Continuous Glucose Monitoring Prediction Method

Joshua O'Callaghan
CIDSE, Arizona State University
ASU ID: 1210714476
jdocalla@asu.edu

Alex Pappas
CIDSE, Arizona State University
ASU ID: 1210558450
apappas5@asu.edu

Sam Steinberg
CIDSE, Arizona State University
ASU ID: 1210415580
ssteinb5@asu.edu

Daniar Tabys
CIDSE, Arizona State University
ASU ID: 1213757322
dtabys@asu.edu

***Abstract*—We constructed and analyzed algorithms that developed a continuous meal detection method. We were given glucose monitor and bolus data for an individual over the course of six months in five minute intervals. First, we parsed the CGM data into two hours intervals and synchronized the CGM data with meal ground truth. Next, we developed two algorithms: 1) an auto regression based model and 2) a recurrent neural network based model. We then applied our models to a new patient. Lastly, we performed an execution time analysis comparison of the methods.**

I. Introduction

The advances in machine learning and data modeling have opened up incredible opportunities for new applications. Continuous glucose monitoring is a system that automatically track blood glucose levels throughout the day. The CGM works through a small sensor underneath the skin. The sensor sends the data to an external device where data can be retrieved and stored. Whenever you eat glucose levels spike, then settle back down in the coming minutes. Our goal was to come up with an efficient algorithm that detects when a meal is being eaten. This information is very important and has a variety of applications, including diabetes management among other things.

II. Author Keywords

CGM, RNN, Auto Regression, Bolus

III. Project Setup

The first part of the project was setting up the data. We were provided with six months worth of CGM sensor outputs data for a given person. In a separate file, we were given Bolus data with meal ground truth. If the Bolus value was high then the person was eating a meal at that time. If the value was low, they were not eating a meal at that time. Before we could train our models, we needed to parse the CGM data and synchronize the meal ground truth with the CGM data. I parsed the CGM data inside python which I will discuss in the next section. To synchronize the meal ground with the CGM data I simply joined the two columns together based off of the time column. Since the two time columns were the same for both variables, this automatically synchronized meal ground truth with the CGM data.

Next we decided on the programming language to write our models in. We decided to user Python since Python supports a variety of powerful machine learning libraries such as Tensor Flow for RNN and a library for SARIMA, used in the auto regression model. In addition, our group was very comfortable coding in Python, making the language easy to work with.

IV. Implementation

A. RNN based meal detection

1) CGM Data Parsing and Data Preperation: The first step was to import the Bolus and CGM data into a Python script. The data had been synchronized in Excel, and the Bolus data had been categorized as 0 (not a meal) or .99 (a meal) in Excel as well. Next I got rid of any null values by deleting any rows with null values. This did not decrease the dataset by a considerable amount, otherwise I would have interpolated the data. Next I created a series to supervised function that parsed the CGM data into two hour increments. This function helped my RNN model learn off of two hour intervals. Next I prepared by data to be fed into the RNN model. I feature scaled the data so it was normalized, split the data into x and y variables, and split it into a training and test set. I found that an 85/15 train-test split gave the most accurate results. I then reshaped the data into 3 dimensions: samples, timestamps, and features. This is the input shape for an RNN algorithm. The next step was the RNN algorithm development.

2) Algorithm Instantiation: The algorithm was developed using the Keras library inside of Tensorflow. I used the Sequential model inside Keras since we had one input (CGM sensor data) and one output (meal ground truth).

3) Algorithm Development: I decided to use a Long Short Term Memory (LSTM), a type of Recurrent Neural Network (RNN), since LSTM's are powerful when classifying time series data [1]. It also handles lag between important events, such as having null values in a dataset, with ease. I added a dropout layer with a value of .4 and two dense layers. The first dense layer had a "tanh" activation function while the second dense layer had a "sigmoid" activation function. I trained many

models an found this setup gave the most accurate results in terms of training and testing accuracy. The loss function used was "categorial crossentropy" since we are classifying the data as a meal or not.

4) Algorithm Implementation: I fit the model on the training data and validated it with the test data. To reiterate, the data was split into training and testing data previously in the program. The first 85 percent of rows was considered training data, and the last 15 percent as test data. I found fitting the model on 3 epochs with a batch size of 50 gave the most accurate results.

5) Training and Testing accuracy: I evaluated the training and testing accuracy using the evaluate function of the keras models library. The function found the RNN model has a training accuracy of 95.4 percent and a testing accuracy of 95.9 percent.

6) Applying RNN Algorithm to New Patient: To apply the RNN Algorithm to a new patient, I first reshaped the data. The training data remained the same, but the test data was the new data given for the new patient. This training and new test data was fit to the model, again using 3 epochs and a batch size of 50. Using the evaluate method in the models library, the training accuracy was 95.9 percent and the testing accuracy was 96.7 percent.

7) Execution Time Analysis of RNN Algorithm: I used the "timeit" method in Python to retrieve an execution time for the algorithm. I made a function that simply ran the algorithm once, ensuring timeit didn't time any other line of code that could take a considerable amount of time to run. Timeit found the RNN Algorithm model took, on average, 5.67 seconds to run. This is a long time and could be do to the complicated LSTM RNN used. In addition, the dataset was not small. Lastly, the batch size and epoch variables could have contributed to the time as well. Adjusting these numbers could lower the time it takes for this model to run.

B. Auto regression based modeling of CGM

1) SARIMA Algorithm: The Seasonal Auto-Regressive Integrated Moving Average (SARIMA) model is the same model as ARIMA but it also depends on the seasonality of the data, which is defined with the addition of P, D, and Q for the seasonal portion of the time series and the *m* is the number of observations per period of time. So the model is as follows:

$$SARIMA \underbrace{(p,d,q)}_{\text{non-seasonal}} \underbrace{(P,D,Q)_m}_{\text{seasonal}} \quad (1)$$

For the development of the SARIMA model, the statsmodels python package was used as well as Jupyter Notebook, Pandas, and NumPy. Statsmodels[2] provides a helpful setup instructions and examples using SARIMAX model, which was taken as the baseline for the code implementation provided.

2) Algorithm Instantiation: This section will describe the setup that was used to apply the SARIMAX model to our CGM data. The analysis done from this point on follows the template provided by statsmodel[2] and Kaggle[3], and was used purely as a reference tool to guide through the analysis and model application. First, to get a good grasp on how the data correlates plotting it is beneficial, and we end up with the following:

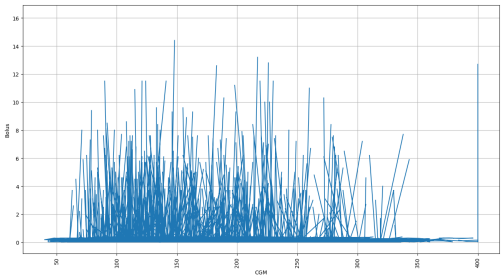

Fig. 1. CGM and Bolus data plot.

The data is erratic and is clearly not stationary, as the variance and mean are not constant through time, which is an indication of *heteroscedasticity*. As the usual plot did not provide any data, we can plot Partial Auto-Correlation Function (PACF, Figure 2) and Complete Auto-Correlation Function (ACF, Figure 3) to check there is any information to be obtained.

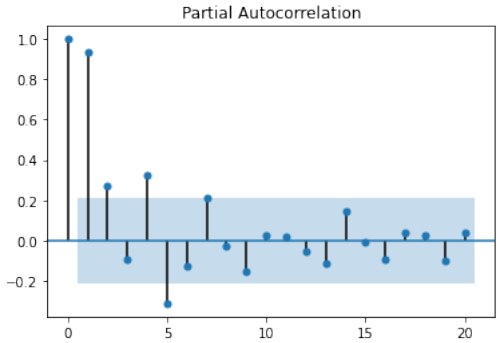

Fig. 2. Partial Auto-Correlation Function (PACF) plot.

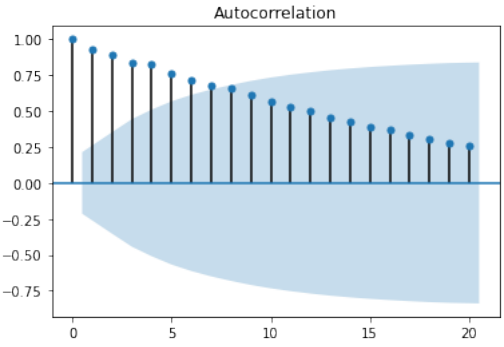

Fig. 3. Complete Auto-Correlation Function (ACF) plot.

Again, similarly to the Figure 1, but apart from the data that was already obtained before nothing stands out. Assuming that the time series is non-stationary we can apply the logarithmic difference in order to make it stationary, we obtain the Figure 4.

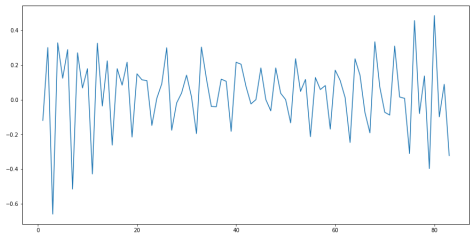

Fig. 4. Logarithmic difference of CGM data plot.

The application of the logarithmic difference has proved to be correct and the plot (see Figure 4) is stationary, but the is still the indication of seasonality of the data. Thus, if we apply the periodic nature of the data to the plot we will obtain Figure 5.

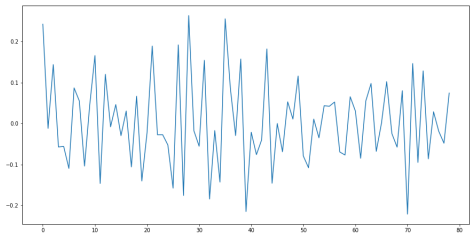

Fig. 5. Logarithmic difference of CGM data plot with periodic difference.

To test that the data is stationary we apply the Dickey-Fuller test, and see that the p-value is not significant enough to accept the null hypothesis, therefore we can confirm that the data is indeed stationary. Now, applying PACF (see Figure 2) and ACF (see Figure 7) on the adjusted data, from Figure 6 we can see that the a spike at *1*, thus indicating the seasonal auto-regressive process of order *AR(1)* and $P = 1$. Looking at the ACF plot (see Figure 7), we can see a spike at *1*, thus indicating the non-seasonal *MA(1)*. This provides us with the rough idea of the processes that correlet to our SARIMA model, but applying the optimize SARIMA function results in the *SARIMA(0,1,2)(0,1,2,4)* best fit scenario.

3) Algorithm Implementation: Now, after obtaining the best fit processes for the SARIMA model, we can apply the algorithm using the statsmodel's built in SARIMAX function. Which provides us with the results shown in Figure 8, which provides us with a model that contains both seasonal and non-seasonal processes.

4) Training and Testing accuracy: After obtaining the model, we can plot the residuals (see Figure 9). Inspecting the QQ plot, we can see observe a nearly straight line, suggesting the absence of systematic departure from normality. Taking a closer look at correlation statistics (Correlogram) implies that there is no auto-correlation between the residuals.

Thus, allowing us to plot the model along with the forecast (see Figure 10). As the plot shows a tight prediction and correlation between the model and the actual data, thus implying a positive forecast result. Evaluating the results, the model

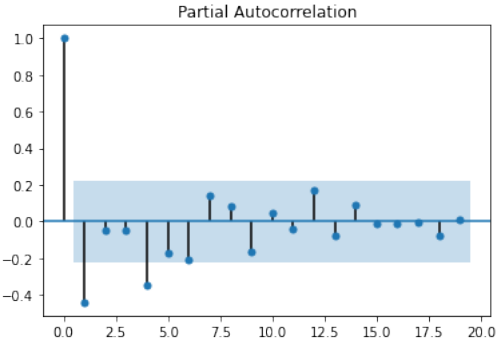

Fig. 6. Partial Auto-Correlation Function (PACF) plot.

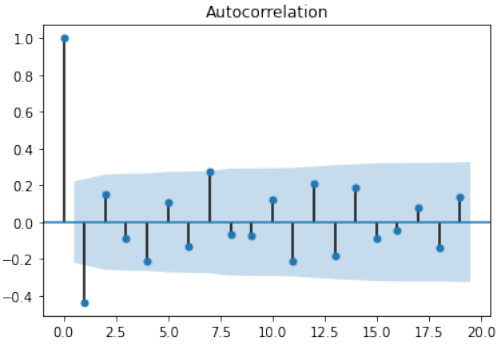

Fig. 7. Complete Auto-Correlation Function (ACF) plot.

Statespace Model Results

Dep. Variable:	data	No. Observations:	79
Model:	SARIMAX(0, 1, 2)x(0, 1, 2, 4)	Log Likelihood	62.232
Date:	Tue, 28 Jul 2020	AIC	-114.465
Time:	16:03:27	BIC	-102.944
Sample:	0	HQIC	-109.869
	- 79		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1	-1.5904	0.212	-7.502	0.000	-2.006	-1.175
ma.L2	0.5960	0.139	4.279	0.000	0.323	0.869
ma.S.L4	-1.2676	0.144	-8.825	0.000	-1.549	-0.986
ma.S.L8	0.4445	0.145	3.063	0.002	0.160	0.729
sigma2	0.0087	0.002	3.989	0.000	0.004	0.013

Ljung-Box (Q):	41.14	Jarque-Bera (JB):	1.09
Prob(Q):	0.42	Prob(JB):	0.58
Heteroskedasticity (H):	0.61	Skew:	0.27
Prob(H) (two-sided):	0.22	Kurtosis:	3.27

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Fig. 8. Statespace Model Results.

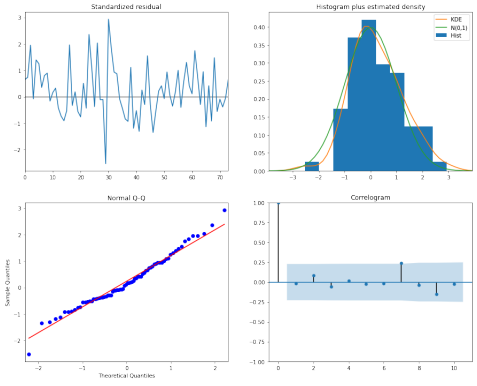

Fig. 9. Statespace Model Results.

accuracy results in 91.5 percent and the execution time being 1.35 seconds.

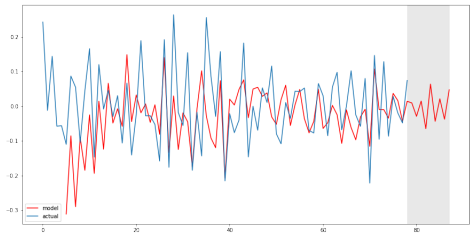

Fig. 10. Model forecast.

V. Completion of Tasks

Task	Description	Assignee
1	Parse CGM Data	Josh O'Callaghan
2	Synchronize meal ground truth with CGM Data	Alex Pappas
3	Develop Auto Regression Model	Daniar Tabys
4	Develop RNN Model	Sam Steinberg
5	Apply algorithms to new patient	Alex Pappas
6	Execution time analysis	Josh O'Callaghan

VI. Limitations

While the project was a success, there were some limitations. First off, some of the data contained null values. This required interpolation, meaning the the average of the values immediately before and after the null value(s) is taken. This new value fills in the null value(s). Our models may have been more accurate provided we had data with fewer null values.

The meal ground truth in the training/test set was another limitation. There were no specific guidelines on which bolus data was a meal and what wasn't, other than high numbers are meals and low numbers are not. Usually the values were easy to classify, since most of the numbers were extremely low (¡0.25), while others were quite high (¡1). Unfortunately there were some values between 0.25 and 1, making it difficult to classify them. Thankfully, these were the only limitations in the project.

VII. Conclusion

In this project, we made two algorithms to implement a continuous glucose monitoring method. We were given CGM sensor and meal ground truth data from an individual over the course of six months. The two algorithms were an Auto Regression based model and a Recurrent Neural Network based model. The Auto Regression was made using SARIMA and the RNN by libraries in Tensor Flow, both coded in Python. The training CGM data was parsed so the models learned in two hours samples of the data. The dataset was then synchronized with the meal ground truth data. Training and testing accuracy was gathered and compared with both models. The RNN was slightly more accurate than the auto regression model. Next, our algorithms were applied to a new patient to verify the models were not over fitting the training data. Lastly, execution times were taken from both algorithms and compared. The auto-regression was more efficient with an average run time of only 1.35 seconds compared to the 5.67 seconds of the RNN. Since the auto regression algorithm takes considerably less time to run and still has a very high accuracy (training and test both above 90 percent), I would choose the auto-regression model. Even so, both of these algorithms can be used to detect when a person is eating a meal which spikes glucose levels. This information is very useful as it is used in a variety of applications including diabetes management.

VIII. Acknowledgements

We would like to thank Professor Banerjee for giving us the opportunity to work on this project. We are all very interested in machine learning and classification, so the project aligned well with our interests. In addition, glucose monitoring impacts a large amount of people. Being able to use our passions of machine learning to have an impact on a real world problem was fulfilling.

References

[1] Mittal, Aditi. *Understanding RNN and LSTM*. https://www.towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e, Last accessed on 2020-08-13. 2019.

[2] Perktold, Seabold, Taylor, J. P. S. S. J. T. (n.d.) *SARIMAX: Introduction*. https://www.statsmodels.org/dev/examples/notebooks/generated/statespace_sarimax_stata.html, Last accessed on 2020-08-13.

[3] YUUT. *How to use SARIMAX*. https://www.kaggle.com/poiupoiu/how-to-use-sarimax, Last accessed on 2020-08-13. 2018.