CSE 572 - Data Mining
Course Project Portfolio Report

Kedar Sai Nadh Reddy Kanchi
School of Computing and Augmented Intelligence
Arizona State University, Tempe
kkanchi@asu.edu

Introduction

The objectives of the projects in this course is to extract feature data from a data set from an Artificial Pancreas system using sensor data, synchronize data from two sensors, and compute and report overall statistical measures from data.

For the first project, The project aims to extract six different metrics, each of which will be analyzed for three different time intervals: daytime (6 am to midnight), overnight (midnight to 6 am), and whole day (12 am to 12 am). These metrics will be computed for two cases: manual and auto modes. The six performance metrics to be extracted are the: percentage of time in hyperglycemia (CGM > 180 mg/dL), the percentage of time in hyperglycemia critical (CGM > 250 mg/dL), the percentage of time in range (CGM >= 70 mg/dL and CGM <= 180 mg/dL), the percentage of time in range secondary (CGM >= 70 mg/dL and CGM <= 150 mg/dL), the percentage of time in hypoglycemia level 1 (CGM < 70 mg/dL), and the percentage of time in hypoglycemia level 2 (CGM < 54 mg/dL). Each of the six performance metrics will be computed for two different cases: Case A, which is the manual mode, and Case B, which is the auto mode. The extracted metrics will then be reported as the mean value of each metric over all days.

For the second project, the data set provided for this project contains meal and no meal data of subjects 1 and 2, along with the ground truth labels of a meal and no meal for the same subjects. Meal data will be extracted from the InsulinData.csv file, and no meal data will comprise 2 hrs of raw data that does not have meal intake. Using Python, we are required to extract features from the training data set and ensure that the features are discriminatory. This project aims to develop a machine learning model using k-fold cross-validation to recognize whether the sample in the training data set represents a person who has eaten (Meal) or not eaten (No Meal).

For the third and final project, this project aims to apply the cluster validation technique to data extracted from a provided data set. The provided training data-set will extract features from meal data and cluster the meals based on the number of carbohydrates in each meal. The features extracted in Project 2 will be used to cluster the meal data into n clusters for clustering. The main objectives of this project are to develop code that performs clustering, test and analyze the results of the clustering code, and assess the accuracy of the clustering using the Sum of Squared Error and supervised cluster validity metrics.

Solution

For all three projects, I have developed the code logic for this project using the directions mentioned in the Analysis Procedure section of the project description and the guidelines explained in the attached videos.

So, to start, for every project before diving into the crux part of the code, I import the necessary Python libraries, loaded the provided CGM Sensor Data & the Insulin Data CSV files using the 'use cols' attribute within the 'read_csv' function to read only the required columns for these three projects. The last common thing we perform for each project on both CSV files is creating a new column called the Date Time Stamp column. This column is created by combining each CSV file's date and time columns.

Solution Implemented for Project 1

Now the next important step is to clean the CGM Sensor Data CSV file, as the project description PDF file mentions, to tackle the missing data problem by following the standard methods like deletion of the entire day of the data or interpolation. Since the CGM Sensor Data CSV file is enormous, I decided that the interpolation method would not be a good way as it might dilute the quality of the results. So, I went ahead with the deletion method. So, for this, I first took the first step to extract all the dates that contain missing values into a list. Now, since we need to delete all the rows whose date is present in the extracted list, it is tough to loop over the data frame as a whole. So, I have used the concept of indexing to implement this logic. So, I set the index on the date column for the CGM Sensor Data and then dropped all the rows whose index, i.e., date, is present in the list of dates that needs to be removed. Since a deletion process is executed on the data frame, I have reset the index to the initial state. With this, the clean-up process for the CGM Sensor Data is completed.

So, with the CGM Sensor Data and the Insulin Pump Data loaded into the Jupyter environment as data frames that have been cleaned and processed to match the project requirements, I now move on to find when the auto mode starts. We need to find the first instance of the AUTO MODE ACTIVE PLGM OFF in the Alarm column in the Insulin Pump Data data frame. It is specified in the project description file that the insulin data is in reverse order of time. This means that the first row is the end of the data collection, whereas the last row is the beginning of the data collection. The data starts

with manual mode. Manual mode continues until you get an "AUTO MODE ACTIVE PLGM OFF" message in column "Q" of the InsulinData.csv. From then on-wards, Auto mode starts. You may get multiple "AUTO MODE ACTIVE PLGM OFF" in column "Q," but only use the earliest one to determine when you switch to auto mode. So, as per this condition to get the first occurrence of the "AUTO MODE ACTIVE PLGM OFF," I have sorted the insulin data by the newly created date time stamp column in ascending order. After sorting the insulin data in ascending order, I have extracted all the rows where the column "Alarm" has the value "AUTO MODE ACTIVE PLGM OFF." And then, from those extracted rows, I have further extracted the 1st row, which gives the first occurrence of the value "AUTO MODE ACTIVE PLGM OFF."

Again, following the directions specified in the project, once you determine the start of Auto Mode from InsulinData.csv, you must figure out the timestamp in CGMData.csv where Auto Mode starts. This can be done simply by searching for the time stamp nearest to (and later than) the Auto mode starts the time stamp obtained from InsulinData.csv. So, following these suggestions, I created two new data frames where one would contain the data which was collected in auto mode. The other would contain the data collected in manual mode using the condition where the value of the date-time stamp in the cloned & sorted CGM Sensor Data frame is greater than or equal to the extracted date time stamp of when the data collection mode changed from manual to auto.

And then, I created an index on both the newly created data frames on the date time stamp columns. Then as per the instructions in the project description file, I figured out all the dates based on the lambda calculation. And then, for each needed percentage, I just used the between the command to input various time zones as specified in the description file for daytime, overnight, and whole day. Once all the required percentages, both auto, and manual, have been calculated, I transformed all those values into a data frame which was then pushed to be saved into the results.csv file without any headers as specified.

Solution Implemented for Project 2

For this project, I had to implement createProcessedMealDataFrame and createnomealdata functions to extract the meal and no-meal data from the CSV files.

Inside both the functions, a copy of the patient insulin data frame is created, and the index is set on the date time stamp column. The created index then sorts the data, and any rows containing a value of 0.0 for the 'BWZ Carb Input (grams)' column are replaced with NaN. The data frame is then filtered only to include rows with non-NaN values. This is because, as per the project requirements, we need to search for column Y in the patient insulin data frame, which is the 'BWZ Carb Input (grams)' column for a non-NAN non-zero value, as this time indicates the start of meal consumption time (tm).

The project requirements mention that the meal data comprises a 2hr 30 min stretch of CGM data that starts from tm-30min and extends to tm+2hrs. However, no meal starts at time tm+2hrs where tm is the start of some meal. Therefore, the next part of the function creates a list of valid timestamps for the meal data by checking the difference between adjacent timestamps and only including those with a difference greater than or equal to 120 minutes because we do not want to have more than two meals within two hours and that is why we had to eliminate such scenarios or in this case such timestamps. Similarly, when we are trying to extract the no meal data the time difference should exceed 4 hours add only then timestamp is added to the list of valid time stamps within the createnomealdata function.

As mentioned above, with the correct meal start time available, we need to extract the meal data comprising a 2hr 30 min stretch of CGM data that starts from tm-30min and extends to tm+2hrs. The function then loops through the list of valid timestamps and creates a time range of 30 minutes before the timestamp and 120 minutes after the timestamp. The function then locates the corresponding date for the timestamp and extracts the CGM data for the time range. This process is repeated for all valid timestamps, and the extracted CGM data is stored in a DataFrame.

Two new lists are created, meal_data and meal_data1, using the createProcessedMealDataFrame function on the two patient data sets, respectively. The iloc method is then used to extract only the first 30 columns of data from each of the meal_data and meal_data1 data frames. This is because we are extracting the meal data for a stretch of 2hrs 30min, and since the CGM data samples are collected every 5 min, the 2hrs 30min stretch of data contains 30 samples.

Similarly, for the extraction of the no meal data, a dataset is created by iterating through the list of valid timestamps in the list created above, selecting CGM measurements for each timestamp, and adding them to the dataset. If there are more than 24 CGM measurements for a given timestamp, they are split into 24-hour segments. This is because we are extracting the no meal data for a stretch of 2hrs, and since the CGM data samples are collected every 5 min, the 2hrs stretch of data contains 24 samples. The resulting dataset is returned as a pandas data frame. Finally, Two new lists are created, no_meal_data and no_meal_data1, using the createnomealdata function on the two patient data sets, respectively.

Using these created 4 data frames, two each for the meal and no-meal data; I moved on to the next phase, feature extraction. Moving in this direction, we will create createmealfeaturematrix and createnomealfeaturematrix functions. Both functions start by dropping any rows containing more than the specified threshold of NaN values from their respective data frames. Then, both function interpolates any remaining NaN values using[1 & 2]. Afterward, any rows containing NaN values are dropped again. The following paragraph mentions the features selected for this project which have been calculated in both functions.

The tau_time and difference_in_glucose_normalized features are then calculated for each row. tau_time is the time difference between the minimum and maximum glucose levels during a meal. The normalized difference is between the maximum glucose level during a meal and the minimum glucose level during the 5 hours before the meal, normalized by the maximum glucose level during the meal. Next, the

frequency-domain features are calculated: power_first_max, power_second_max, index_first_max, and index_second_max. Since we have established that the CGM data is a sinusoidal wave, the sinusoidal frequency response is critical. These features are calculated by taking the discrete Fourier transform (DFT) of the CGM measurements for each row, then finding the second-largest and third-largest values in the resulting DFT array. Furthermore, three more features are calculated for each row: 1stDifferential, 2ndDifferential, and standard_deviation. 1stDifferential is the maximum difference between two consecutive glucose level measurements during a meal. 2ndDifferential is the maximum difference between any two consecutive 1stDifferential values for a given meal. standard_deviation is the standard deviation of all glucose level measurements during a meal.

Moving into the last segment of the code, the feature matrices for meal and non-meal data are concatenated. The 'label' column is added to each feature matrix to indicate whether it is a meal or non-meal data. The resulting feature matrices are shuffled using the shuffle() function and split into ten folds using the KFold() function. The DecisionTreeClassifier() model trains the model on the principal data, the feature matrix, without the 'label' column. The model's accuracy is evaluated using the score() function, and the scores are appended to the scores_rf list. The mean score is then printed to evaluate the performance of the model. The precision, recall, accuracy, and F1 score are calculated using the precision_score(), recall_score(), accuracy_score(), and f1_score() functions, respectively. Ultimately, the decision tree classifier is trained using the whole dataset, and the trained classifier is saved using the dump method from the joblib module.

Solution Implemented for Project 3

Next, we implement the function to extract the meal data information. So, we reuse the same createProcessedMealDataFrame function, which was implemented as Part of the Project 2 solution but with a few minor additions. Instead of returning one list, we now return two lists from this function. This is because we would like to maintain one list to contain the meal data information, but the new list will contain the number of carbohydrates in each meal.

Then I implemented a function called "function_to_clean_data_and_sample_data," which cleans and samples the meal data. This function truncates the meal data to 30 columns, reverses the order of the rows, and removes rows that do not have 30 columns or have a "nan" value. The function returns the cleaned and sampled meal data and the corresponding amount of meal data.

Next, the team wrote a function called "calculate_velocity" that calculates the velocity of the meal data. This function inputs the cleaned and sampled meal data and calculates the difference between each adjacent data point. The output is a list of velocities. The team also wrote functions to calculate the meal data's maximum velocity, minimum velocity, and average velocity. These functions take the list of velocities as input and return the corresponding value. Similarly, maximum acceleration, minimum acceleration, and average acceleration were calculated using the acceleration of meal consumption. The acceleration was calculated by taking the difference between the consecutive velocities and dividing it by the time difference between them.

We now move on to extracting the remaining features rom the meal data to train the machine-learning model. These features include entropy, interquartile range (IQR), and the top six maximum values resulting from fast Fourier transform (FFT). Entropy is a measure of the randomness of the data. It was calculated using the entropy function from the Scipy library. IQR is a measure of the spread of the data. It was calculated using the iqr function from the Scipy library. For fft, we apply fft and consider the top 6 max values resulting in 6 columns.

In order to apply clustering techniques to the dataset, we must first obtain the ground truth values for all of the data points. The ground truth values are obtained by dividing the meal amount into bins of size 20, grouping the rows in the meal data according to their meal amount labels, and assigning them to the corresponding bins. By dividing the difference between the maximum and minimum values by 20, one can calculate the number of bins, or n. The meal data can be grouped and categorised based on the quantity of carbohydrates in each serving after the ground truth values have been obtained.

To calculate SSE, we define the function which takes bin_value_and_bins_weighted_value as input. This function calculates the SSE value for a given bin. We then use this function to calculate the total SSE value for all bins. To calculate entropy and purity, we define the function get_BinsMG, which takes result_labels, true_label, and clusterNum as inputs.

Now coming to the clustering analysis on meal data, The code is divided into using two different clustering algorithms, namely K-means and DBSCAN, and calculates their SSE (Sum of Squared Errors) and purity values. The optimal number of clusters for a dataset is determined using an elbow curve in the KMeans algorithm. In this graph, the SSE (sum of squared errors) is plotted against the number of clusters. The ideal K value for the dataset can be found at the elbow point, where SSE starts to level off. This method presents a quantitative way to figure out how many clusters are necessary for a particular dataset. The code initializes the number of clusters to that ideal optimal K-value and fits the K-means model on the final feature matrix for meal data. It generates bins of size 20 for meal amounts and makes a copy of it. It generates bins using the K-means model labels. It calculates the SSE value for each generated bin and sums them up, weighted by the number of elements in each bin. This is the KMeansSSE value. It generates a contingency matrix using the actual meal amount bins and K-means model labels. It calculates each cluster's entropy and purity values using the contingency matrix and stores them in two lists. Whereas for the DBSCAN clustering[5 & 6], the code initializes the DBSCAN model with an epsilon value of 0.5 and a minimum number of samples of 2 and fits it on the final feature matrix for meal data. It generates bins using the DBSCAN model labels. It calculates the SSE value for each generated bin and sums them up, weighted by the number of elements in each bin. It generates a contingency matrix using the meal amount bins and DBSCAN model labels. It calculates each cluster's

entropy and purity values using the contingency matrix and stores them in two lists.

Furthermore, finally, coming to the last round of calculations, the code calculates the total number of elements in each cluster and stores it in the count's array. It normalizes the count's array to obtain the proportion of elements in each cluster and stores it in the coeffs array. It calculates the final entropy and purity values by multiplying each cluster's entropy/purity values with their corresponding proportion of elements and then summing them up.

Results

	CGM > 180 mg/dL	CGM > 250 mg/dL	CGM >= 70 mg/dL and CGM <= 180 mg/dL	CGM >= 70 mg/dL and CGM <= 150 mg/dL	CGM < 70 mg/dL	CGM < 54 mg/dL	CGM > 180 mg/dL	CGM > 250 mg/dL	CGM >= 70 mg/dL and CGM <= 180 mg/dL	CGM >= 70 mg/dL and CGM <= 150 mg/dL	CGM < 70 mg/dL	CGM < 54 mg/dL	CGM > 180 mg/dL	CGM > 250 mg/dL	CGM >= 70 mg/dL and CGM <= 180 mg/dL	CGM >= 70 mg/dL and CGM <= 150 mg/dL	CGM < 70 mg/dL	CGM < 54 mg/dL
Case A: Manual mode	11.63194	10.06944	18.83681	17.01389	1.388889	0	25.34722	17.1875	44.87847	34.80903	6.365741	4.340278	31.16319	22.22222	63.71528	47.56944	6.828704	4.340278
Case B: Auto mode	6.543803	4.74537	21.8078	19.02778	2.314815		19.80287	7.326389	51.6353	38.32885	5.257937	1.629274	22.54704	8.038194	73.4431	56.74283	5.404589	1.629274

Figure 1: Results for Project 1

```
Prediction score is 48.64035087719298
Precision: 0.484
Recall: 0.805
Accuracy: 0.478
F1 Score: 0.605
[1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0
 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1
 1 1 1 1 1 1 1 1 0 0 0 0 0 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 0
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 1 1 1 0 0 0 0 0]
```

Figure 2: Results for Project 2

SSE for Kmeans	SSE for DBSCAN	Entropy for KMeans	Entropy for DBSCAN	Purity for KMeans	Purity for DBSCAN
35349	117592	4.632794203	2.577829046	1.392976589	0.919732441
K-Means (K = 6)		DBSCAN epislon value = 0.5		DBSCAN minimum value = 2	

Figure 3: Results for Project 3

The Figure-3 displays all the values that were included in the results.csv file. In theory, the SSE score for DBSCAN should be lower than that of KMeans clustering. This is because DBSCAN is better equipped to handle outlier and noise values in data. As previously stated, DBSCAN can detect clusters of any shape[5]. The table in Figure-3 shows that the KMeans algorithm produces the best results when K is set to 6. In DBSCAN, the best values for epsilon and minimum samples are 0.5 and 2, respectively. Furthermore, the table shows significant differences between the two clustering algorithms.

K-means and DBSCAN clustering algorithms differ in several key ways. K-means generates spherical clusters, while DBSCAN generates clusters of any shape. DBSCAN is more efficient in handling noisy and outlier datasets, while K-means is inefficient in such cases. DBSCAN requires two parameters, radius and minimum points, while K-means only needs the number of clusters (K). K-means performance is unaffected by data density, while DBSCAN does not perform well on sparse data. It is important to consider these differences when choosing between the two algorithms for a specific clustering task[5 & 6].

Lessons Learned

Throughout this project, several valuable lessons were learned. Firstly, to start any project, it is crucial to have a thorough understanding of the problem at hand. Then, it is crucial to understand that data cleaning and preprocessing are essential steps in any data analysis project. In this project, the data cleaning process involved replacing and dropping NaN values, as well as using linear interpolation to fill in any remaining NaN values. Linear interpolation helped create a more complete dataset for further analysis. Additionally, Lambda functions were utilized during the data preprocessing process for calculating specific dates and other calculations within the data frame.Secondly, working with large datasets requires careful consideration and planning. Indexing is a powerful tool for manipulating data frames, especially when dealing with large data sets. While working with time-series data, it is important to correctly index the data using the date-time column. Sorting the index and filtering the rows with non-NaN values help in getting the correct data. Thirdly, machine learning models require carefully selected features to achieve good performance, and feature engineering is an iterative process that requires experimentation and testing. It was important to understand the domain knowledge related to the CGM data and select appropriate features that capture the relevant information. It was also important to understand the mathematical concepts underlying the data analysis is important. The Fourier transform was used to calculate the frequency-domain and time-domain features required for this project. This tool proved to be very powerful in analyzing the CGM data and extracting useful features. Therefore, the right balance between domain knowledge and statistical analysis should be considered while selecting the features. Fourthly, clustering algorithms are useful for grouping data points based on similarities. However, selecting the appropriate clustering algorithm and hyperparameters can be challenging. Lastly, while working on the project, I encountered several errors and challenges, but this provided me with an opportunity to improve my debugging skills and find effective solutions to resolve the issues. Overall, this project emphasized the importance of careful data preparation, feature extraction, and analysis in machine learning projects.

References

- Acuna Mayo M, Chepulis L, Paul RG (2019) Glycemic-aware metrics and oversampling techniques for predicting blood glucose levels using machine learning. PLoS ONE 14(12): e0225613.
- Acuna E, Aparicio R, Palomino V. Analyzing the Performance of Transformers for the Prediction of the Blood Glucose Level Considering Imputation and Smoothing. Big Data and Cognitive Computing. 2023; 7(1):41.
- K-means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks
- K-Means Clustering Explained
- DBSCAN Python : The Optimal Value For Epsilon (EPS)
- K-means, DBSCAN, GMM, Agglomerative clustering — Mastering the popular models in a segmentation problem