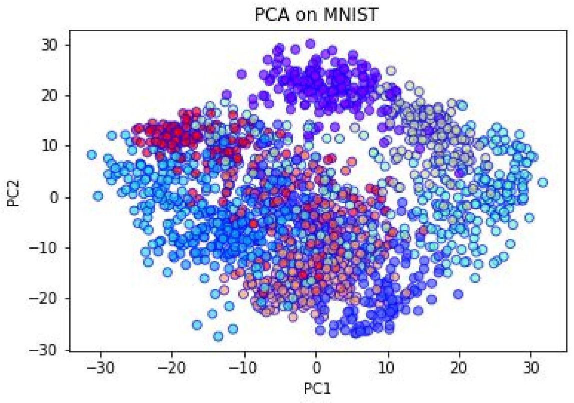
Like principal component analysis we find the eigenvalues and we order the vectors in descending order to make a new matrix. Then we can reconstruct the data in a way that preserve classification information.[3][4][8]

4.2.3 Code Implementation

```
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis
import LinearDiscriminantAnalysis
from sklearn.decomposition import PCA
from google.colab import files
from sklearn.datasets  import
load_digits
mnist = load_digits()
x = mnist.data
y = mnist.target
lda = LinearDiscriminantAnalysis()
lda_transform= lda.fit_transform(x, y)
pca = PCA()
pca_transform = pca.fit_transform(x,y)
```

The code above uses the MNIST database of handwritten digits imported by a python package called sklearn. Sklearn is a python package that provides tools for data mining and data analysis. Sklearn has implementations of Principal Component Analysis and Linear Discriminant Analysis already built, and for this section of the project I will be using sklearn to run Linear Discriminant Analysis, and I used sklearn's Principal Component Analysis class to use as a comparison between the two methods.





The two photos above were generated using a python package called matplotlib. The top photo is a plot of the first two linear discriminants, and the bottom is a plot of Principal Component Analysis run on the MNIST database but using two principal components. Though this dataset should render the use of more linear discriminants and principal components, the two graphs tells us a story. Linear Discriminant Analysis transformed the data into two-dimensions, and it preserved the classifications. This is evident from the clustering of the data. Principal component Analysis transformed the data into two-dimensions, but it did virtually nothing with respect to the classification. The graph is scattered around.

5. K-Nearest Neighbor & Other Applications

The K-Nearest Neighbor technique may not be directly involved in dimensionality reduction, but the technique is one of the most widely used techniques for classification. Therefore, it should warrant a discussion with respect to dimensionality reduction. The K-Nearest Neighbor technique entails finding the closest *K* points to a point and determining its classification from the classifications of the *K* points closest to it. This is very simple to visualize in a two or three-dimensional space. However, in high dimensionality calculating the closest points become problematic. This is because the application of Euclidean distance is problematic with higher dimensions. Therefore, dimensionality reduction is typically done prior to performing K-Nearest Neighbor. Earlier in the paper we can see the use of Principal Component Analysis's reduction of dimensions prior to the use of K-Nearest Neighbor. Not only does that make K-Nearest Neighbor realistically usable, it reduces computation time and storage while preserving the information needed to perform K-Nearest Neighbor.

6. Summary & Conclusions

High-dimensionality datasets are a commonplace in the real world, and handling data with high-dimensionality is problematic for several reasons. Dimensionality reduction helps solve these problems by improving computation time, reducing required storage space and reducing the ambiguity of higher dimensional data. This paper explained two techniques, Principal Component Analysis and Linear Discriminant Analysis, and some of their applications. Linear Discriminant Analysis is useful when we want to preserve information about the classifications, where Principal Component Analysis essentially uses the features with the greatest magnitudes of importance, eigenvalues, to describe the dataset. These two techniques are among two of the most popular way to reduce dimensionality. Along with these two techniques we explained a few simple, yet effective, filters, High Correlation Filters, Low Variance Filters, Missing Value Ratio Filters, to help eliminate columns from higher dimension datasets. These techniques provide useful ways to eliminate dimensions from higher dimensional data.

7. REFERENCES

[1] Anon. SPSS Tutorials: Pearson Correlation. Retrieved November 24, 2019 from https://libguides.library.kent.edu/SPSS/PearsonCorr

[2] Barbara Illowsky and Susan Dean. 2013. *Introductory Statistics*, Houston, TX: OpenStax.

[3] Cory Maklin. 2019. Linear Discriminant Analysis In Python. (August 2019). Retrieved December 1, 2019 from https://towardsdatascience.com/linear-discriminant-analysis-in-python-76b8b17817c2

[4] Elhabian, Shireen, and Aly A. Farag. University of Louisville, CVIP Lab, http://www.sci.utah.edu/~shireen/pdfs/tutorials/Elhabian_LDA09.pdf.

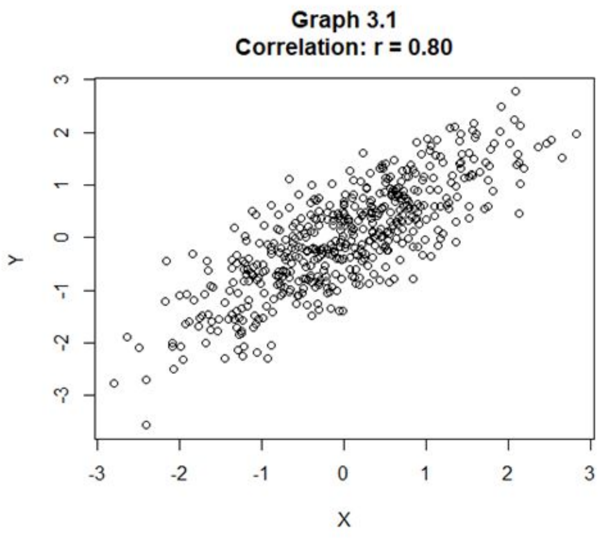
[5] Lindsay I. Smith. 2002.(February 2002). Retrieved November 26, 2019 from http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf

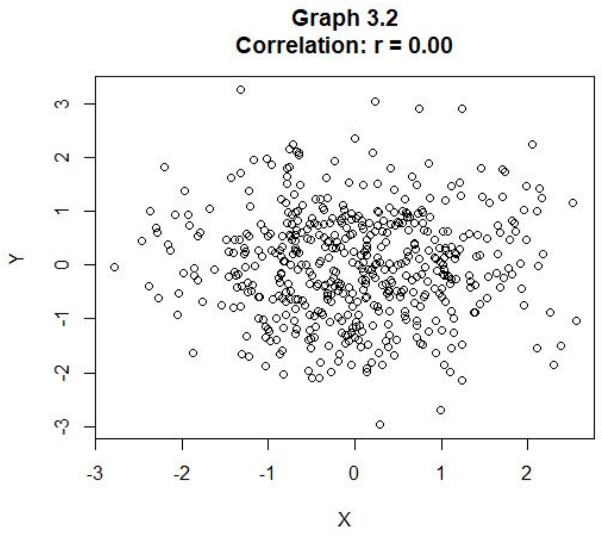
[6] Matt Brems. 2019. A One-Stop Shop for Principal Component Analysis. (June 2019). Retrieved November 26, 2019 from https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c

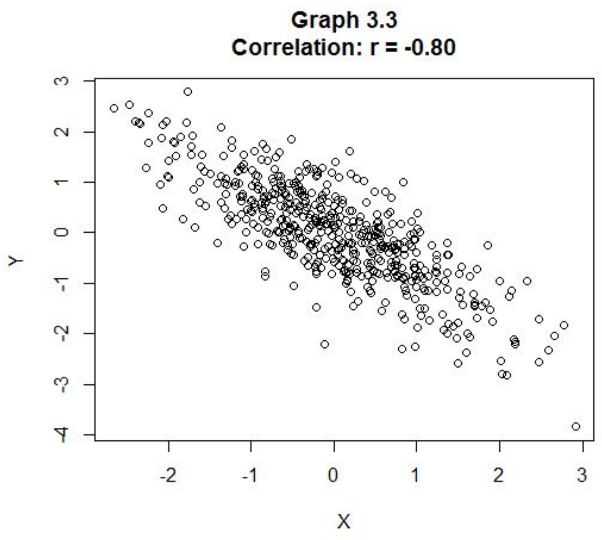
[7] Marc Peter Deisenrith, A.Aldo Faisal, and Cheng Soon Ong. 2019. *Mathematics for Machine Learning*, Cambridge University Press. pp. 317-318.

[8] Sebastian Raschka. 2014. Linear Discriminant Analysis. (August 2014). Retrieved December 1, 2019 from https://sebastianraschka.com/Articles/2014_python_lda.html

8. Appendix


Graph 3.1
Correlation: r = 0.80
Y
X
-3
-2
-1
0
1
2
3


Graph 3.2
Correlation: r = 0.00
Y
X
-3
-2
-1
0
1
2
3


Graph 3.3
Correlation: r = -0.80
Y
X
-4
-3
-2
-1
0
1
2
3