Project 1 Report

DATA MINING

CSE 572: Spring 2020

Submitted to:

Dr. Ayan Banerjee

Ira A. Fulton School of Engineering

Arizona State University

Submitted by:

Abhik Kumar Dey (akdey@asu.edu)

Introduction

We have been provided with the data of 5 diabetic patients. The data contains the glucose levels every 5 mins for 2.5 hrs. during a lunch meal and there are several such time series data for the 5 patients. The goal of the project is to predict the meal timing of the patients so that insulin can be given to them to keep the glucose levels within normal range. This is implemented by preprocessing the given data sets and then extracting features from them, thus creating a feature matrix. Then we implement Principle Component Analysis (PCA) with the feature matrix to select top 5 features and create the new feature matrix.

Data Processing

The dataset for the 5 patients provided needs to be cleaned and manipulated before working on feature extraction. The data set contains a lot of redundant (blanks, NaN) data. I followed 70-30 ratio in cleaning the data(blanks/NaN). If a row has more than 70% of blanks/NaN, we are dropping the row. For the remaining 30% data, we are interpolating them using the method as polynomial and degree as 2. Below are the code snippets to implemented to interpolate and clean the data:

```
def clean_data():
    max_limit = 0.7 # Allowing 30% NaN data to interpolate, delete the row if >= 70%
    drop_row = []

    # Interpolate NaN if row has less NaN records than max_limit for cgm_series
    for i in range(len(df_glucose_pat)):
        no_of_nan_glucose = df_glucose_pat.iloc[i].isnull().sum()
        if no_of_nan_glucose > 0:
            percent_of_data = (no_of_nan_glucose/len(df_glucose_pat.iloc[i]))
            if percent_of_data < max_limit:
                df_glucose_pat.loc[i],df_date_num_pat.loc[i] = interpolate_data(df_glucose_pat.iloc[i],df_date_num_pat.iloc[i])
```

```
def interpolate_data(cgm_series_row_data,cgm_dt_row_data):
    #print(cgm_series_row_data)
    cgm_series_data_transpose = []
    cgm_dt_data_transpose = []
    glu =[]
    datetime = []

    for element in cgm_series_row_data:
        cgm_series_data_transpose.append(element)

    for element in cgm_dt_row_data:
        cgm_dt_data_transpose.append(element)

    data = {'dt': cgm_dt_data_transpose, 'glucose':cgm_series_data_transpose}
    df = pd.DataFrame(data)
    df['dt'].interpolate(inplace=True)
    datetime  = df['dt'].values.tolist()
    df.set_index('dt', inplace=True)
    df['newZVal'] = df['glucose'].interpolate(method = 'polynomial', order = 2)
    glu = df['newZVal'].values.tolist()
    return glu, datetime
```

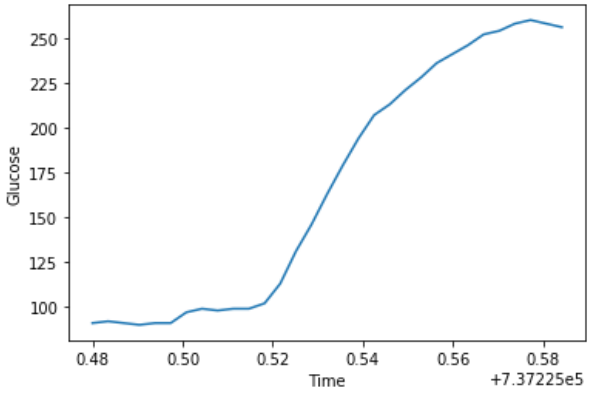
Features Extracted from the data

The below are the features extracted from the 5 pair of datasets provided for the diabetic patients:

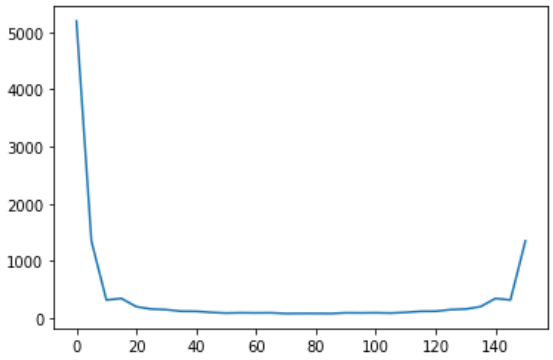
a. Fast Fourier Transformation (FFT)
b. Welch Method
c. CGM Velocity
d. Polynomial Fit

Fast Fourier Transformation (FFT)

Fast Fourier Transform samples signal over a time period and converts it into frequency domain. If we plot the given data, we get the below graph:



From the above graph it is not possible to determine when the glucose level increased rapidly. Thus, the data here needs to be sampled and hence with FFT we convert the data to frequency domain. The graph for FFT we get is as below:



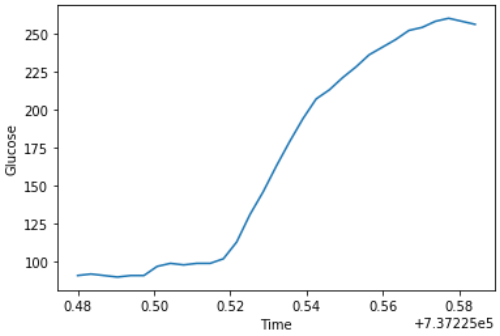
From the FFT Graph, we can identify which are the highest peak. The we can visualize the data properly than the original data. Thus, with FFT we sample the data and convert it into frequency domain, and it helps us to analyze the data more clearly.

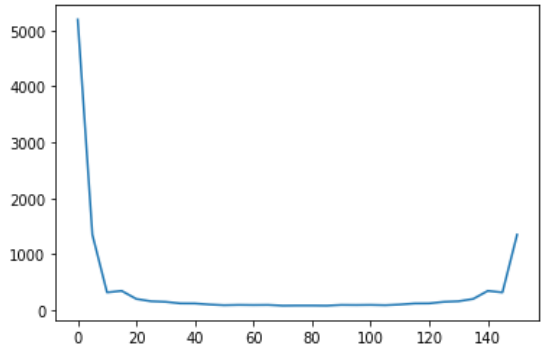
Intuition

With FFT we can analyze the data more properly. When we plot FFT w.r.t frequency, the first data that we get is always high. So, we ignore that data and select the second and third highest peak. These two peaks determine rise in the glucose levels in the patients. We select the two highest peaks so that we get the range within which the glucose level rises and as a result the patient has taken meal and needs to be administered with insulin. Thus, the two peaks are taken as feature for FFT.

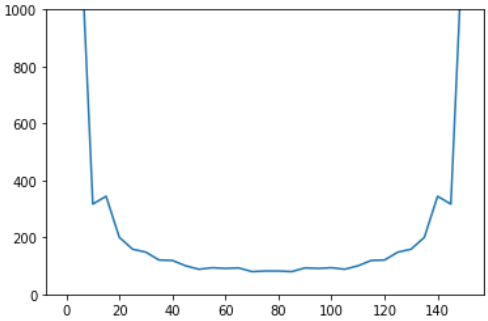
```
def calFFT():
    x = []
    df_glucose_pat['FFTPeak1'] = -1
    df_glucose_pat['FFTPeak2'] = -1


    for i in range(0,len(df_glucose_pat_cp.iloc[0])):
        x.append(i*5)

    for elem in range(len(df_glucose_pat)):
        cgm_fft_values  = abs(scipy.fftpack.fft(df_glucose_pat_cp.iloc[elem].values))
        #print (len(cgm_fft_values))
        #print (len(x))
        #cgm_dt_values = df_date_num_lunch_pat1.iloc[elem].values
        val = set(cgm_fft_values)
        val = sorted(val, reverse = True)
        #print (cgm_fft_values)
        #print (val)
        firstHighPeak = list(val)[1]
        secondHighPeak = list(val)[2]
        df_glucose_pat['FFTPeak1'].iloc[elem] = firstHighPeak
        df_glucose_pat['FFTPeak2'].iloc[elem] = secondHighPeak

        print (cgm_fft_values)
        #print (firstHighPeak,secondHighPeak)
        plt.plot(x,cgm_fft_values)#,use_line_collection = True
        plt.ylim(0,1000)
        plt.show()
```

Graph Analysis





The image on the left denotes the Glucose-time series plot for a meal and its respective FFT plot is on the right. Basically, the FFT graph if zoomed into is a mirror image. So, we can analyze the first half of the data and determine when the peaks are occurring.



FFT Values: [**5196.** **1349.45635067** 316.78745638 **344.2740833** 200.03954245
158.40828574 147.9794586 120.33043475 118.95734556 100.14516361
87.96354103 93.31047901 90.86879283 92.64708807 79.66613403
81.80853385 81.80853385 79.66613403 92.64708807 90.86879283
93.31047901 87.96354103 100.14516361 118.95734556 120.33043475
147.9794586 158.40828574 200.03954245 344.2740833 316.78745638
1349.45635067]

Data in Red is ignored as the first value always have peak value. The data in bold are the two peaks considered as attribute for FFT.

Welch Method

The Welch Method provides spectral density of the signal. The reason for using Welch Method is it reduces the noise in the signal. We will select the maximum peak in the graph which will tell us the frequency when there has been a steep rise in the glucose level. This infers that the patient has consumed meal and needs to be administered with insulin.

Intuition

The Welch method reduces noise in the spectral density and the curve resulting is smooth and gives a single peak with which we can analyze the data easily. Also, we have considered the mean and standard deviation of record with which we can check if the data is following normal distribution. The attributes for Welch Method selected are:

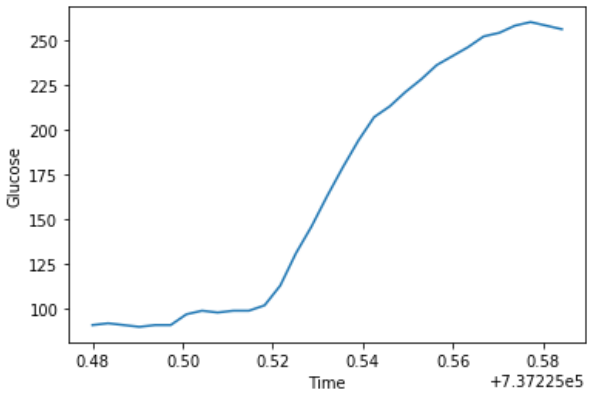
a. Maximum Amplitude

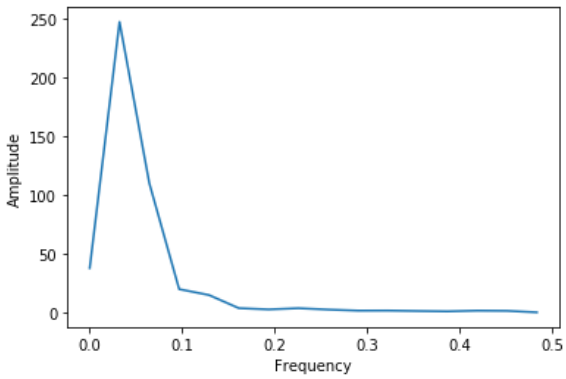
b. Mean

c. Standard Deviation.

```
#take maximum amplitude
#df_glucose_pat
def calWelch():
    df_glucose_pat['maxAmplitude'] = -1
    df_glucose_pat['stdAmplitude'] = -1
    df_glucose_pat['meanAmplitude'] = -1
    for elem in range(len(df_glucose_pat_cp)):
        v, welch_values  = np.array((signal.welch(df_glucose_pat_cp.iloc[elem].values)))
        print (welch_values)
        print (v)
#         print (max(np.sqrt(cgm_fft_values)))
#         print (np.where(cgm_fft_values == max(cgm_fft_values)))
        df_glucose_pat['maxAmplitude'].iloc[elem] = np.sqrt(max(welch_values))
        df_glucose_pat['stdAmplitude'].iloc[elem] = np.std(np.sqrt(welch_values))
        df_glucose_pat['meanAmplitude'].iloc[elem] = np.mean(np.sqrt(welch_values))


        plt.plot(v,np.sqrt(welch_values))#,use_line_collection = True
        plt.xlabel('Frequency')
        plt.ylabel('Amplitude')
        plt.show()
```

Graph Analysis:





Welch Values: [1.42901503e+03 **6.10212321e+04** 1.20894744e+04 3.99296944e+02
 2.24733385e+02 1.47329872e+01 7.62825020e+00 1.43799588e+01
 6.86771408e+00 2.99160216e+00 3.18046841e+00 2.01098437e+00
 1.24578195e+00 2.96685015e+00 2.39686874e+00 8.25317958e-02]

The value marked in bold is the maximum amplitude or the peak when the glucose level has a steep rise.

CGM Velocity

If we check the time series plot of the CGM data, we can find that there is an increase in the glucose level of the patient. Thus, the rate of increase in the glucose level is known as CGM Velocity. The velocity is calculated using window approach. The window taken is 2, that is pairing record[0] and record[2], record[1] and record[3] and so on. As the window is 2, the time interval is 10 mins. Thus, the formula for the CGM velocity is:

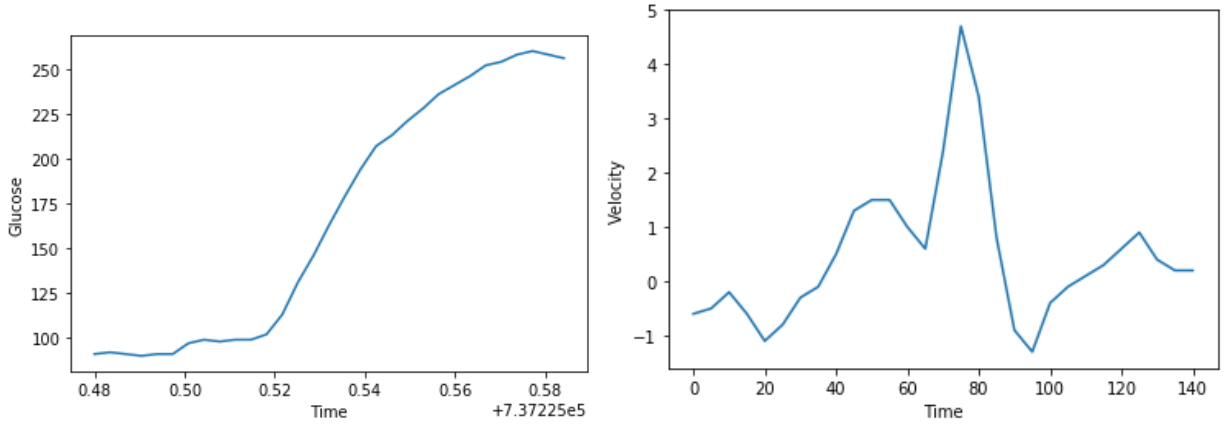
(record[i] - record[i+window_size])/time_interval

```
def calCgmVelocity():
    df_glucose_pat['meanCgmVel'] = -1
    df_glucose_pat['stdCgmVel'] = -1
    df_glucose_pat['medianCgmVel'] = -1

    time_interval = 10 # taking time interval as 10 mins
    for elem in range(len(df_glucose_pat_cp)):
        #print ('*******************************************')
        #print ('Counter - ',elem)
        window_size = 2
        velocity = []
        row_data = df_glucose_pat_cp.iloc[elem].values
        row_length = len(row_data)
        #print (row_data)
        counter = 0
        x_cgmvel = []
        for i in range((len(row_data) - window_size)):
            x_cgmvel.append(counter)
            counter += 5
            disp = (row_data[i] - row_data[i + window_size])
            vel = disp / time_interval
            velocity.append(vel)
        df_glucose_pat['meanCgmVel'].iloc[elem] = np.mean(velocity)
        df_glucose_pat['stdCgmVel'].iloc[elem] = np.std(velocity)
        df_glucose_pat['medianCgmVel'].iloc[elem] = np.median(velocity)
        print (velocity)
        #print (x_cgmvel)
        plt.plot(x_cgmvel,velocity)#,use_line_collection = True
        plt.xlabel('Time')
        plt.ylabel('Velocity')
        plt.show()
```

Intuition

The velocity is calculated for the whole duration of the meal on a day and we check for sudden rise in the data. This sudden rise infers that the patient has taken meal. We have taken the Standard Deviation, mean and median of the velocity. These 3 attributes will help to analyze if the spike is following any pattern. If mean and median becomes equal, we can say that the data follows Normal distribution.

Graph Analysis:



The image in the left shows the glucose vs time graph. The image on the right is a plot between velocity vs time. We can see that at what time interval there is a spike in the glucose level and according to that insulin needs to be administered.

Velocity : [-0.6, -0.5, -0.2, -0.6, -1.1, -0.8, -0.3, -0.1, 0.5, 1.3, 1.5, 1.5, 1.0, 0.6, 2.4, **4.7**, 3.4, 0.8, -0.9, -1.3, -0.4, -0.1, 0.1, 0.3, 0.6, 0.9, 0.4, 0.2, 0.2]

If the data is noticed, the negative value represents dip in the glucose level. The value in bold is the sudden spike in the glucose level. This infers that patient has taken meal.

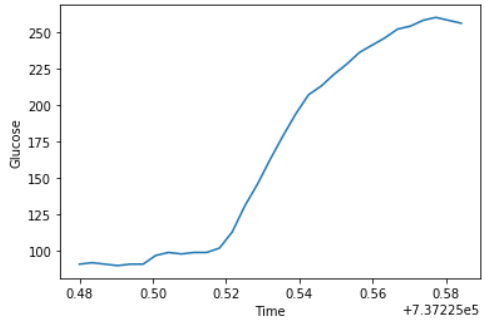
Polynomial Fit

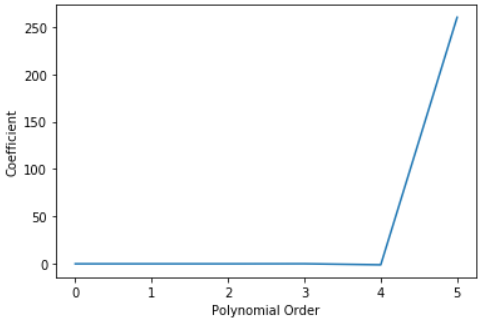
The plot of CGM vs time-series determines the relationship between glucose and time series. They exhibit non-linear relationship. Using Polynomial Fit a relation can be established between CGM and Time. The method returns coefficients which can be used to determine the relationship.

```
def calPolyfit():

    df_glucose_pat['polyCoeff1'] = -1
    df_glucose_pat['polyCoeff2'] = -1
    df_glucose_pat['polyCoeff3'] = -1
    df_glucose_pat['polyCoeff4'] = -1
    df_glucose_pat['polyCoeff5'] = -1
    df_glucose_pat['polyCoeff6'] = -1
    time_interval = [j * 5 for j in range(0, len(df_glucose_pat_cp.iloc[0]))]
    for elem in range(len(df_glucose_pat_cp)):
        polyfit = list(np.polyfit(time_interval, df_glucose_pat_cp.iloc[elem], 5))
#         print ("############################################################")
#         print (len(polyfit))
        print (polyfit)
        plt.plot(polyfit)
        plt.xlabel('Polynomial Order')
        plt.ylabel('Coefficient')
        plt.show()
        for i in range(len(polyfit)):
            col = 'polyCoeff'+str(i+1)
            df_glucose_pat[col].iloc[elem] = polyfit[i]
```

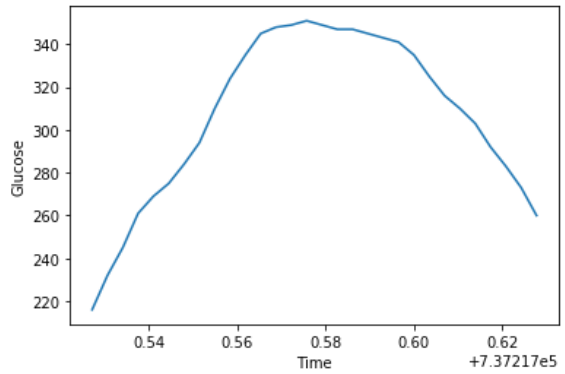
Intuition and Graph Analysis

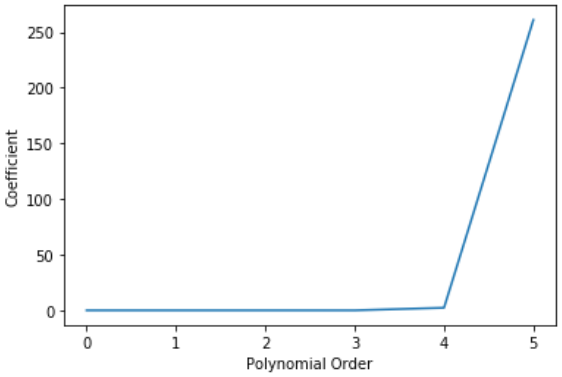
Coefficients: [-5.074237561314661e-08, 1.8320269267399997e-05, -0.0020802916076340934, 0.0704121177130023, -1.0456810332627213, 260.26210755563291]



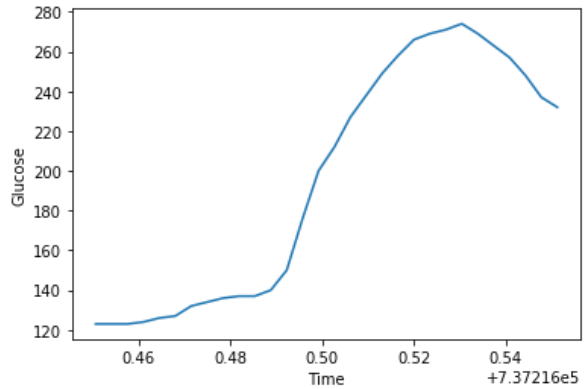


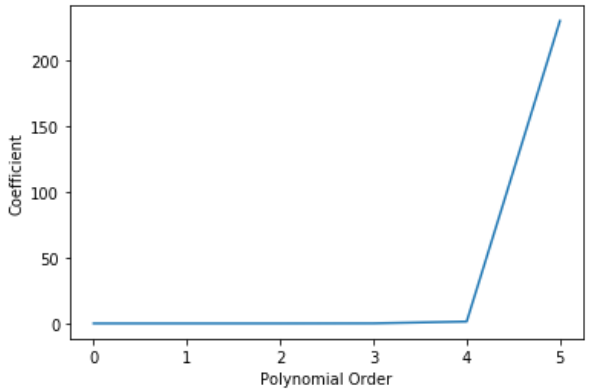
Coefficients: [1.2620021911391195e-08, -3.157046738149864e-06, 0.0001783793699811452 4, -0.01563615985350923, 2.2915394883966296, 261.0381662929101]





Coefficients: [-6.382937077172392e-08, 2.2242245892706887e-05, -0.0023445741572581686, 0.0521702595008733, 1.4126869781725415, 230.45976367086553]





From the graph between CGM vs time-series and its respective polynomial fit, it can be inferred that how different the curves be, they almost have same range of coefficients. The Polynomial Order taken is 5.

Feature Matrix after Feature Extractions

The final features selected are:

a. FFTPeak1
b. FFTPeak2
c. maxAmplitude
d. stdAmplitude
e. meanAmplitude
f. meanCGMVel
g. stdCGMVel
h. medianCGMVel
i. polyCoeff1
j. polyCoeff2
k. polyCoeff3
l. polyCoeff4
m. polyCoeff5
n. polyCoeff6

```
feature_matrix
```

	FFTPeak1	FFTPeak2	maxAmplitude	stdAmplitude	meanAmplitude	meanCgmVel	stdCgmVel	medianCgmVel	polyCoeff1	polyCoeff2	polyCoeff3	poly
0	1349.456351	344.274083	247.024760	62.466126	28.273183	1.141379	1.155353	0.8	-5.074238e-08	0.000018	-0.002080	0
1	868.543990	199.260486	165.186180	47.270907	24.550318	0.348276	1.714567	0.0	1.262002e-08	-0.000003	0.000178	-0
2	1247.374320	235.889209	241.346509	61.606202	29.222471	0.768966	1.617172	0.4	-6.382937e-08	0.000022	-0.002345	0
3	1128.069701	290.575522	218.460831	56.325630	28.697653	0.941379	1.256299	0.4	-8.438794e-08	0.000031	-0.003786	0
4	359.871194	74.397788	78.325015	21.863822	11.991005	0.096552	0.724680	-0.1	-8.604611e-09	0.000005	-0.000909	0
...	...	...	...	...	...	...	...	...	...	...	...	
189	176.269494	122.635951	30.885062	9.376581	8.335463	-0.289655	0.780515	0.0	3.361760e-08	-0.000012	0.001369	-0
190	702.648819	211.684771	143.521198	37.340731	21.031858	0.341379	1.331195	0.0	-2.059156e-08	0.000011	-0.001671	0
191	1840.682254	747.920642	304.254205	73.479792	30.247033	1.665517	1.367675	2.3	2.898310e-08	-0.000012	0.001889	-0
192	897.386585	324.014331	153.220860	37.254723	17.104393	0.689655	0.970370	1.0	1.161337e-08	-0.000004	0.000658	-0
193	607.001519	445.093525	80.846272	20.778227	15.006177	0.834483	1.038658	0.7	4.673256e-08	-0.000018	0.002413	-0

194 rows × 14 columns

We will now perform PCA with this feature matrix and select the top 5 components.

Principle Component Analysis

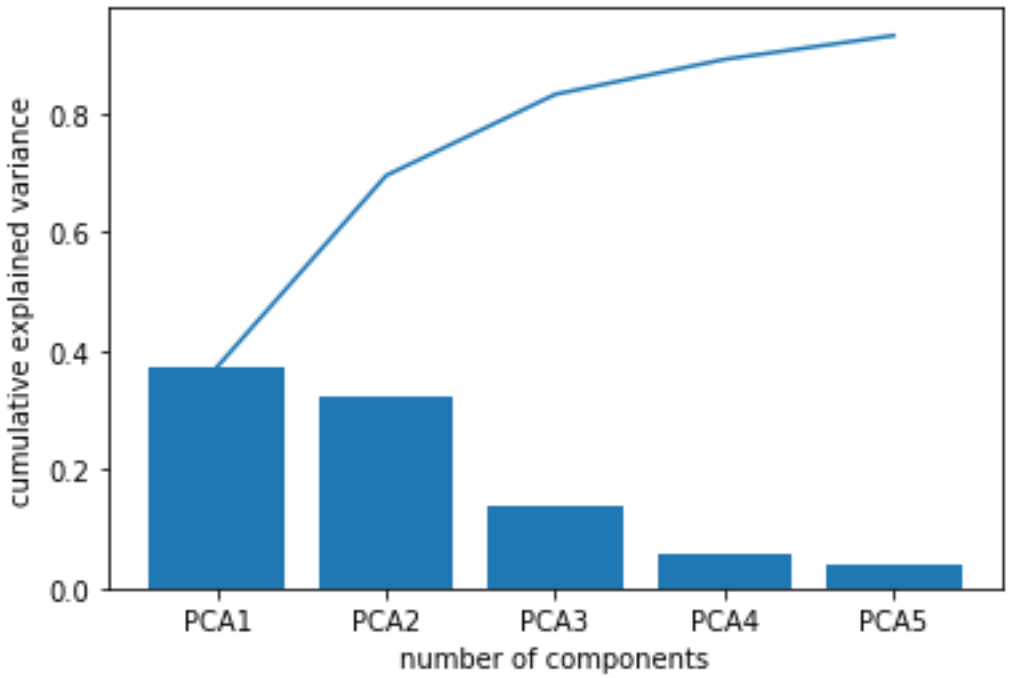
The new feature matrix, PCA is implemented. PCA is used to reduce the dimensionality in order to avoid Curse of Dimensionality. Number of components provided is 5, so PCA returns the top 5 components from the feature matrix. These top 5 features have highest eigen values and the data points are projected towards these PCA components.

New Feature Matrix having top 5 components

	component_1	component_2	component_3	component_4	component_5
0	3.807724	0.648645	-1.345061	-0.470146	0.395251
1	0.897808	1.798504	0.606522	-0.838448	0.105719
2	3.345856	0.192521	-0.148442	-1.471869	0.993765
3	3.567809	-1.484417	-1.294475	-0.326118	0.229870
4	-2.120782	-0.513615	-0.665453	-0.824087	-0.408919
..	...	...	...	...	...
189	-3.795175	1.105453	-0.172119	-0.247805	-0.404298
190	0.589659	-0.322327	-0.385826	-0.175372	-0.212644
191	5.191587	6.069066	-1.370872	1.028110	1.226847
192	0.542320	2.546024	-1.507345	0.005448	0.163644
193	-1.035436	3.533161	-1.270678	1.354428	-0.131812

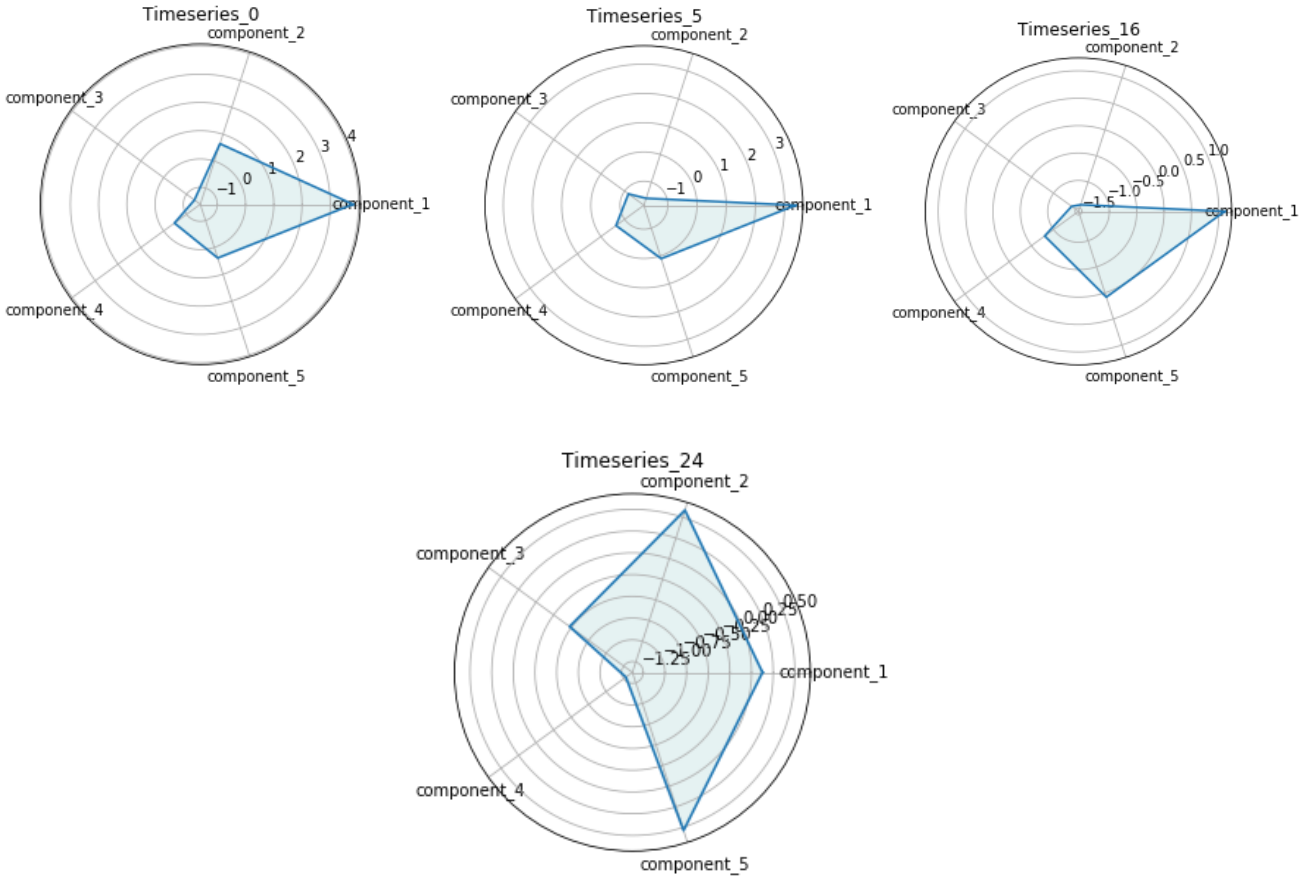
[194 rows x 5 columns]

Scree Plot



Cumulative Explained Variance - [0.3724371 0.32314027 0.13698271 0.05895097 0.040062
66]

Spyder Plot



Analysis of the top 5 features

From the scree plot 93% of the information is captured with 37% of the information from the 1st PCA, 34% from the 2nd PCA.