Data Mining of Blood Glucose Monitor Output

Jeffrey Pan
1223505816
Arizona State University
Tempe, Arizona
jpan59@my.asu.edu

Blood glucose data from a glucose monitor was used to determine whether the patient had just eaten a meal or not. Eight features were extracted from the time-series data and put through a multi-layer perceptron classifier. In addition, given that a set of data was taken immediately after eating a meal, a model was also created to determine the size of that meal (in carbs). This model was created using k-means to place the data into specific bins (ranges of carb values). Despite moderate error measurements (using entropy and purity), this paper provides useful insight into the procedures to create a base that can be used to ultimately create a successful classifier.

Keywords—multi-layer perceptron, classifier, feature extraction, machine learning, k-means clustering

I. Introduction

Given a series of sensor data from an insulin pump, what kind of patterns can we extrapolate? One possibility is to use glucose data read in to determine whether a meal has been eaten or not. If done properly, this can help alleviate the amount of manual input the user has to give (or perhaps even catch meals that the user forgot to input). Another possibility is to determine how often the patient has critical blood glucose levels and whether it occurs during the day or at night. Extracting useful information This paper will describe the steps taken to analyze the raw data and the accuracy of such analysis.

II. Background

The Medtronic 670G system, an Artificial Pancreas medical control system, uses a continuous glucose monitor (CGM) to collect blood glucose measurements every five minutes. The user manually inputs carbohydrate intake during meal into a Bolus Wizard, which then calculates the bolus insulin to be delivered. The user then programs an insulin pump to deliver that amount.

A. Data

Raw data is given from two sources: the CGM and the insulin pump. In this paper, I will be utilizing two main columns of data from the CGM: the date time and the CGM reading in mg/dL. From the insulin pump, the date time and "BWZ Carb Input" are used to determine what time the user ate food and how much food was eaten.

B. Goals

The project was split into three main goals. The first is to extract metrics that show the percentage time in certain blood glucose level ranges (like hyperglycemia, CGM > 180 mg/dL, or hypoglycemia, CGM < 70 mg/dL). The second is to find a way to classify a 2-hour series of glucose data into mealtime or non-mealtime. The third is, given that a 2-hour series of data was mealtime, what was the carb intake of that meal? These two tasks require different approaches and algorithms, which are described in section III below.

III. Solution

A. Reading in Data

To work with the raw data that was given as a csv file, the specific columns that I wanted to work with needed to be read in and placed into local memory. The data also needed to be separated into two separate arrays, one for mealtime and one for non-mealtime data. The mealtimes given in the insulin data file were first read in. Then, blood glucose data from the CGM was read in. The two hours following and the 30 minutes prior to a meal became a set of data for the mealtime array, and any period of two hours after mealtime arrays became a set of data for the non-mealtime array.

B. Percentage of Time Metrics

The time-series data were read in, and various metrics were calculated:

1. Percentage of time in hyperglycemia (CGM > 180 mg/dL)
2. Percentage of time in hyperglycemia critical (CGM > 250 mg/dL)
3. Percentage time in range (CGM >= 70 mg/dL and CGM <= 180 mg/dL),
4. Percentage time in range secondary (CGM >= 70 mg/dL and CGM <= 150 mg/dL),
5. Percentage time in hypoglycemia level 1 (CGM < 70 mg/dL), and
6. Percentage time in hypoglycemia level 2 (CGM < 54 mg/dL).

These metrics were extracted in three different time intervals: daytime (6 am to midnight), overnight (midnight to 6 am), and whole day (daytime and overnight combined).

C. Feature Extraction

To compare these separate scenarios, features must be extracted from the CGM readings that can help discriminate between mealtimes and non-mealtimes. I have selected eight (8) different features. The first four are:

1. Tau – The time at which the max glucose measurement in that 2-hour period occurs minus the time of the meal or beginning of 2-hour non-mealtime
2. dGN – The delta between the max glucose measurement and the glucose measurement at mealtime (or beginning of the 2-hour non-mealtime segment)
3. dCGM – Derivative of CGM glucose data
4. dCGM2 – Second derivative of CGM glucose data

The shape of the glucose values after mealtime typically follows a sinusoidal graph. It increases shortly after the meal,

then decreases. To capture the shape of this curve and convert time-based patterns into sinusoidal cycles, a Fourier transform can be used[1]. The next four features are extracted by taking a Fourier transform on the data and finding the location of and value of the 1st and 2nd highest peaks:

5. pf1 – Value of the first peak
6. f1 – Location of the first peak
7. pf2 – Value of the second peak
8. f2 – Location of the second peak

D. Training Models to Classify

After extracting features, a model was created that, given a 2-hour series of glucose data, can differentiate between mealtime or non-mealtime. Because I am classifying between only two possible options, this is considered a binary classification. I chose to use a multi-layer perceptron (MLP) classifier (MLPClassifier).

Perceptrons, modeled after the human neuron, consist of a series of weights applied to input features and an activation function. Perceptrons are widely used in classification methods. Adding extra layers causes the model to learn deeper and more complex interactions between the various features, but too many layers could result in overfitting.[2] The data was split into training and testing sets (80/20 split) to train the model. The MLPClassifier model in sklearn can be easily saved as a pickle file and transferred to be used on another input document.

E. Using Clustering to Determing Meal Size

More than just classifying between mealtime and non-mealtime, analysis can be done to determine the carb input of meals. I used two methods of analysis:

1. K-Means – K-means clustering groups items into k number of bins, with items that have similar features grouped into the same bin[3]. K-means doesn't work well with data of large dimensions (distances become too large), so principal component analysis can be applied to reduce the number of dimensions[4]. Bins are separated into sizes of 20 mg/dL. For this set of data, there were seven bins.
2. Bisecting k-means – An extension of the standard k-means algorithm, bisecting k-means starts with all data in a single bin, and then separating the points into two bins that minimum the sum of squared error (SSE) within each bin. Bisecting k-means, while similar, has less trouble with initialization compared to k-means. Since the dataset was placed into seven bins, the data was bisected six times (to create seven clusters).

One common downside of k-means or bisecting k-means is that it can't handle non-globular clusters or clusters of different sizes and densities. The meal sizes fell into a normal distribution, which meant that there were many data points that fit into bins 3, 4, and 5, and less so for the others. K-means would attempt to put them into bins of equal sizes. A possible alternative is using DBSCAN, a density-based clustering algorithm. However, in practice, the data has so many outliers that is very difficult to have DBSCAN create proper bins without manually combing through the results. In addition, since the classification bins are closely related to each other, it will be hard to distinguish between where one bin ends and another starts.

F. Analyzing Results and Honing Algorithms

Various measurements were used to determine the effectiveness of and to help hone the algorithms.

1. MLPClassifier – A goal was to keep the MLP algorithm from being too complex while performing well. After the MLPClassifier was trained and used to predict the test cases, the accuracy_score() function was used to determine the effectiveness. In addition, test cases given by the professor of this class also helped to ensure the network did not overfit. The algorithm could be tweaked by changing the way the train and test sets were split (instead of 80/20), changing the learning rate of the MLPClassifier, or changing the size of the hidden layers.
2. Sum of Squared Error (SSE) – Squared error is calculated by finding the Euclidian distance between each data point and the center of their cluster. The sum of the squared error of each point in a cluster is taken to result in the SSE value for that cluster. A cluster that is spread apart will have a large SSE. In bisecting k-means, the cluster with the largest SSE is split into two clusters. Since the SSE is reliant on the scale of the input variables (different features will result in wildly different SSE measures), is it used as a relative measure and not an absolute measure to determine how good a cluster is.

$$SSE = \sum_{i=1}^{k} \sum_{x \in C_1} dist^2(m_i, x)$$

3. Entropy/Purity – Entropy and purity are measurements of how well separated the clusters are. If a cluster has an even mix of elements from each bin, then the clustering algorithm has not helped categorize the data points, the purity is low, and the entropy is high. If each cluster has only elements from a single cluster, then the purity will be 1, and the entropy will be 0.

   In the entropy equation below, j represents the number of a specific type of element (those that belong in the same bin) in a cluster and t is the total number of elements in that cluster. This is calculated for each different class in a cluster and the negative sum gives you the entropy of that cluster. Total entropy is weighted average among all clusters.

$$Entropy(t) = -\sum p(j|t) \log p(j|t)$$

   Purity for a cluster is calculated by counting the maximum number of elements that a single class has in that cluster and dividing it by the total number of elements in that cluster. The total purity for the clustering algorithm is the weighted average of the purity of each cluster.

$$purity(c_i) = \frac{1}{n_i} \max\left(n_i^j\right)$$

IV. Result

A. Percentage of Time Metrics

There were 18 different metrics (different blood glucose ranges over different periods of time), but notably, the patient spends approximately 30% of each full day in hyperglycemia and 6% of each day in hypoglycemia. These values are much higher during the day than at night, reaching 35% of the time in hyperglycemia and 8% of the time in hypoglycemia. These values drastically decrease at night. A full table of this data is shown below:

Metric	1	2	3	4	5	6
Night	19%	4%	72%	53%	3%	0%
Day	35%	13%	57%	44%	8%	3%
Full	32%	11%	61%	46%	7%	2%

B. MLP Classifier

The first part of this project was to train a binary classifier to determine whether a series of data followed or meal or not. Although training accuracy was high (> 85%), running it through the tests that were given to the class resulted in barely above 50%, which is similar to random guessing.

Looking forward, better accuracy could be achieved by altering a few things. First is to have better feature selection. One major difference between mealtime and non-mealtime is the shape of the glucose curve in the ensuing couple hours. Measurements in addition to Fourier transforms could help differentiate the two classes. In addition, the values of a single test subject may vary drastically compared to another subject. Having a wider set of training data could help classify other sets of data. Lastly, tweaking the shape of the hidden layers did not result in a better accuracy, but perhaps further testing could be done.

Perhaps a more efficient way to approach this classification was instead to use a time series classification algorithm. Instead of trying to extract features that represent the shape of the curve, instead use a shapelet-based classifier that finds a pattern in a series[5].

C. Clustering

The K-means and bisecting K-means algorithms resulted in entropy values of approximately 1.5 and purity values of 0.3. While not terrible, this certainly can be improved drastically.

A common theme between the MLP classifier and clustering is the use of the extracted features. Improving these features will likely result in the biggest improvement of entropy and purity. In addition, DBSCAN is still a useful tool, as long as outliers are classified properly. One method is to have outliers put in a throwaway bin while the rest of the points are passed through DBSCAN normally. Another method is to reduce dimensionality and distance to reduce the existence of far outliers. Lastly, the DBSCAN algorithm itself can be tweaked to include the previous outliers into existing bins.

V. Lesson Learned

This project is a great example of how to take a series of raw data and apply classification algorithms on it. One of the most difficult components of this project is feature extractions. Picking proper features is harder than I previously thought. Not only do the features have to be varied enough to differentiate from one another, they also must be representative of the data at hand. Otherwise, a feature that looks the same whether it's mealtime or non-mealtime simply adds extra complexity to the classifier algorithm.

Another lesson was altering the parameters of the classifiers to optimize the model. Especially in a complex model like a multi-layer perceptron, it is difficult to visualize how the different input features lead to a classification. One of the possibly frustrating, yet intriguing, parts about machine learning is this black box. How should the training set be split up into training/test sets? How many layers should do the classifier have? What is a proper learning rate that maximizes efficiency and provides good results? Working with these on a practical level instead of purely theoretical is invaluable experience.

Although classifiers are very complex, there are a lot of existing tools that make classification simpler. For example, implementing a multi-layer perceptron is as simple as running the MLPClassifier function, and splitting a set of data into randomized training and test sets can be done with the train_test_split function. Perhaps most interesting was the ability to save a classifier to be used in a separate program. Instead of having to comb through the data and train a model each time you wanted to use the classifier, the classifier could be saved as a pickle file and then simply loaded into a separate program.

Information gathered from this project could be applied to real-life scenarios. For example, using the information about percentage of time spent in hyper- or hypo-glycemia could help the patient alter their diet or medication. Spending 30% of each day in hyperglycemia is not healthy and steps can be taken to reduce it. Machine learning can come into play by associating a carb intake size with a specific rise in blood sugar, thus determining what is a safe level of carbs to eat. Instead of reacting to a spike in blood sugar, a model that can anticipate the rise or fall of blood sugar can help direct the correct flow of insulin to best help the patient.

VI. Conclusion

In this study, raw data from a blood glucose reader was given for a patient. Simple information was extracted from this time-series data, such as time spent in hyper- or hypo-glycemia. Another goal was to take 2-hour time periods of the blood glucose levels and determine whether the patient had just eaten a meal or had not had a meal for at least 2 hours before. To do so, eight features were extracted (Tau, dGN, dCGM, dCGM2, pf1, f1, pf2, f2) and put through a multi-layer perceptron classifier to classify them as either mealtime or non-mealtime. In addition, further classification was performed on the mealtime data to determine how large the meal was (in carbs). This was accomplished by using k-means clustering and classifying the data as specific "bins" that represented different meal sizes.

While the specific features and parameters used in this paper did not result in the best accuracy measurements, this paper shows the basis for such analysis. With a bit of tweaking to the algorithm and input features, much more accurate results can be obtained.

Ultimately, this project serves as an example of how to extract useful information from a stream of data. The information can be taken directly, as a percentage of time in a specific state, or it can be passed through classifiers. Information can then be used to take actionable measures, such as diet changes from the patient, or it can be analyzed by a doctor to change medication.

REFERENCES

[1] "An interactive guide to the Fourier transform," *BetterExplained*. [Online]. Available: https://betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform/. [Accessed: 20-May-2022].

[2] C. Bento, "Multilayer Perceptron explained with a real-life example and python code: Sentiment Analysis," *Medium*, 30-Sep-2021. [Online]. Available: https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141. [Accessed: 21-May-2022].

[3] D. M. J. Garbade, "Understanding K-means clustering in machine learning," *Medium*, 12-Sep-2018. [Online]. Available: https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1. [Accessed: 21-May-2022].

[4] M. Brems, "A one-stop shop for principal component analysis," *Medium*, 26-Jan-2022. [Online]. Available: https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c. [Accessed: 21-May-2022].

[5] A. Amidon, "A brief survey of time series classification algorithms," *Medium*, 27-Aug-2021. [Online]. Available: https://towardsdatascience.com/a-brief-introduction-to-time-series-classification-algorithms-7b4284d31b97. [Accessed: 21-May-2022].