Name- Suryadeep Chatterjee
CSE-572
Tuesday batch

Git link- https://github.com/surya-de/project_1_data_mining.git

Premise:

I have been given a set of CGM values with glucose readings of 5 different patients. The CGM values are associated with a time series which signifies the interval in which the data has been recorded.

Goal:

The target here is to identify patterns in the glucose level to identify the meal pattern which will help us understand the insulin intake interval of the patient.

Key Findings:

Some key features that can be understood from the data are-

1. In order to find the pattern we need to focus more on the trend of the curves instead of the amplitude. In this case the amplitude is the glucose level.
2. There are some sudden spikes in glucose levels. For the purpose of this project I have assumed that the sudden spikes signify intake of meal.
3. Glucose data has been captured at an interval of 5 mins per day.

Data Cleansing:

The data contains several missing values. Approach followed to handle missing value-

1. **Approach 1-** Removing the null value.
    - **Problem-** With this approach a huge amount of data is lost and might not be a good approach.

2. **Approach 2- Interpolation**
    - **Problem-** This might lead to getting wrong values if the interpolation is not done correctly.

In this project I have followed a hybrid approach.

Steps-

1. Initially I have done an analysis of the data and have realised that the data follows a polynomial pattern.
2. Based on the pattern I have performed a **Spline Interpolation.** The spline interpolation helps us get the missing data.
3. After interpolation the data records with a huge amount of missing values continue to have Null values.
4. I have dropped the Null values from the records.

A. Features used:

1. Polyfit approach
2. Fast Fourier Transform
3. CGM Velocity
4. Welch Method

B. Why did I use the features?

The features which I found helpful in recognizing the pattern are-

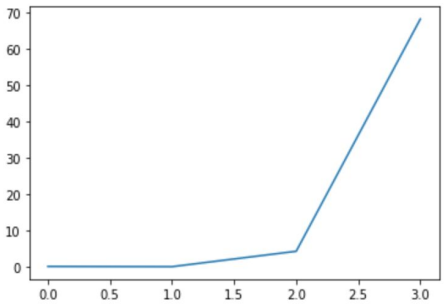
1. Polyfit approach-

Description- Polyfit targets to fit the data points into a polynomial curve that best explains the data points and helps us get the coefficient values of the curve.

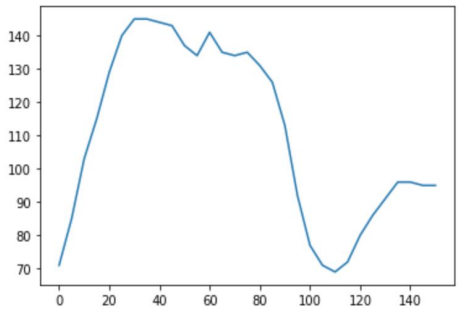
Why did I choose this feature?

1. The coefficient value helps us to understand the trend of the curve or how the value of the cgm varies over a time period.
2. As I mentioned earlier, we are only concerned about the trends in the dataset, polynomial coefficient values come in handy to understand that.
3. For example, the rise or fall of a polynomial curve is mainly dictated by the coefficient values of the polynomial. A negative coefficient value signifies the curve is going to fall for all positive values of the X and Y coordinates.

Diagrams-



i. Degree Vs Coefficient



ii. Interval Vs CGM values

Codes-

```
# Module to perform polynomial fit
# to get the coefficient values.
def perform_polyfit(i):
    colmns = ['coeff_0', 'coeff_1', 'coeff_2', 'coeff_3']
    co_eff = []
    itr = 0
    g_lvl = list(glucose_df.iloc[i])
    interval = [j * 5 for j in range(0, len(glucose_df.iloc[i]))]
    p_fit = list(np.polyfit(interval, g_lvl, 3))
    co_eff.extend(p_fit)
    # Plot chart
    #plt.plot(p_fit)
    #plt.show()
    for cols in colmns:
        glucose_features.at[i, cols] = co_eff[itr]
        itr += 1
```

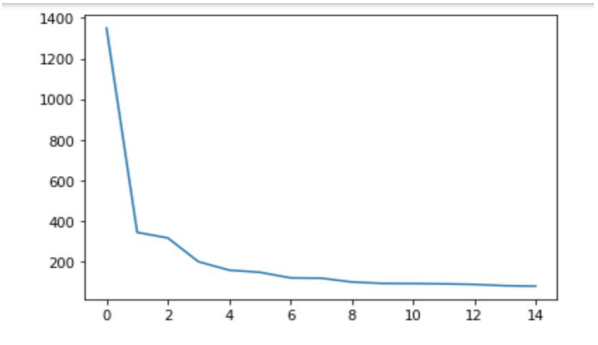
2. Fast Fourier Transform-

Description- FFT helps us reduce the noise while converting the data from time domain to frequency domain.

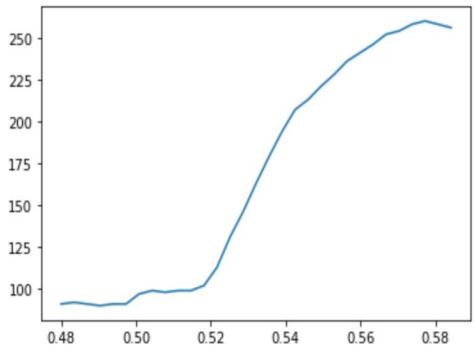
Why did I choose this feature?

1. While evaluating the pattern in data I realised that more than the peaks at certain time we are more interested in the pattern of the peak.
2. Associating the peak values with time may give in wrong results at all the data does not start from the same time. They vary hugely.
3. Hence, the best way is to understand the pattern on the curve's peak.
4. FFT helps us take this leap from time domain to frequency domain.

Diagrams-



i. CGM value on frequency domain



ii. Interval Vs CGM values

The image on the right hand side is the FFT curve of the corresponding cgm data. From this image we can crealy find the trend in the peak of glucose value.

Codes-

```
# Module to perform fft.
def performFft(i):
    itr = 0
    g_lvl = list(glucose_df.iloc[i])
    fft_plot = abs(fftpack.fft(g_lvl))
    fft_vals = sorted(set(fft_plot), reverse = True)
    glucose_features.at[i, 'high_1'] = fft_vals[1]
    glucose_features.at[i, 'high_2'] = fft_vals[2]
    glucose_features.at[i, 'high_3'] = fft_vals[3]
    # Plot chart
    # Uncomment the below lines to
    # plot the curve.
    #print([fft_vals[1], fft_vals[2]])
    #plt.plot(fft_vals[1:])
    #plt.show()
```

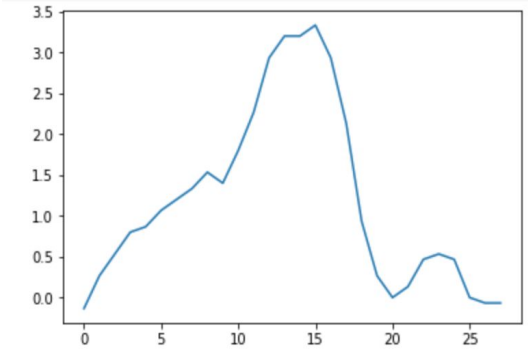
3. CGM Velocity-

Description- The CGM velocity is basically the velocity of the CGM data. This helps us identify the rate at which the cgm value is increasing. I have used a sliding window approach to identify the velocity. For each iteration I can find the rate at which the cgm values are incrementing. An interval with maximum velocity/rate signifies a stiff increase in the CGM value. Hence, I have used the maximum value of velocity as a feature.

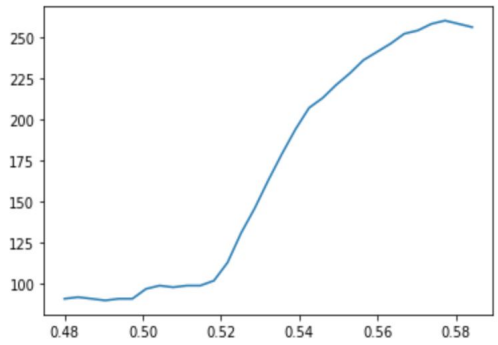
Why did I choose this feature?

1. As mentioned earlier we are more interested in knowing how the cdm values are increasing.
2. To understand that, the cgm velocity helps us understand the rise or fall in the slope of the curve at a given uniform interval.
3. Example- Suppose a cure is increasing 20 units in a 10 min interval and another curve is increasing at 50 units in a 10 min interval. From this we can say that the second curve is encountering a strict rise.
4. Hence, this is really important to understand the kind of rise in the cgm value.

Diagrams-



i. CGM velocity



ii. Interval Vs CGM values

Codes-

```
# Module to perform CGm velocity method.
def cgmVelocity(i):
    window_size = 3
    time_line = 15
    velocity = []
    val_store = glucose_df.iloc[i]
    for j in range(0, len(glucose_df.iloc[i]) - window_size):
        interim = (val_store[j] - val_store[j + window_size]) / time_line
        velocity.append(interim)
    # Find standard deviation of the series.
    s_dev = pd.Series(velocity).std()
    mean_val = pd.Series(velocity).mean()
    median_val = pd.Series(velocity).median()
    glucose_features.at[i, 'cgm_velocity_stdv'] = s_dev
    glucose_features.at[i, 'cgm_velocity_mean'] = mean_val
    glucose_features.at[i, 'cgm_velocity_median'] = median_val
```

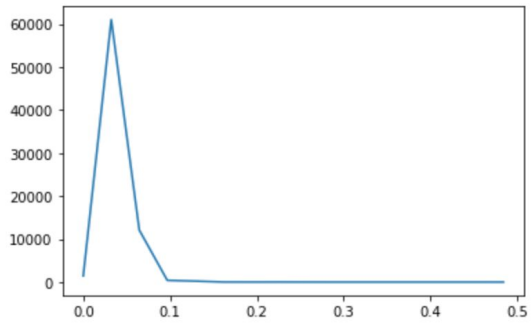
4. Welch Method-

Description- Welch method helps us understand the spectral density in different frequencies.

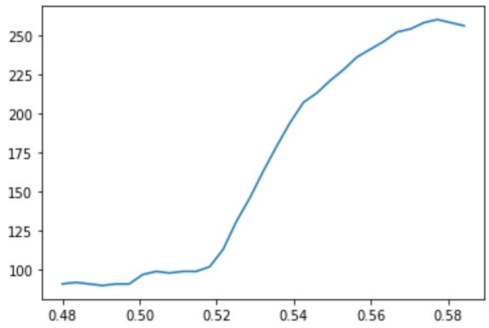
Why did I choose this feature?

1. In this feature we try to find the frequency for overlapping values.
2. This will again help us understand the pattern in a frequency domain.

Diagrams-



i. Welch Value on frequency domain



ii. Interval Vs CGM values

Codes-

```
# Module to perform Welch method.
def performWelch(i):
    store_interim = glucose_df.iloc[i]
    hz, welch_values  = np.array((signal.welch(store_interim)))
    welch_std = pd.Series(welch_values).std()
    welch_mean = pd.Series(welch_values).mean()
    welch_median = pd.Series(welch_values).median()
    glucose_features.at[i, 'max_welch'] = max(welch_values)
    glucose_features.at[i, 'std_welch'] = welch_std
    glucose_features.at[i, 'mean_welch'] = welch_mean
    glucose_features.at[i, 'median_welch'] = welch_median
    #plt.plot(hz, welch_values)
    #plt.show()
```

C. Values of each of the features-

1. **Polyfit-** For polyfit I have considered the coefficient values. The order of the Polynomial expression that gave the best value is 3.

   The coefficient values for one time series-
   [0.00022536230590931776, -0.04774944531356867, 1.0873955563618842, 253.6599965499399]

   How is this value useful?
   From the coefficient values we can clearly understand the rise and fall of a curve which will help us understand the trend.

2. **Fast Fourier Transform-** For fast fourier transform I have considered the top 3 values which are significant in deciding the curve in frequency domain.
   The FFT values for one time series-
   [1349.4563506679858, 344.27408329703223, 344.27408329703223]

How is this value useful?
The FFT values will give us the top 3 peaks in the curve on the frequency domain. The idea is to relate the peaks a hike in glucose level which can be assumed because of the meal intake.

3. **CGM Velocity-**
   In CGM velocity I have used a sliding window technique to understand the rate of attaining peak over the timeline.
   Window size- 3
   Time Interval- 15
   Formula- (cgm_val[t] - cgm_val[t - 15]) / time_interval

   The CGM values for one time series-
   std_dev- 1.12490903675626
   mean- 1.1904761904761902
   median- 0.9

   How is this value useful?
   These values will help us understand the distribution of the data for each time series.
   For each time series to understand the cgm velocity I have used the following -
   a. Standard Deviation
   b. Mean
   c. Median
   These values will help us understand how the velocity value varies.

4. **Welch Method-**
   In the Welch method I have calculated the frequency and peak value to understand the pattern of the curve for each time series in the frequency domain.

   The Welch values for one time series-
   Max_Value- 61021.23211124709
   Std Dev- 15315.242315558187
   Mean- 4701.389744623397
   Median- 7.247982143237667

   How is this value useful?
   For each time series I have considered the max value, standard deviation, mean and median values. With the help of these values I can understand the distribution in the training data set which will help me understand the pattern.

D. Feature Matrix-

For the 4 features I have got total 14 feature attributes for 196 time series data of all the 5 patients.

Steps-

1. I have concatenated all the time series data for all the 5 patients and created a single data frame.
2. Performed data cleansing on the entire data.
3. Added features attributes for the entire data frame.
4. Extracted the feature attributed from the raw data and performed PCA.

Columns and Shape-

```
In [88]: glucose_feature_matrix.columns

Out[88]: Index(['coeff_0', 'coeff_1', 'coeff_2', 'coeff_3', 'high_1', 'high_2',
                'high_3', 'cgm_velocity_stdv', 'cgm_velocity_mean',
                'cgm_velocity_median', 'max_welch', 'std_welch', 'mean_welch',
                'median_welch'],
               dtype='object')

In [89]: glucose_feature_matrix.shape

Out[89]: (196, 14)
```

E. PCA-

1. The feature matrix created in the previous step is used to push it in the PCA component.

2. In PCA we would like to know the top 5 components from the feature matrix which can be used to train the model in the next phase.

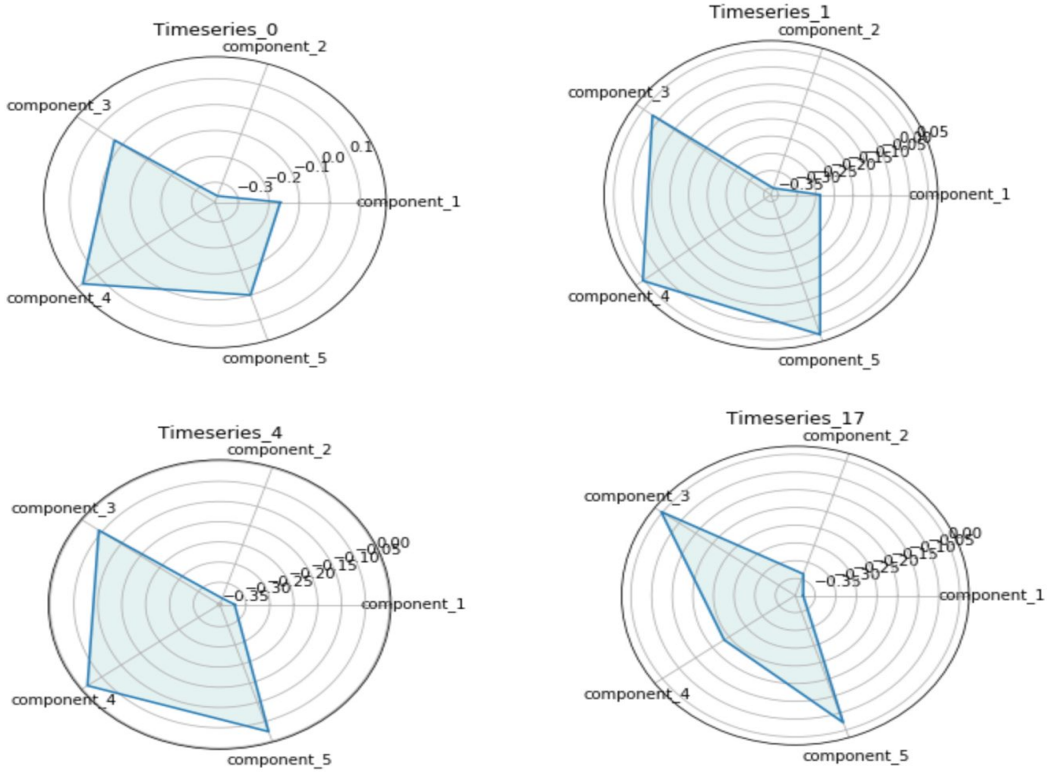
```
pca_components
```

	component_1	component_2	component_3	component_4	component_5
0	-0.162758	-0.351445	0.030568	0.159230	-0.000026
1	-0.238092	-0.349393	0.021559	0.053343	0.055720
2	-0.191260	-0.348941	0.004803	0.093679	0.018351
3	-0.198098	-0.348334	0.011154	0.099342	0.016121
4	-0.323716	-0.338647	-0.042247	-0.013080	-0.024073
5	-0.207715	-0.346617	-0.004482	0.095817	-0.013203
6	-0.261925	-0.342545	-0.023787	0.035441	-0.012859
7	-0.205158	-0.348684	0.025388	0.117498	-0.000721
8	-0.254943	-0.344114	-0.009867	0.068505	-0.011658
9	-0.303677	-0.338429	-0.054479	-0.031252	-0.018270
10	-0.236226	-0.345766	-0.007051	0.061845	-0.003746
11	-0.183003	-0.349635	0.025222	0.134033	-0.010835
12	-0.258589	-0.344043	-0.012209	0.060741	-0.011569
13	-0.255598	-0.342073	-0.042137	0.023151	-0.013225
14	-0.307332	-0.339128	-0.042494	0.018017	-0.034515
15	-0.226316	-0.346984	0.010550	0.098924	-0.004616

This is how the data looks for the top 5 components where each row represents the data for each time series.

Spider Charts-

The components for each time series-



Overall Variance in the Components-

```
['0.922214', '0.073832', '0.002966', '0.000672', '0.000124']
```

F. Explain the features in PCA-

For each component we can know the features that contributed the most in getting that principal component.

Steps-

1. The PCA takes into consideration the values with highest variance.
2. I have extracted the matrix which represents the variances of all the feature attributes for each of the 5 principal components.
3. In the 5 components the columns with maximum values have contributed more in calculating that respective Principal component.
4. **Example-**
   I have 14 feature attributes and 5 Principal components.
   The variance matrix has 5 * 15 values.
   The maximum value for each component has the most relevance in calculating the Principal Components.

Variable Matrix-

```
In [99]: just_comps

Out[99]: array([[0.00486045, 0.27308875, 0.276119  , 0.27639005, 0.27762358,
                 0.27811331, 0.2781836 , 0.27769491, 0.27758938, 0.277884  ,
                 0.27825524, 0.27825519, 0.27825398, 0.27801207],
                [0.98321967, 0.16874864, 0.00736822, 0.06430559, 0.00971224,
                 0.00743919, 0.00718417, 0.00961619, 0.0089998 , 0.0061067 ,
                 0.00667792, 0.00668914, 0.00678833, 0.00812492],
                [0.102051  , 0.42561934, 0.6092424 , 0.41897569, 0.32636006,
                 0.15334981, 0.10003941, 0.29420789, 0.02003648, 0.17721805,
                 0.00825283, 0.00699801, 0.00396785, 0.05641891],
                [0.03158397, 0.06522893, 0.05116125, 0.34401625, 0.13059087,
                 0.13602053, 0.09265821, 0.23131807, 0.7037902 , 0.38021293,
                 0.15442047, 0.15467514, 0.15680867, 0.25276921],
                [0.01580099, 0.13193482, 0.11325193, 0.69661152, 0.26123777,
                 0.09893649, 0.0260219 , 0.24837064, 0.29967304, 0.14686823,
                 0.09872903, 0.09940068, 0.10517464, 0.44920749]])
```

Code-

```
In [121]: def findTopfeatuesinComponents():
              colmns = glucose_feature_matrix.columns
              for i in range(0, 5):
                  component_name = 'PC_'+ str(i + 1)
                  interim = sorted(just_comps[i], reverse = True)
                  fst_max = interim[0]
                  scnd_max = interim[1]
                  print(component_name)
                  print('------------')
                  print('Top 1st Feature', colmns[list(just_comps[i]).index(fst_max)])
                  print('Top 2nd Feature', colmns[list(just_comps[i]).index(scnd_max)])
```

This code will help us know the top features while calculating PCA.

The Top two features for each Principal Component-

```
In [119]: findTopfeatuesinComponents()

PC_1
------------
1st max_welch
2nd std_welch
PC_2
------------
1st coeff_0
2nd coeff_1
PC_3
------------
1st coeff_2
2nd coeff_1
PC_4
------------
1st cgm_velocity_mean
2nd cgm_velocity_median
PC_5
------------
1st coeff_3
2nd median_welch
```