CSE 572: Data Mining (Spring 2020)

Project 1
Feature Extraction from CGM Data

Submitted to:

Professor Ayan Banerjee
Ira A. Fulton School of Engineering

Arizona State University

NAME:

Arth Dave(adave3@asu.edu) [1217077082]

February 12, 2020

INDEX

1. Introduction- PageNo:-3

2. Feature Extraction PageNo:-3
    1. Fast-FourierTransform
        1.1 Intuition
        1.2 Explanation of FFT figure with the algorithm.
    2. Polynomial Fit
        2.1Intuition
        2.2Explanation of PFT figure with the algorithm.
    3. Moving_Average
        3.1Intuition
        3.2Explanation of FFT figure with the algorithm.
    4. RootMeanSquare
        4.1Intuition
        4.2Explanation of FFT figure with the algorithm.

3. Feature Matrix PageNo:- 12

4. PCA PageNo:- 13

5. PCA Analysis(reason for choosing the 5 components)-PageNo:-14

1. Introduction

The project aims to predict the meal timing of 5 diabetic patients in order to monitor insulin so that the glucose level is kept in place. We do the above by taking a data model which takes input and predicts the time at which the glucose shall be provided. The data is collected from some tool which will receive glucose readings every 5 minutes for 2.5 hours. We do this by doing the analysis of the CGM data and extract necessary information from it.

There are 4 feature extraction methods which have been selected:

1. Fast Fourier Transform
2. Polynomial Fit
3. Moving Average
4. Windowed Root Mean Square

Thus, by using these 4 features we do the next implementation.

2. Feature Extraction

In order to extract features from the data, we clean the data first. This is done in the code under the function called pre-processing. There are many ways to deal with missing values. The method used in the code was to first append the files and then use interpolation on both CGMvalues and CGMTimeseries. But, this did not clear all the missing value so the remaining were also to be handled. Using, the dropna function we removed the remaining missing values as well. Thus, after clearing the data the feature extraction method started. Feature extraction helps in getting rid of the large and unnecessary data by combining variables into different features. This will help in using the important data and ignoring the irrelevant large data.

1. Fast Fourier Transform

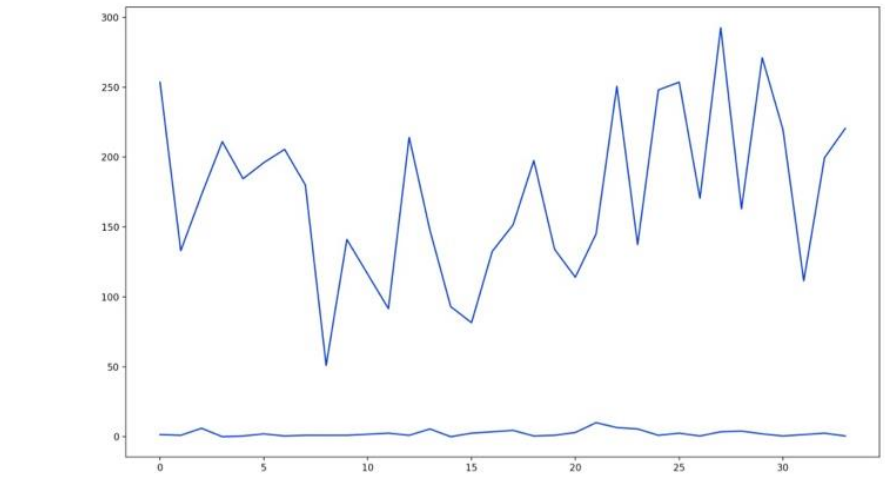
1.1 Intuition :-

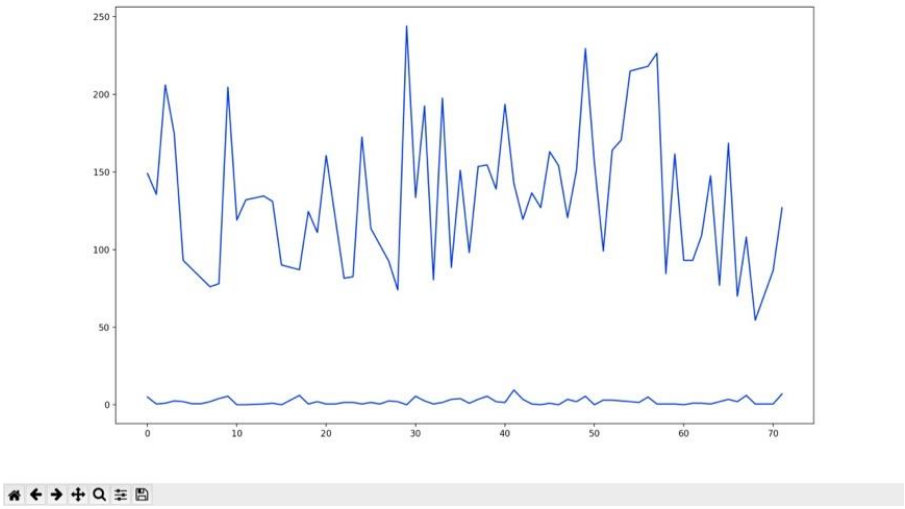
Fast Fourier Transform is a method which converts a space or time signal to signal of the frequency domain. Fast Fourier transform also rapidly computes by factorizing the DFT matrix as the product of sparse factors.

```
# 1st Feature
def fourier_transform(Cgmvalues,CGMTimeseries):
    fourier_transform=list()
    for i in range(5):
        person_ft=list()
        for r in range(len(Cgmvalues[i])):
            temp=np.fft.irfft(Cgmvalues[i][r],n=2)
            #print(temp)
            person_ft.append(temp)
        fourier_transform.append(person_ft)
    print("fourierTransform-Completed")
    return np.array(fourier_transform)

def fftplot(FFT):
    graph_FFT = pd.DataFrame(FFT[3])
    graph_FFT = graph_FFT.apply(abs)

    plt.plot(graph_FFT, color="blue")
    plt.show()

    graph_FFT = pd.DataFrame(FFT[2])
    graph_FFT = graph_FFT.apply(abs)

    plt.plot(graph_FFT, color="blue")
    plt.show()
```

1.2 Explanation of FFT figure with the algorithm.

- The following are the plots for the FFT extraction method.





- The function takes values of both CGMvalues and CGMtimeseries.
- As, we can see in both the graphs respective to different subjects in the first graph we see 50 to almost 300, respective to the time series, this shows how well the feature gives the variance as there is not straight line in the graph.
- Now, for each value in the array we use the algorithm of FFT which is np.fft.irfft(Cgmvalues[][],n)
- Using this algorithm for out data we plotted the above figures of FFT.
- First figure is for the 3rd subject's CGM value and the second figure is for the 4th subject's CGM values.

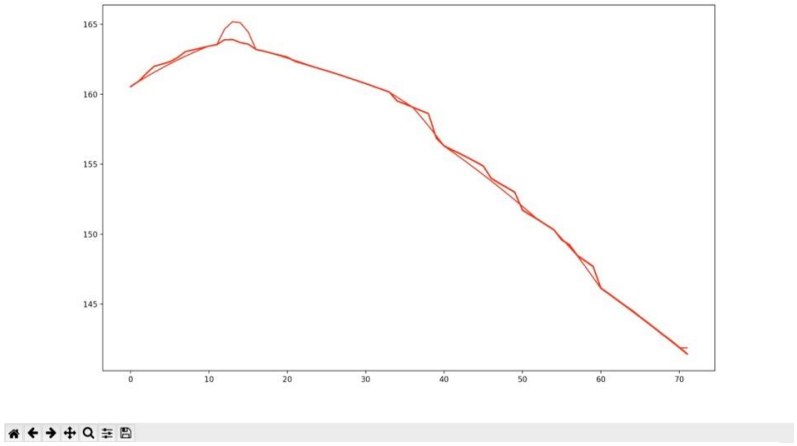
2. Polynomial Fit

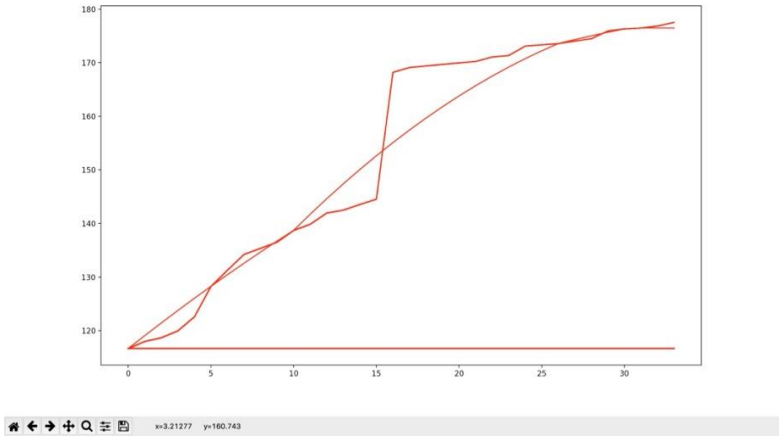
2.1 Intuition

We use polynomial fit to represent any complex non-linear relationship provided by the polynomial models. It can take various parameters like deg, rcond, full, cov, etc. Through this method we construct a mathematical equation that fits best for the time series data and represents the CGM curve accurately. Thus, we use this to understand the relationship among CGM glucose level and time. Also the order of the data is important as the lower order might not fit correctly.

```
# 2nd feature

def polyFit(Cgmvalues,CGMTimeseries):
    polylist =[]
    for i in range(5):
        yval = Cgmvalues[i].flatten()
        xval = CGMTimeseries[i].flatten()
        polylist.append(np.polyfit(xval,yval,2))
    print("polyFit running")
    return np.array(polylist)


def pltpft(PFT):
    #%matplotlib inline
    graph = np.polyval(PFT[3], CGMTimeseries[3])
    plt.plot(graph, color="red")
    plt.show()
    print("plot running")
```

2.2 Explanation of PFT figure with the algorithm





- The PFT function takes both CGM values and CGMtimeseries as data for further processing.
- When we look at the plots, we can see that similar curves have coefficients in a similar range when compared to curves with a different trend in CGM glucose. Therefore, when PCA is performed on this data, it may give a higher variance and hence is a good feature extraction method to consider.
- So if we take value 737225.58068287 which is the value of row1, column , we will get result as 121.45643. As we can see that for the first graph, the value goes on

decreasing and for the other subject, the graph is in a increasing pattern. This shows that this feature will provide great variance for the PCA.

- The algorithm uses the flatten method i.e CGMvalues[].flatten()
- Next step is to use the np.polyfit(x,y,n)
- Doing this, we will be extract the polyfit feature and will get the necessary information from the raw data.

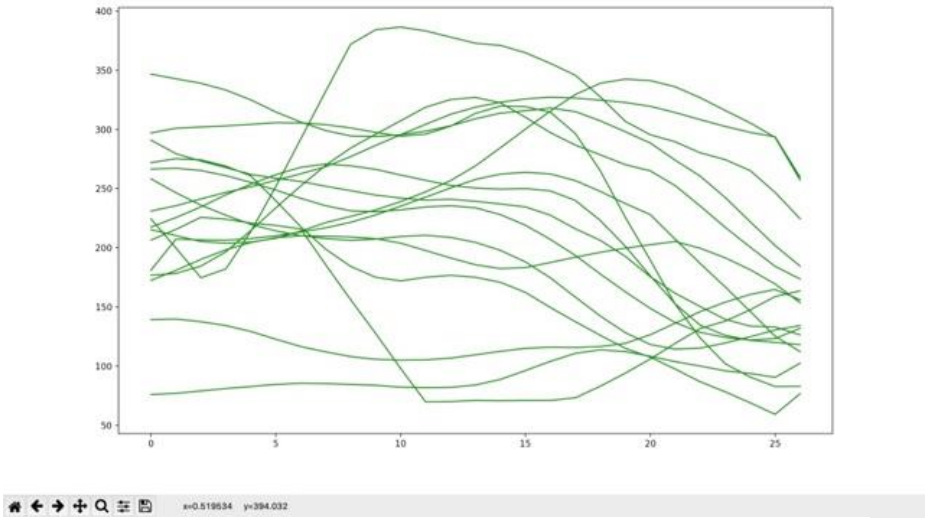
3. Moving Average

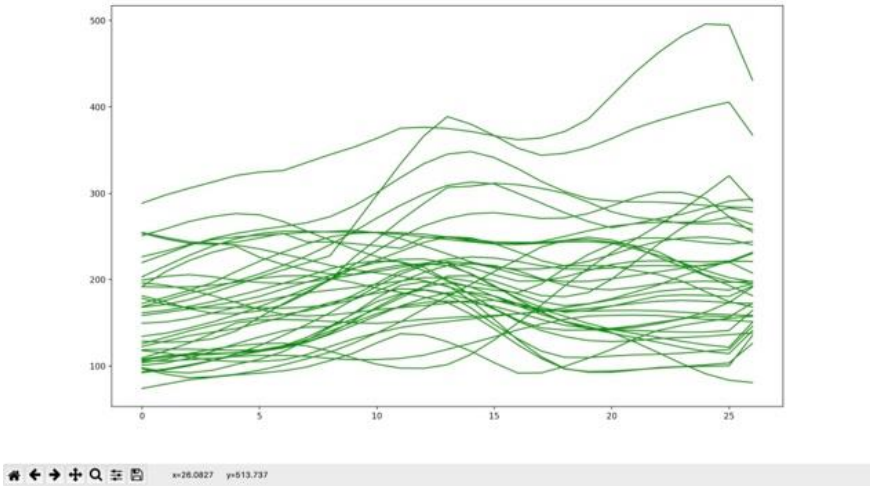
3.1 Intuition

A moving average is also called a rolling average which is used to analyze the time-series data by calculating averages of different subsets of the complete dataset which is our case is the CGMvalues and CGMtimeseries. The first moving average is calculated by averaging the first fixed subset of numbers and then we move forward to the next set of numbers. It captures the short-term fluctuation while focusing on longer trends. Thus, in general moving average smoothens the data.

```
#3rd feature
def MOV(Cgmvalues,CGMTimeseries):
    mov = []
    for i in range(5):
      personCgmvalues = Cgmvalues[i]
      personCGMTimeseries = CGMTimeseries[i]
      person_MOV = []
      window = 4
      for row_i in range(len(personCgmvalues)):
        person_MOVpoint = []
        for col_i in range(window, len(personCgmvalues[row_i])):
          person_MOVpoint.append(
              sum(personCgmvalues[row_i][col_i - window : col_i + 1]) / windo
        person_MOV.append(person_MOVpoint)
      mov.append(person_MOV)
    return np.array(mov)


def pltMov(Mov):
    for i in range(0, len(Mov[0]), 2):
        plt.plot(Mov[0][i], color="green")
    plt.show()
```

3.2 Explanation of PFT figure with the algorithm





- The MOV algorithm takes both CGMvalues and CGMtimeseries as the data.
- The graphs in the algorithm for two random subjects from the 5 subjects provided.
- Both the graphs clearly show that the feature has been successful in generating a good variance for the further processing including we get the necessary information rather than the duplicate data.
- The algorithm takes a window to keep on moving forward.
- It takes the sum function as follows sum(CGMvalues[][]/window), now appending this to out list will give us the information we need. At the end we add the np.array().

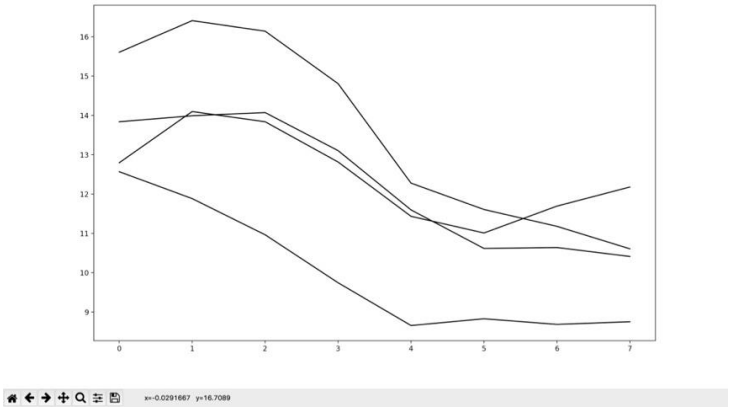
4. RootMeanSquare

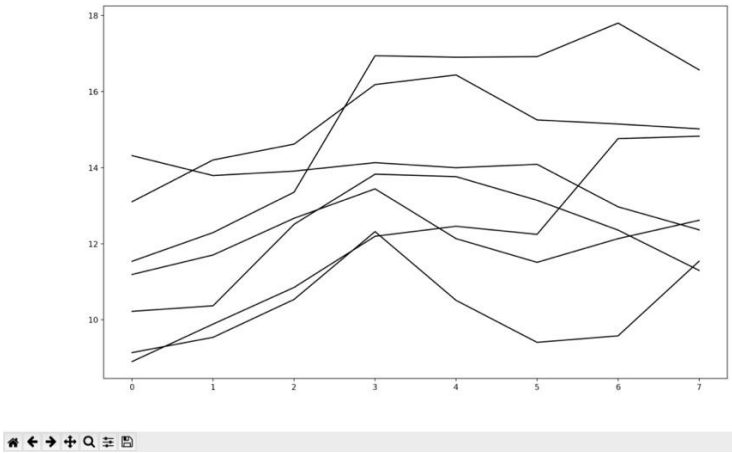
4.1 Intuition

Root Mean Square (RMS) is the square root of the arithmetic mean of square of values. It given by the below formula. Thus, it takes three basic mathematical function i.e square-root, mean and square. In this feature extraction method we set a threshold on the increase in the CGM values between two successive windows, it is possible to predict the point at which the meal consumption starts.

```
#4th feature
def rootMeanSquare(Cgmvalues,CGMTimeseries):
    rmsvelocity=list()
    for i in range(5):
        personCgmvalues=Cgmvalues[i]
        personCGMTimeseries=CGMTimeseries[i]
        rmsperson=list()

        for r in range(len(personCgmvalues)):
            rmsVelWin=list()
            for c in range(0,len(personCgmvalues[i]),4):
                if c+4 <len(personCgmvalues[i]):
                    temp=sum(personCgmvalues[r][c:c+4])/4
                    rmsVelWin.append(np.sqrt(temp))
                else:
                    temp=sum(personCgmvalues[r][c:])/(len(personCgmvalues[i])-c)
                    rmsVelWin.append(np.sqrt(temp))
            rmsperson.append(rmsVelWin)
        rmsvelocity.append(rmsperson)
    print("rms runniung")
    return np.array(rmsvelocity)


def pltrms(RMS):
    for i in range(2, len(RMS[2]), 10):
        plt.plot(RMS[2][i], color="black")
    plt.show()
```

4.2 Explanation of RMS figure with the algorithm





- The RootMeanSquare takes both CGMvalue and CGMtimeseries as data.
- Her, as well in the graph we see different values for each timeseries.
- The feature successfully provides a good variance as output which is the requirement for the PCA.
- This function also creates a window for four iterations.
- First we apply the sum function to the data as sum(data[][]/)4, this is for four iterations.
- Then, we apply the square-root to it as , np.sqrt(data)
- Thus to compute every value in our data we generate two for loops and then appending accordingly in two lists which at the end results as a data in array.
- Thus, finally we do the np.array() to the final array created.

3. Feature Matrix

After the features are extracted, it can be observed that each one of them had several sub-features. This happened because of splitting time into windows, or the feature comprising of smaller features. For every feature we created a Data-frame .At the end we had 57 sub-features. We reduced it further by using the dropna function. Finally we are left with 37 sub-features which we will use for further processing. The feature matrix created was of 178 rows and 37 columns.

```
featureMatrix-Completed
<bound method NDFrame.head of           0       1       2       3   ...         44
0     321.50  320.50  317.50  312.75  ...  11.090537   9.937303   9.604686   9.556847
1     352.75  365.25  376.00  386.50  ...  18.553975  17.406895  16.201852  13.408123
2     309.25  318.50  327.75  333.50  ...  12.278029  11.608187  11.180340  10.609220
3     278.00  274.75  270.25  267.75  ...  10.173495   9.591663   9.721111   9.678154
4     175.25  179.00  183.50  188.00  ...  12.389512  11.236103  10.954451  10.551461
..       ...     ...     ...     ...  ...        ...        ...        ...        ...
173   175.00  174.75  175.00  174.75  ...  12.439855  12.175796  12.429803  13.291601
174   275.25  278.50  280.75  282.75  ...  14.177447  12.041595  12.247449  13.076697
175   405.50  405.50  403.75  397.50  ...  12.500000  10.185774   8.573214  10.614456
176   339.75  340.25  336.75  330.50  ...  13.638182  12.459936  12.288206  13.114877
177   309.50  302.00  290.25  278.25  ...  12.903488  12.124356  11.842719  12.328828

[178 rows x 37 columns]>
```

The code for feature matrix is as follows:-

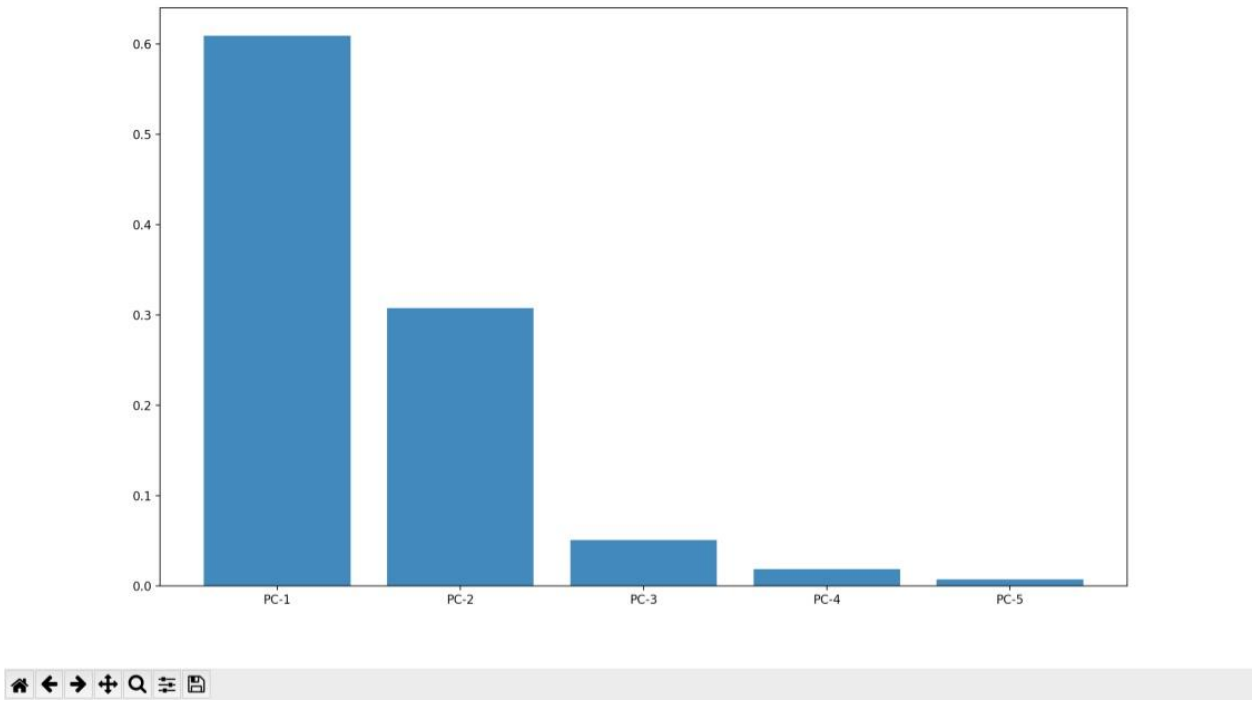
```
def featureMatrix(PFT,Mov,FFT,RMS):
    feature_MOV=pd.DataFrame()
    for personLOC in Mov:
        feature_MOV = feature_MOV.append(personLOC, ignore_index=True)
    feature_FFT = pd.DataFrame()
    for personFFTS in FFT:
        feature_FFT = feature_FFT.append(personFFTS, ignore_index=True)
        feature_FFT = feature_FFT.apply(abs)
    featureRMS = pd.DataFrame()
    for personRMS in RMS:
        featureRMS = featureRMS.append(personRMS, ignore_index=True)
    featurePolynomial = pd.DataFrame()
    for personPoly in PFT:
        polynomial_person_series = pd.Series(personPoly)
        featurePolynomial = featurePolynomial.append(polynomial_person_series, ignore_index=True)
    featurematrix = pd.concat((feature_MOV, feature_FFT, featureRMS, featurePolynomial), axis=1, ignore_index=True)
    print(featurematrix.shape)
    featurematrix = featurematrix.dropna(axis=1)
    print(featurematrix.shape)
    print("featureMatrix-Completed")

    return featurematrix
```

4. Principal Component Analysis(PCA)

After all the features are selected the next step to avoid overfitting in these high dimensional spaces. Principal Component Analysis is a method that is used to prevent the curse of dimensionality by extracting only features that show high variance in the data set. High correlated variables are abstracted from the feature vector to form a significant set of variables known as principal components which define the variance of the data set.

There are several steps involved in the process:

- Normalizing data
- Calculating the covariance matrix
- Computing eigen vectors and eigen values.
- Choosing components and forming feature vector.
- Principal Components



Now, after doing this we got 5 principal components which were the top 5 ones. Thus, giving 5 X 37 matrix as PC-1, PC-2, PC-3, PC-4, PC-5.

```
explained variance ratio [0.60915731 0.30738561 0.05074609 0.01876585 0.00723745]
            0         1         2   ...        34        35        36
PC-1  0.178678  0.180269  0.181014  ...  0.005926  0.005209  0.003290
PC-2 -0.256636 -0.258069 -0.255657  ...  0.006380  0.007668  0.006091
PC-3 -0.195249 -0.190843 -0.187214  ... -0.000179 -0.008116 -0.008471
PC-4  0.255977  0.195322  0.124906  ...  0.007050 -0.005063 -0.008768
PC-5 -0.262643 -0.153908 -0.044133  ...  0.009032 -0.000766 -0.011790

[5 rows x 37 columns]
```

5. PCA Analysis

- The reason why this top 5 principal components were the ones that were chosen was because these 5 had the maximum variance. As seen in the PCA graphs, the first feature going upto 0.6 , the second feature with 0.3 variance and so on. Thus, these components are in decreasing order.