Diabetes prediction and time series forecasting

Emily Tao <etao@hawk.iit.edu>
Yin Yang <yyang191@hawk.iit.edu>
Jim Witkiewicz <jwitkiew@hawk.iit.edu>
Geongu Park <gpark7@hawk.iit.edu>
Woojin Choi <wchoi6@hawk.iit.edu>

1. Abstract

Blood glucose monitoring is an essential part of diabetes management. Meals, activities, and insulin delivery all have an impact on the blood glucose concentration of a person, and great efforts are made to maintain the appropriate levels of blood glucose. In this project, we explore datasets from the subject of diabetes, including the Pima Indian Diabetes dataset and a dataset from the Tidepool database. We create classifiers for (1) diabetes prediction from the Pima dataset and (2) the trend of blood glucose values from features of the Tidepool dataset. Finally, (3) we model blood glucose time series data with a variety of methods including univariate ARIMA, LSTM, and multivariate LSTM. Then, we attempt to forecast future values using these models. No method is perfect, but we learned a great deal about the data from the modeling process. Ultimately the multivariate LSTM performed the most accurately out of the three forecasting models.

2. Overview

2.1. Problem Statement

Diabetes is a set of diseases characterized by the body's inability to produce or respond to insulin, used to process carbs from food. According to the CDC in 2020, just over 1 in 10 Americans have diabetes ("National Diabetes Statistics Report, 2020.")

Blood glucose monitoring is essential for diabetes management. With modern technology, patients with diabetes can continually monitor their blood glucose levels and adjust insulin doses, striving to keep blood glucose levels as close to normal as possible. Blood glucose levels that deviate from the normal range can lead to serious short-term and long-term complications. An automatic prediction model that warns people of imminent changes in their blood glucose levels would enable them to take preventive action. Further, automatic detection can enable automatic insulin delivery, lifting this burden from the patient.

In this project, we attempt to predict a subject's blood glucose level by applying time series models and deep learning techniques on CGM (continuous glucose monitoring) data. We utilize the Tidepool diabetes data, which is a time series data containing the data from blood glucose monitor users who are Type 1 diabetic. Specifically, we attempt to predict exact blood glucose values up to 40 minutes ahead (for ARIMA models) and 4 hours ahead (for LSTM models). We are also interested in knowing whether we can predict the trend (up/down) of the blood glucose values from either a classification standpoint, or using our time series methods.

2.2. Methodology

2.2.1. ARIMA

ARIMA (AutoRegressive Integrated Moving Average) models include autoregressive (AR) terms and/or moving average (MA) terms. An **autoregression (AR)** model forecasts the variable of interest by a linear combination of past values of the variable. The term autoregression indicates that it is a regression of the *variable* against itself. A **moving average (MA)** model forecasts the variable of interest by a linear combination of past forecast *errors* in a regression-like model (Brockwell and Richard, 2016).

By combining the autoregression (AR) and a moving average (MA) model, we can obtain a non-seasonal ARMA model. The full model can be written with the y terms coming from the AR model, and the Ɛ term coming from the MA model.

$$y'_t = c + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

Note that an ARMA model can only be applied on the "stationary" time series, which means the data does not depend on the time at which the series is observed. The *I* in ARIMA stands for *Integrated,* meaning that differencing can be used to obtain a stationary time series from the data. With the backshift notation (B), we can rewrite the full model of ARIMA(p,d,q) as:

$$(1 - \phi_1 B - \cdots - \phi_p B^p) \times (1 - B)^d y_t = c + (1 + \theta_1 B + \cdots + \theta_q B^q)\varepsilon_t$$

where *p* is the order of the autoregressive part, *d* is the degree of differencing involved, and *q* is the order of the moving average part. To implement an ARIMA model, we must determine the parameters *p* and *q*, and *d*.

There are two functions that can be used to apply the ARIMA model in R. One is to use the ***arima()*** in the ***forecast*** library. For this, we must customize the parameters p, d, q ourselves. The other method is to use the ***auto.arima()*** function that is also in the ***forecast*** library. This function will automatically determine the best p, d, q values for users, which seems to be a more appropriate method in practical situations.

We will first compare the forecast results by using the ***arima()*** and ***auto.arima()*** functions respectively. Then apply the ***auto.arima()*** function to different time periods of our data to try to obtain an overall performance of the ARIMA model on the CGM dataset by averaging the validation results.

2.2.2. Univariate LSTM

Long Short Term Memory (LSTM) neural networks are a special kind of RNN, capable of learning long-term dependencies. LSTM is explicitly designed to avoid the long-term dependency problem by their chain-like structure with four neural network layers interacting in a very special way.

There are four components that form the core idea of the LSTM network.

(1) The **forget gate** decides what information to ignore from the cell state. It depends on the value of h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. Here, 1 represents "completely keep" while a 0 represents "completely forget."

(2) The **input gate** decides what information to store in the cell state. It decides which values will be updated and how.

(3) The third layer updates the old cell state, C_{t-1}, into the new cell state C_t.

(4) The **output gate** decides what information to output from the cell. (Olah, 2015)

LSTM networks allow the formation of long-term dependencies which traditional models may fail to capture. For our project, we consider a univariate LSTM network to forecast the CGM outcome variable as an alternative to the ARIMA models and compare the performance.

2.2.3. Multivariate LSTM

So far, the models we discussed are univariate: they use only the previous CGM values to predict future CGM outcomes. However, we are interested in whether the other variables related to insulin delivery and activities can be used to forecast future CGM outcomes more accurately. Vector ARIMA methods exist, but our preliminary attempts to implement them failed to run. This may be due to the sparse nature of the variables leading to rank-deficient or ill-conditioned data matrices, which cannot be solved using least-squares methods. Therefore, we turn to deep learning methods to include the additional variables.

3. Data Processing and Exploration

3.1. Warm-up: Pima Indian Diabetes data set

At the beginning of the course, the group was concerned about going immediately into a time series analysis when most of the group was not experienced in it, nor would it be covered in class. Therefore, we found a data set on Kaggle to use as a warm-up to familiarize ourselves with the topic of diabetes and do a mini-analysis project.

Ultimately, we were able to correctly classify 79% of the observations as diabetic or non-diabetic using logistic regression. The full write-up of this classification sub-problem is included at the end of the report.

3.2. Tidepool data selection

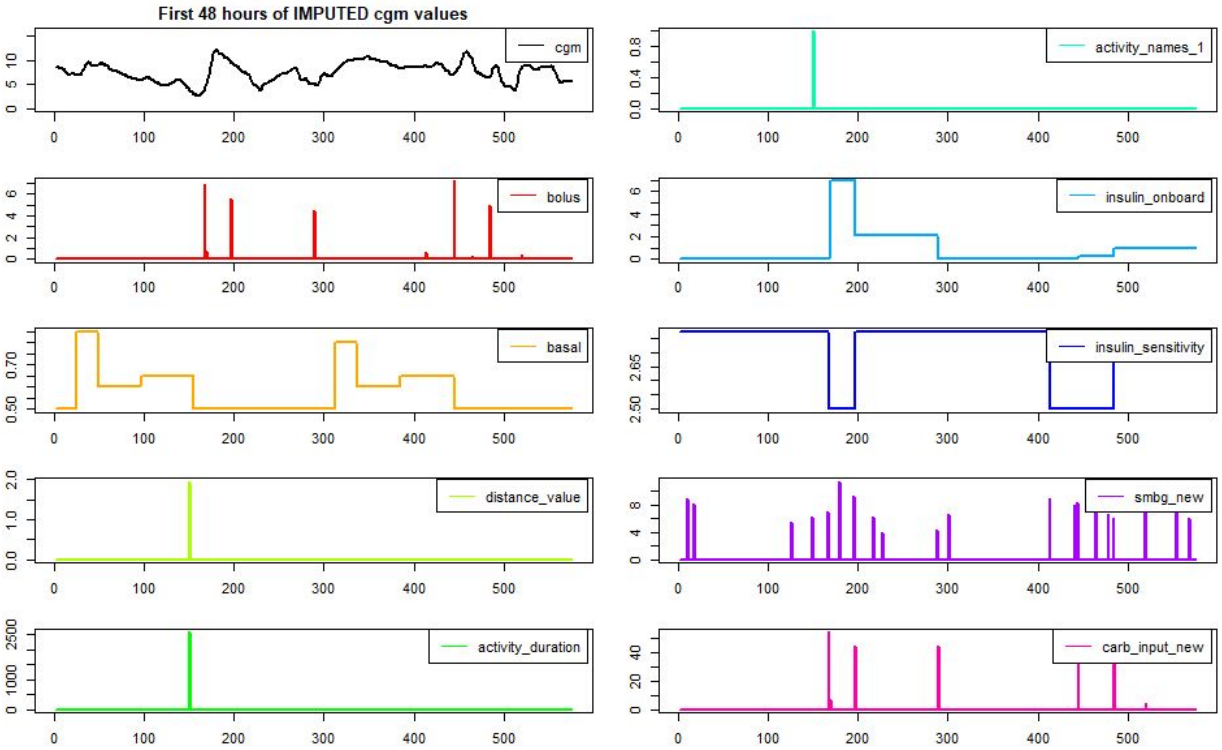
The Tidepool data is provided in the form of 300 CSV files, each corresponding to a recording from the continuous glucose monitoring device of one person. The files contain time series data of 20 variables sampled once every 5 minutes. We quickly realized that concatenating all of the data files into one data set would be practically impossible, due to both the size of the data (tens of thousands of rows for each data file), inter-subject variability, and time dependence for forecasting analysis. Therefore, we decided to select a single data file to develop our project. The file we selected is "**S_SET03_CASE005_M2_P2**."

3.3. Cleaning the data

The original data contains multiple time variables, both an epoch time and time split into day/month/year/etc. We combined all of these variables into a single column of POSIXct timestamps.

The two target variables referring to the user's blood glucose measurement in milligrams per deciliter, **IMPUTED** and **Denoised**, are virtually identical; **Denoised** simply has been put through a signal processing filter to remove some of the random noise in the data. These variables contain occasional missing segments where greater than 50 data points are missing in a row. For the majority of our analysis, we chose to use the less processed form (**IMPUTED**) because the Denoised version introduces more NA values around the existing missing sections. For the rest of this report, we refer to the **IMPUTED** variable as simply the CGM value.

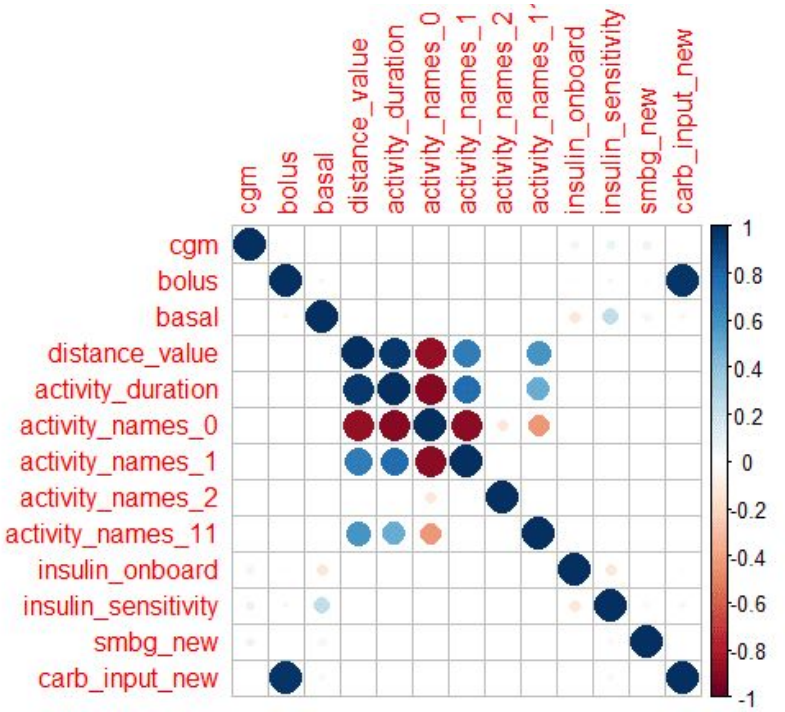
All the variables besides the CGM value, timestamp, and basal insulin (**basal**) value are sparse, containing many NA values. **Insulin_onboard** and **insulin_sensitivity** are sparse estimates of continuous features, so NA values were filled with the previous existing value. All other NAs were converted to 0 before inclusion in the project models. In addition, the variable **nutrition_carbohydrate_net** contains no observations. We disregarded it for our analysis. **Activity_Name** is a categorical variable with four levels; this was converted to dummy variables {**activity_names_0**, **activity_names_1**, **activity_names_2**, **activity_names_11**}. The following plot shows all the unique variables plotted across time with the same 2-day time window.



4. Data Analysis

4.1. Correlation between features

In addition to being sparse, several features are highly correlated. This is because the subject may have generated multiple records, for example **insulinOnBoard**, **carb_input_new**, and **bolus** at the same time. This makes logical sense as bolus insulin is a fast-acting form of insulin typically delivered at meal-times (Freeland, 2020). The following figure shows the correlation between each of the cleaned features. The correlation between features helps us decide which features to keep for our multivariate forecasting method later.

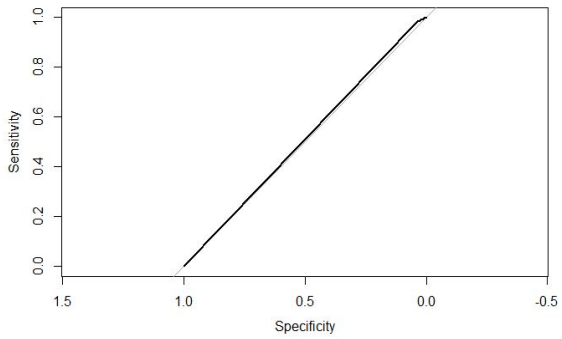


4.2. Classifying trend from data features

Before moving on to more complex time series models, we were interested in whether we could predict the trend of future data points from the data features, that is, whether the device user's blood glucose level would increase or decrease in the next 5 minutes. To study this, we implemented a simple logistic regression model with a categorical outcome variable corresponding to decrease (y=1) or increase (y=0).

Of the available features, only **bolus** was deemed significant with a p-value of less than 0.05. Using any linear combination of features resulted in a test accuracy of at best, 62.6%, using only the bolus feature. The confusion matrix and ROC curve of this classifier are shown below.

The simple classifier's performance in predicting the trend of the blood glucose level was relatively low. This suggests to us that the relationships between the data features and target CGM values may be weaker or more complex than a linear model can capture.



	Increase (true)	Decrease (true)
Increase (predicted)	14	9
Decrease (predicted)	1544	2585

5. Model Training

5.1. ARIMA

5.1.1. Stationarity & Differencing of Time-Series Data

In order to use an ARMA/ARIMA model, the data must be stationary. We can convert a non-stationary data set into a stationary one using differencing. Using tests for stationarity, we can determine the order of differencing to use in the ARIMA model.

The **Augmented Dickey-Fuller (ADF)** test allows for higher-order autoregressive processes by including Δy_{t-p} in the model. The null hypothesis is that the data are non-stationary. Therefore, we want to *reject* the null hypothesis by having a p-value <0.05. Using the aTSA package, we run the ADF test on time series differences of the CGM data. With single differencing, we get a p-value of 0.035 and reject the unit root hypothesis at level 0.05. Thus we can say that the once-differenced data is stationary.

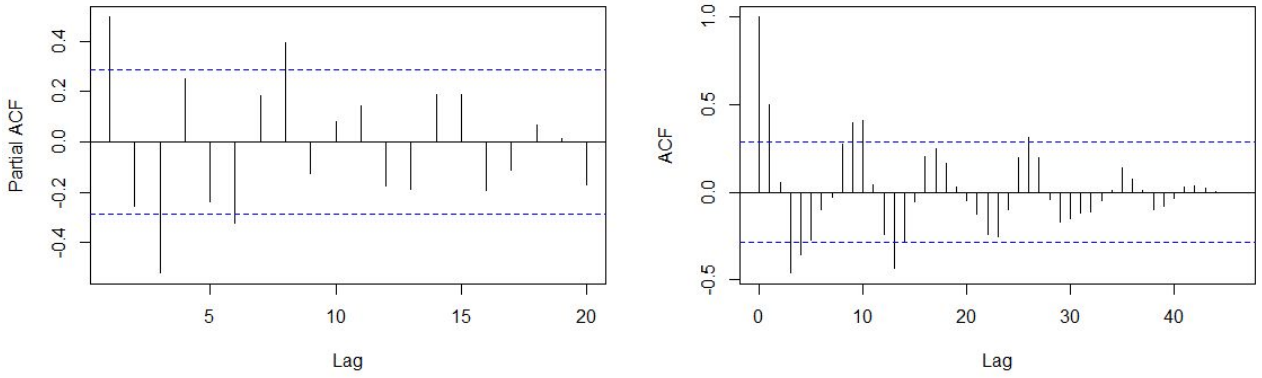
The **Kwiatkowski-Phillips-Schmidt-Shin (KPSS)** test is similar to the ADF test, but the null hypothesis is that the data are stationary. For this test, we do *not* want to reject the null hypothesis. With single differencing, we get a p-value of 0.1 indicating that we cannot reject the hypothesis that the data is stationary.

The results of these two tests are summarized in the following table. Both tests suggest that the appropriate order of differencing for the ARIMA model is d=1.

ADF		**KPSS**	
order	**p-value**	**order**	**p-value**
none	0.8966	none	0.01
1	0.035	1	0.1
2	0.01	2	0.1

5.1.2. Autocorrelation and partial autocorrelation

The **autocorrelation function (ACF)** measures the linear predictability of a time series, i.e., $\rho(s,t)$ is the predictability of x_t using only x_s (Shumway and Stoffer, 2018). It can be used to determine the order of moving-average models to use. The **partial autocorrelation function (PACF)** performs a similar function for autoregressive models. Together, we use both ACF and PACF to determine the orders (p, q) of our ARIMA model.
To estimate a model-order we look at: (1) At which number of lag that ACF/PACF dies out sufficiently? (2) At which number of lag that ACF and PACF show significant interpretable peaks? In the following figure, we plot the ACF and PACF at each lag. Since the ACF plot tapers to 0 in some fashion and the PACF pattern has significant values at first 3 lags; then gradually leads to zero afterward, we decided to select an AR(p=3), MA(q=0) model.



5.1.3. ARIMA model selection and training

For univariate ARIMA(p,d,q) models, we trained two models using either our user-selected ARIMA(3,1,0) or the auto.arima() function from the ***forecast*** library. When the auto.arima() function was used, the data was fitted to an ARIMA(3,1,1) model with drift.

```
Call:
arima(x = ts_trn[[20]], order = c(3, 1, 0))

Coefficients:
         ar1     ar2      ar3
      0.6357  0.0851  -0.3694
s.e.  0.1320  0.1604   0.1307

sigma^2 estimated as 0.01917:  log likelihood = 25.77,  aic = -43.54
```

```
Series: ts_trn[[20]]
ARIMA(3,1,1) with drift

Coefficients:
         ar1     ar2      ar3     ma1    drift
      0.1762  0.2661  -0.5893  0.4482  -0.0873
s.e.  0.1937  0.1350   0.1111  0.2402   0.0219

sigma^2 estimated as 0.01534:  log likelihood=33.33
AIC=-54.65   AICc=-52.55   BIC=-43.55
```

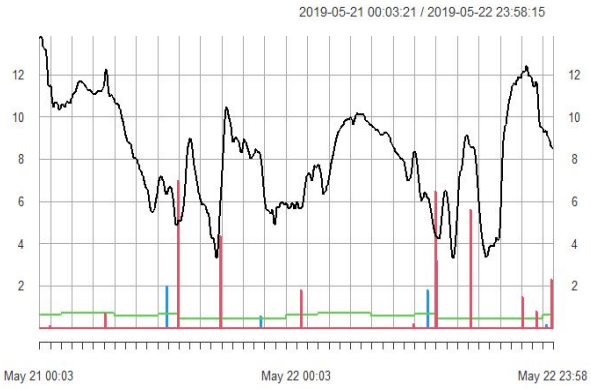
5.2. Univariate LSTM

The **Keras** R API was used to create the LSTM models from univariate CGM data. We guided our architecture and method by using several tutorials and examples and fine-tuned the model from there (R-Bloggers, 2018; Wanjohi, 2018). Multiple rounds of training and validation were used to assess the effect of using different model architectures. The models were trained by minimizing mean absolute error using Adam optimizer.

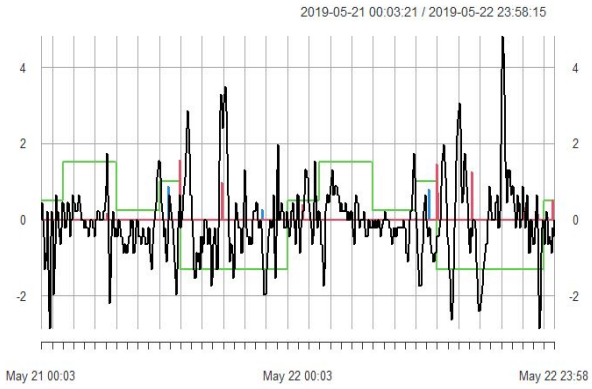
5.4. Multivariate LSTM

Due to high correlation with other features and lack of data of some of the variables such as **activity_names_2** and **smbg_new**, we decided to trim down our selection of variables for multivariate LSTM forecasting. Ultimately we settled on the feature set: {**cgm, bolus**, **basal**, **distance_value**, **activity_duration**, **activity_names_1**, **activity_names_11, insulin_onboard**}
The next step was rescaling and normalizing the feature variables so that the model would not be biased by extreme values. From the ARIMA analysis, we determined that single-order differencing would be helpful to obtain a stationary time series. The differenced CGM values were then normalized to have mean 0 and standard deviation 1. All other variables were rescaled to [-2,2] to be on the same magnitude as the target variable. Then, the data were time-lagged and reshaped to a matrix conforming to the necessary (batches x timesteps x features) format.

To compare adequately with the ARIMA models, two sizes of models were trained: (1) 1.5 day (432 point) input matrix, 4 hour (48 point) output matrix; (2) 4 hour input matrix, 40 minute (8 point) output matrix. The models were trained by minimizing mean absolute error using Adam optimizer with a batch size of 1, until training MAE plateaued (about 15 epochs).



Original data



Data after differencing and rescaling

6. Model Validation

6.1. ARIMA

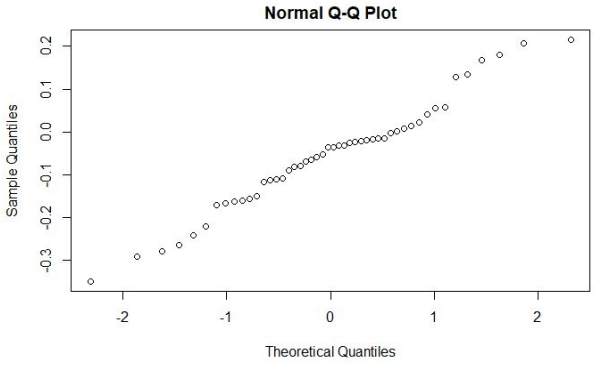
6.1.1. Residuals

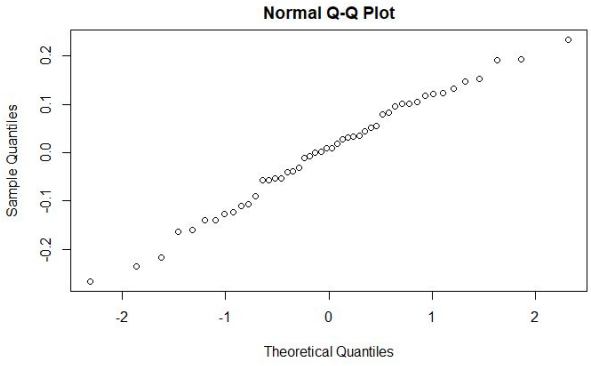
We inspected the residuals of the ARIMA models using quantile-quantile plots. The quantile-quantile plot (Q-Q plot) is a graphical tool to help us assess if a set of data plausibly came from some theoretical distribution. It is a scatterplot created by plotting two sets of quantiles against one another. If both sets of quantiles came from the same distribution, we should see the points forming a line that is roughly straight.

For our user-selected ARIMA(3,1,0) model (left), the points approximately fall along a line in the middle of the graph, but curve off in the extremities.This behavior usually means that the

data have more extreme values than expected. We are concerned that the residual might be heavy-tailed and might not have a strong normality.

In contrast, for the automatically generated ARIMA(3,1,1) model (right), the points form a straighter line down the diagonal of the plot. This suggests that the residuals follow a somewhat normal distribution and the model is a better fit for the data than the previous model.

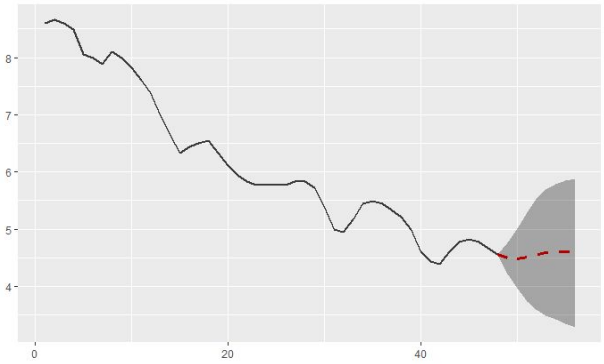




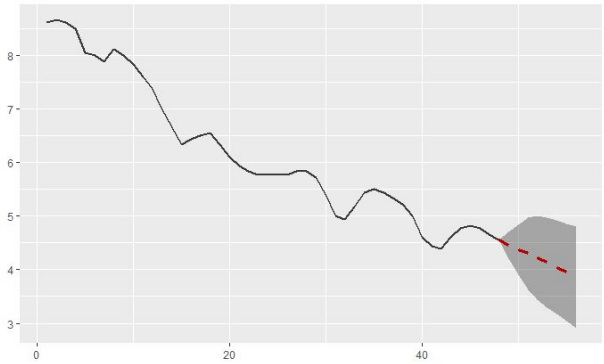
6.1.2. Predictions and performance

In the next section, we compare the prediction performance between our user-selected model and the automatic model. The following plots show a 56-point slice of the data, where the last 8 data points (40 minutes) have been forecasted by the model with 95% confidence intervals. In general, as shown by this example, the ARIMA(3,1,1) model performed better with smaller confidence intervals than the ARIMA(3,1,0) model.

ARIMA(3,1,0)



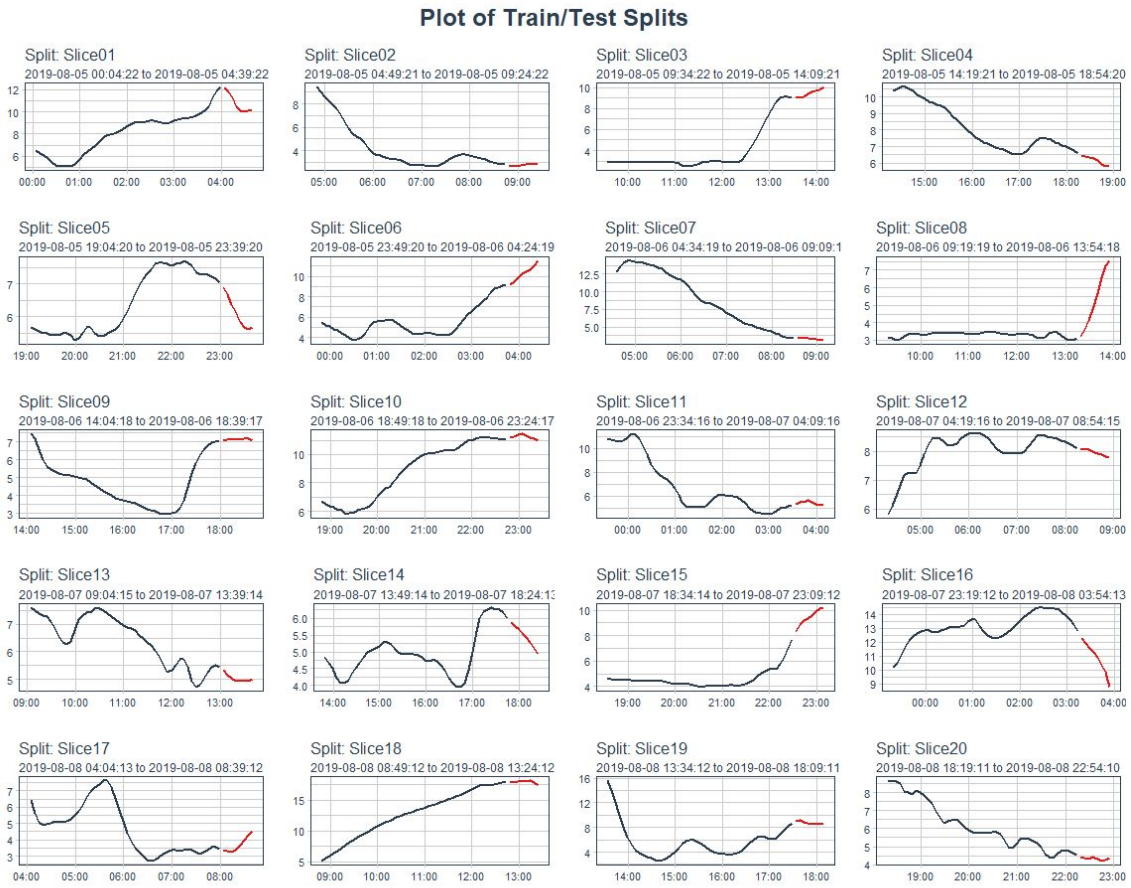
ARIMA(3,1,1) with drift

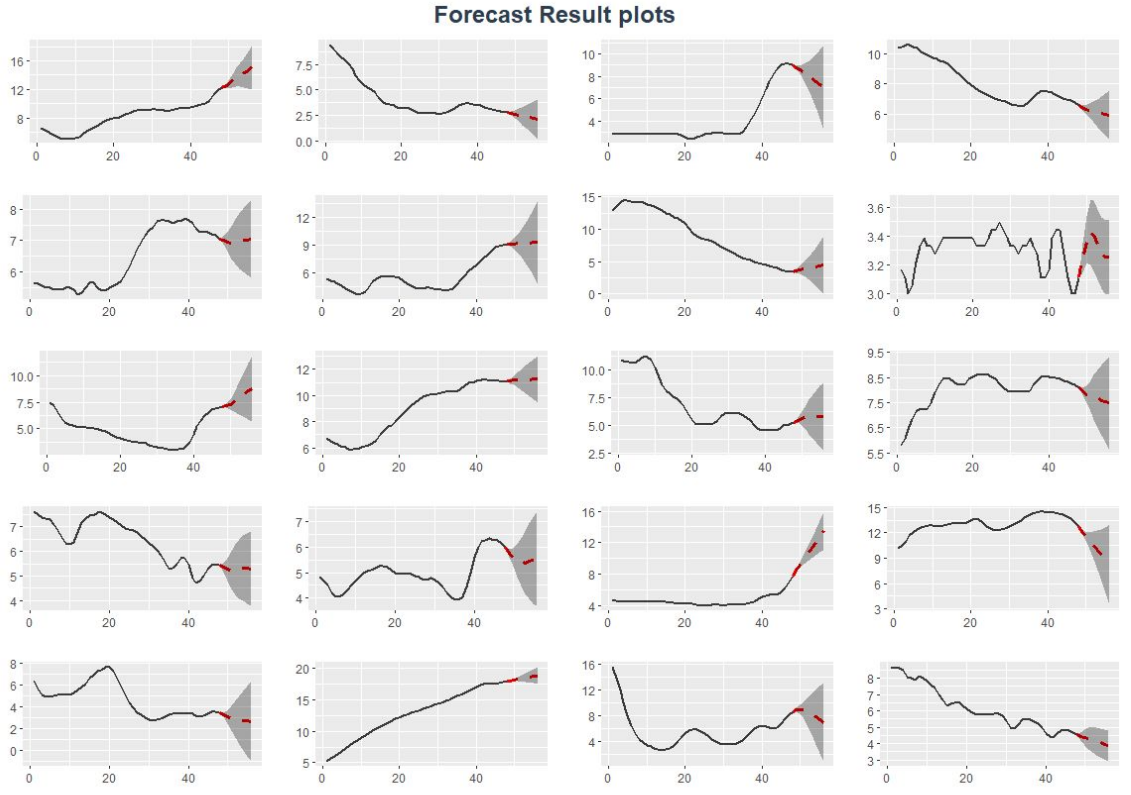


The performance of these models on the single slice is summarized in the following table. The ARIMA(3,1,1) model has lower error than the ARIMA(3,1,0) model. This observation holds throughout the slices tested on both models. Therefore, we select the ARIMA(3,1,1) model as our final ARIMA forecasting model. The following page shows the results for all 20 slices.

	ARIMA(3,1,0) *(slice)*	***ARIMA(3,1,1)*** *with drift (slice)*	***ARIMA(3,1,1)*** *with drift (average)*
Mean Absolute Error (MAE)	0.2913	0.2382	1.0896
Residual Sum of Squares (RSS)	1.0391	0.6757	18.9650

Mean Squared Error (MSE)	0.1299	0.0845	2.3706
Root Mean Square Error (RMSE)	0.3604	0.2906	1.2022


Plot of Train/Test Splits
Split: Slice01
2019-08-05 00:04:22 to 2019-08-05 04:39:22
Split: Slice02
2019-08-05 04:49:21 to 2019-08-05 09:24:22
Split: Slice03
2019-08-05 09:34:22 to 2019-08-05 14:09:21
Split: Slice04
2019-08-05 14:19:21 to 2019-08-05 18:54:20
Split: Slice05
2019-08-05 19:04:20 to 2019-08-05 23:39:20
Split: Slice06
2019-08-05 23:49:20 to 2019-08-06 04:24:19
Split: Slice07
2019-08-06 04:34:19 to 2019-08-06 09:09:1
Split: Slice08
2019-08-06 09:19:19 to 2019-08-06 13:54:18
Split: Slice09
2019-08-06 14:04:18 to 2019-08-06 18:39:17
Split: Slice10
2019-08-06 18:49:18 to 2019-08-06 23:24:17
Split: Slice11
2019-08-06 23:34:16 to 2019-08-07 04:09:16
Split: Slice12
2019-08-07 04:19:16 to 2019-08-07 08:54:15
Split: Slice13
2019-08-07 09:04:15 to 2019-08-07 13:39:14
Split: Slice14
2019-08-07 13:49:14 to 2019-08-07 18:24:1
Split: Slice15
2019-08-07 18:34:14 to 2019-08-07 23:09:12
Split: Slice16
2019-08-07 23:19:12 to 2019-08-08 03:54:13
Split: Slice17
2019-08-08 04:04:13 to 2019-08-08 08:39:12
Split: Slice18
2019-08-08 08:49:12 to 2019-08-08 13:24:12
Split: Slice19
2019-08-08 13:34:12 to 2019-08-08 18:09:11
Split: Slice20
2019-08-08 18:19:11 to 2019-08-08 22:54:10


Forecast Result plots

6.2. LSTM

6.2.1. Univariate LSTM

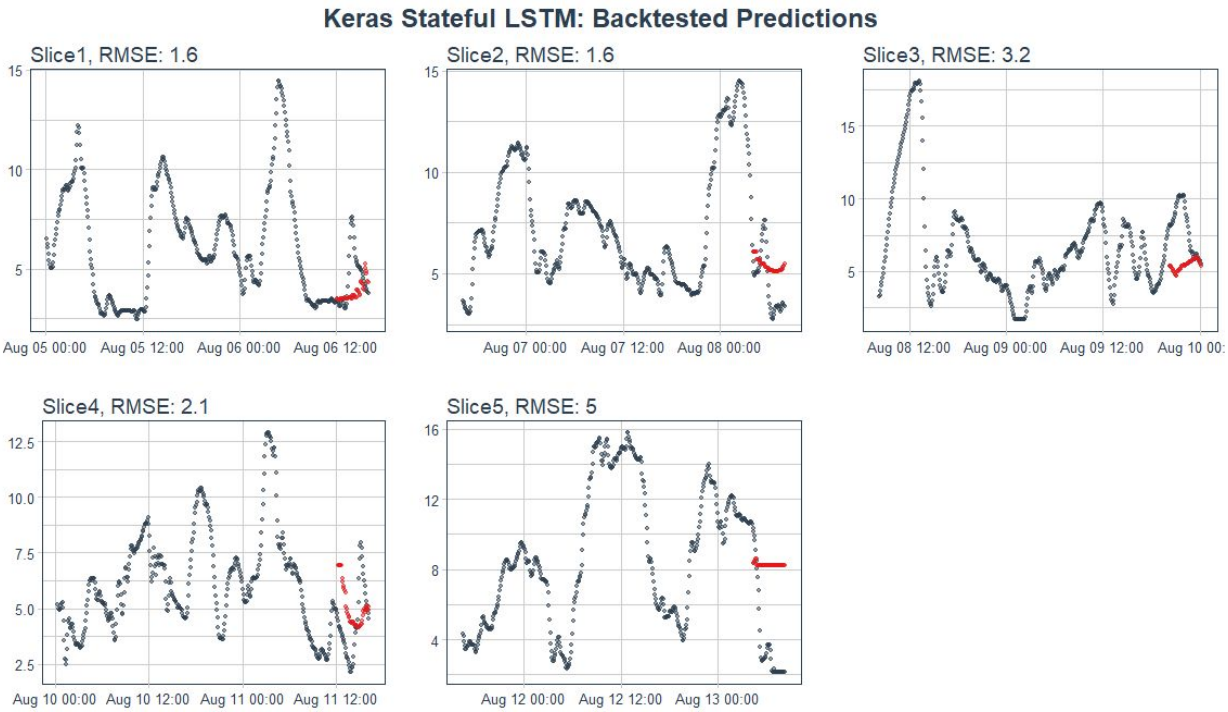
After multiple training and validation steps, the model architecture that performed best included a single LSTM layer followed by dense layers.

```
Model
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
lstm_3 (LSTM)	(6, 1, 1)	12
dense_12 (Dense)	(6, 1, 28)	56
dense_13 (Dense)	(6, 1, 8)	232
dense_14 (Dense)	(6, 1, 48)	432
dense_15 (Dense)	(6, 1, 48)	2352

```
Total params: 3,084
Trainable params: 3,084
Non-trainable params: 0
```

The following plots show the forecasts of the final univariate LSTM model using data from one and half day (12*36 data points) to predict the next 4 hours (48 data points) of CGM values. The average performance metrics are included in the table in the following section, showing the comparison between the univariate and multivariate LSTM.



6.2.2. Multivariate LSTM and comparison

Similar to the univariate case, the best model we found for multivariate LSTM prediction included 1 LSTM layer followed by dense layers.

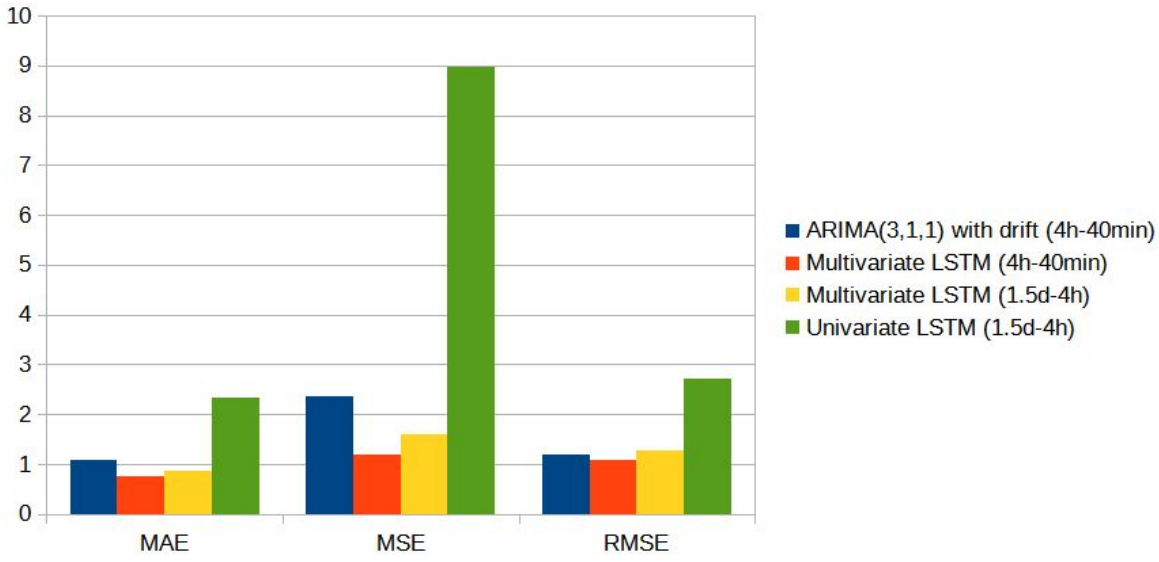
```
Model
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (1, 30)                   4560
_________________________________________________________________
dense (Dense)                (1, 50)                   1550
_________________________________________________________________
dense_1 (Dense)              (1, 8)                    408
=================================================================
Total params: 6,518
Trainable params: 6,518
Non-trainable params: 0
_________________________________________________________________
```

The first of the following plots shows the forecasts of the final multivariate LSTM model using data from one and half day (12*36 data points) to predict the next 4 hours (48 data points) of CGM values. Note that the predicted values themselves are differences. In order to reconstruct the forecast, the raw predictions are progressively added to the last input data point.

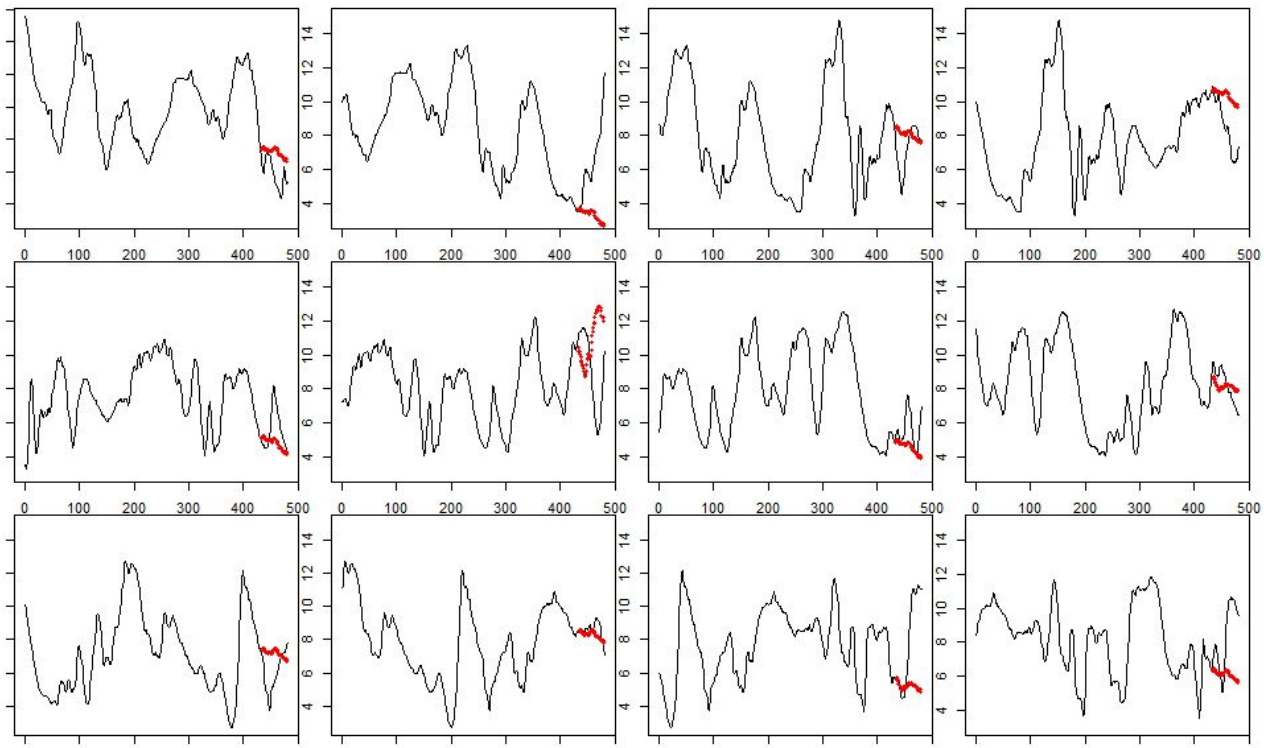
The second plot shows forecasts of the final multivariate LSTM model using 4 hours (48 data points) to predict the next 40 minutes (8 data points) of CGM values, the same time window we used in the ARIMA models.

In the summary (average) validation statistics, it appears that the multivariate model performed more accurately than the univariate model, even beating out the ARIMA (3,1,1) model on the 4-hour/40-minute splitting scheme.

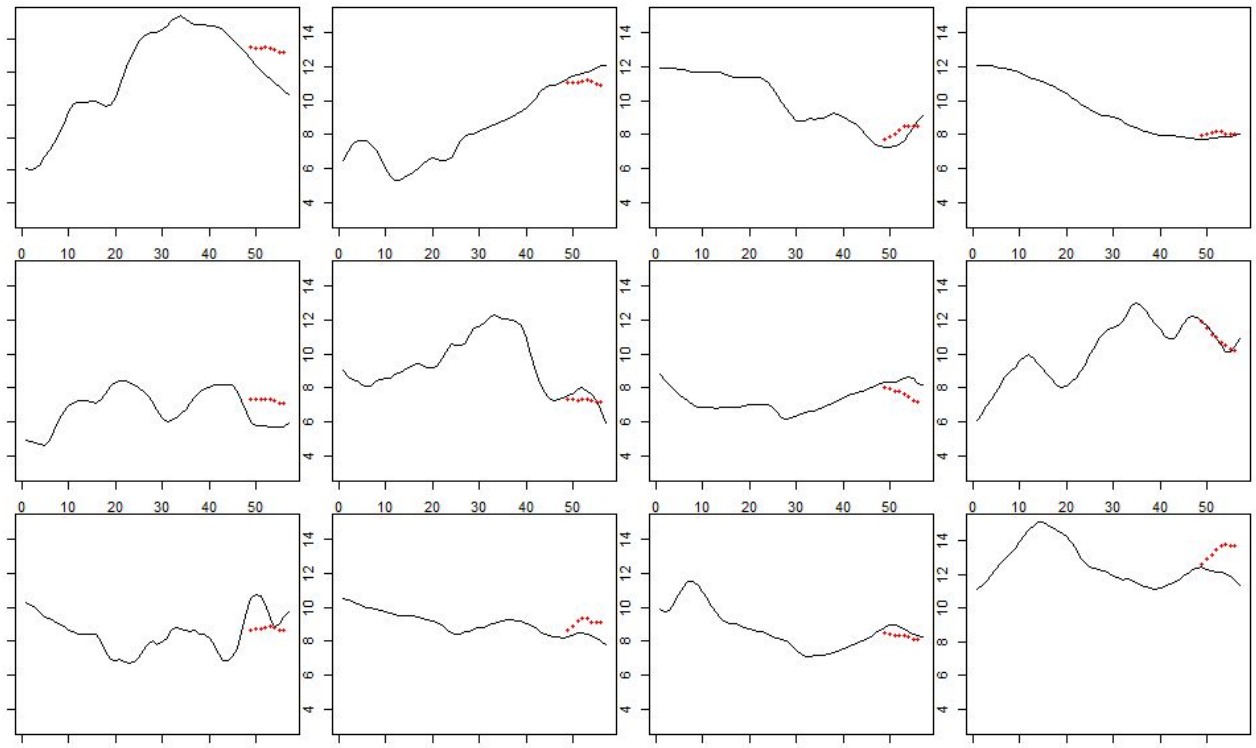
	Univariate LSTM (1.5d-4h)	*Multivariate LSTM (1.5d-4h)*	*Multivariate LSTM (4h-40min)*	*ARIMA(3,1,1) with drift (average)*
Mean Absolute Error (MAE)	2.334635	0.8569	0.7513	1.0896
Mean Squared Error (MSE)	8.969945	1.5984	1.1918	2.3706
Root Mean Square Error (RMSE)	2.707877	1.2643	1.0917	1.2022



Multivariate LSTM **Test Predictions**: 1.5d-4hr splits



Multivariate LSTM **Test Predictions**: 4hr-40min splits



7. Conclusion

7.1. Univariate Forecasting

We first choose to use the regression-based time series model, ARIMA, for forecasting, due to its high interpretability. We encountered a few problems using this approach.

(1) First, the time-series data we are using is quite irregular, which makes sense since the blood glucose level will change drastically in a short time due to factors such as diet, exercise, and stress, etc. In this case, it is not easy to decompose the data into the level, trend, seasonality, and noise components.

(2) Second, considering real applicability, it is unrealistic to specify parameters for the ARIMA model respectively for each slice of time series data. Models for each of the 4-hour slices were not consistent, and some slices could not be made stationary with differencing. For these reasons, the predictions made using the ARIMA model were not very effective, and even the trend of the data could not be forecast with much accuracy.

As an alternative to our ARIMA implementation, we did try the R package ***Prophet*** for a brief time. Since the Prophet can automatically detect trend changes by selecting change points from the data, we thought that it can help us improve the prediction. However, it did not work well on our data. According to the research paper released by Facebook, Prophet is a process of predicting time series data based on an additive model and is more suitable for business behavior data with obvious internal laws.

Lastly for the univariate case, we decided to use the artificial recurrent neural network method, Long short-term memory (LSTM), for our rolling time series data. It was a challenge for the group to manage the many factors that will influence the performance of the model, i.e. different lengths of train/test sets, batch sizes, number of model layers and nodes, which will all considerably affect the forecasting results.

7.2. Multivariate Forecasting

Although still having issues, the multivariate LSTM model performed the best out of all our forecasting methods, obtaining the lowest average MAE, MSE, and RMSE across all test slices. This is promising for future multivariate analysis, as it appears that the variables related to insulin delivery and activities are informative. However, due to the lack of interpretability of neural network models, it is difficult to assess which variables had the most impact. For future development, we suggest further analysis and treatment of the included variables in order to gain the most value out of them.

In addition, it is unclear how comparable the univariate and multivariate LSTM models are, as they were developed by different group members using different data processing methods to obtain the final training matrices. In order to compare them on equal footing, the univariate or multivariate case should be replicated using the same system.

How did we do, ultimately? Inspecting the forecast plots shows that by multiple modeling methods, the most common ‘prediction’ is simply that the blood glucose level of the subject will stay the same (or maintain the same rate of change as the last few data points). On a point by point basis, this results in the lowest error, but it is not very informative. More advanced methods may be needed to create a truly informative model for forecasting these data.

8. Data Sources

Continuous glucose monitoring data comes from the Tidepool Project, accessed in cooperation with the Cinar Lab at IIT. We credit the donors to the Tidepool Big Data Donation Project for this data. The data is not hosted online, but the file used for our analysis is provided with our source code for the duration of this course.

The supplementary Pima data set is sourced from Kaggle:
https://www.kaggle.com/jamaltariqcheema/pima-indians-diabetes-dataset

9. Source Code

Our source code can be found in the following Google Drive folder:
https://drive.google.com/drive/folders/11nMrHIyy-LT0Yfff9V4bvw__afOGOcPt?usp=sharing

10. Bibliography

[1] "Basal-Bolus Insulin Therapy: How to Use It, Benefits, and Risks." Medical News Today, MediLexicon International, 19 Mar. 2019, www.medicalnewstoday.com/articles/316616.

[2] Brockwell, Peter J., and Richard A. Davis. Introduction to Time Series and Forecasting. Springer, 2016.

[3] Christopher, Olah. "Understanding LSTM Networks." Understanding LSTM Networks -- Colah's Blog, 27 Aug. 2015, colah.github.io/posts/2015-08-Understanding-LSTMs/.

[4] Donors to the Tidepool Big Data Donation Project. Tidepool Project, 2020.

[5] Freeland, Megan N. "The 6 Types of Insulin and How to Use Them - GoodRx." *The GoodRx Prescription Savings Blog*, 8 Oct. 2020, www.goodrx.com/blog/insulin-types-how-to-use/.

[6] "National Diabetes Statistics Report, 2020." Centers for Disease Control and Prevention, Centers for Disease Control and Prevention, 11 Feb. 2020, www.cdc.gov/diabetes/library/features/diabetes-stat-report.html.

[7] R-Bloggers. "Time Series Deep Learning: Forecasting Sunspots With Keras Stateful LSTM In R." *R-Bloggers*, 18 Apr. 2018, www.r-bloggers.com/2018/04/time-series-deep-learning-forecasting-sunspots-with-keras-stateful-lstm-in-r/.

[8] Shumway, Robert H, and David S Stoffer. *Time Series Analysis Using the R Statistical Package*. Free Dog Publishing, 2018.

[9] Wanjohi, Richard. "Time Series Forecasting Using LSTM in R." *Richard Wanjohi, Ph.D*, 5 Apr. 2018, rwanjohi.rbind.io/2018/04/05/time-series-forecasting-using-lstm-in-r/.

11. *Supplemental* - Pima Indian Diabetes Classification Problem

10.1. Problem Statement

Pima Indian Diabetes dataset is a well-known dataset that is being used widely on machine learning prediction problems. The dataset contains 8 feature variables and 1 target variable. The objective of the dataset is to predict whether or not a patient has diabetes by analyzing the given feature variables such as Pregnancies, Age, Glucose, BloodPressure, DiabetesPedigreeFunction, BMI, SkinThickness and Insulin.

10.2. Data Processing

The following is a summary of the variables included in the data set. Several predictor variables have missing values. To have a better analysis on this dataset, we used average imputation to replace missing values with the mean of the predictor variable.

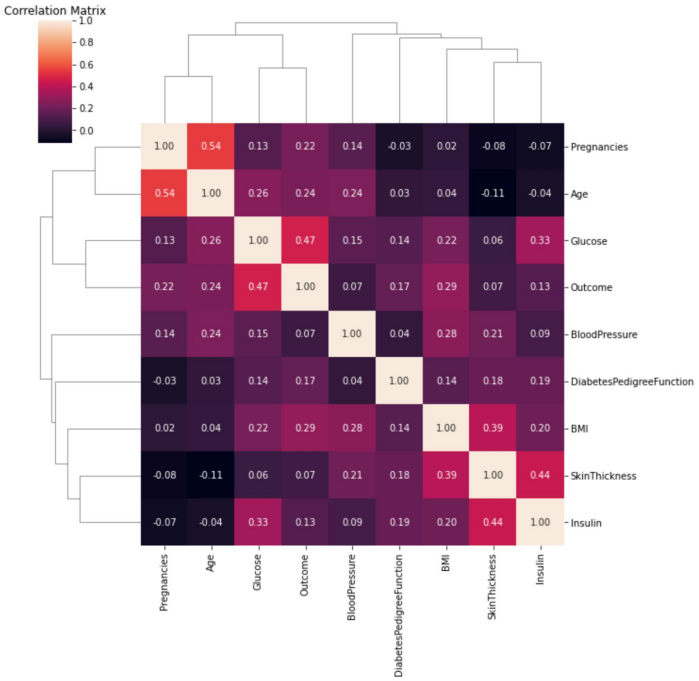
Column	# Observations	Missing
Pregnancies	768	0
Glucose	763	5
BloodPressure	733	35
SkinThickness	541	227
Insulin	394	374
BMI	757	11
DiabetesPedigreeFunction	768	0
Age	768	0
Outcome	768	0

The range of each predictors has huge differences between the predictors. For example, the maximum value of Pregnancies feature is 17 and the maximum value of Glucose feature is 199. Since the range difference is extreme, some features with extensive range will intrinsically influence our final prediction. To solve this problem, we have normalized every features' values to range from 0 to 1.

10.3. Feature Selection

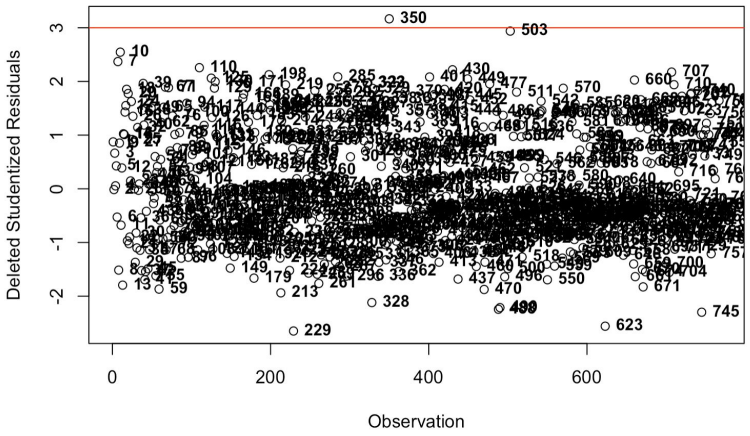
There are 8 predictor variables that we can use in our analysis. We want to check if our predictors have any multicollinearity to avoid possible reduction in statistical significance of the independent variables by calculating Variance Inflation Factor. If we detect any independent variables with higher than 10 VIF, we will have to drop them. Additionally, we can also take a look at correlation matrices. Since none of the VIF factors are higher than 10, and the predictors are not that highly correlated, we do not have to drop any predictors.

	VIF Factor	features
0	24.402204	Intercept
1	1.429998	Pregnancies
2	1.353074	Glucose
3	1.239114	BloodPressure
4	1.451069	SkinThickness
5	1.231335	Insulin
6	1.567784	BMI
7	1.045751	DiabetesPedigreeFunction
8	1.618502	Age

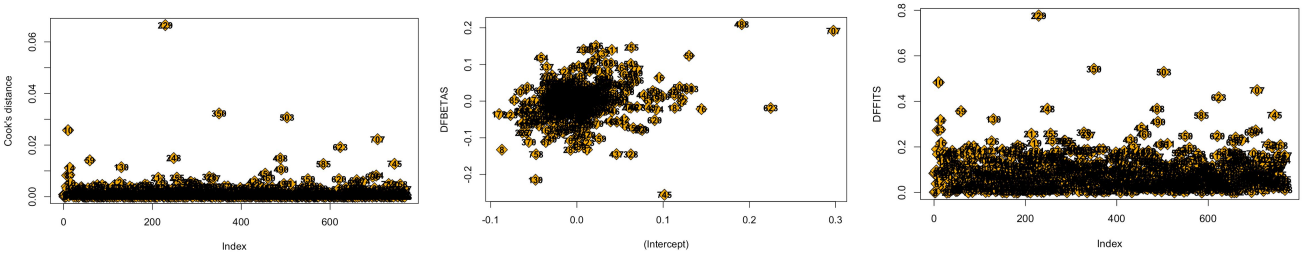


10.4. Outlier and Leverage Detection

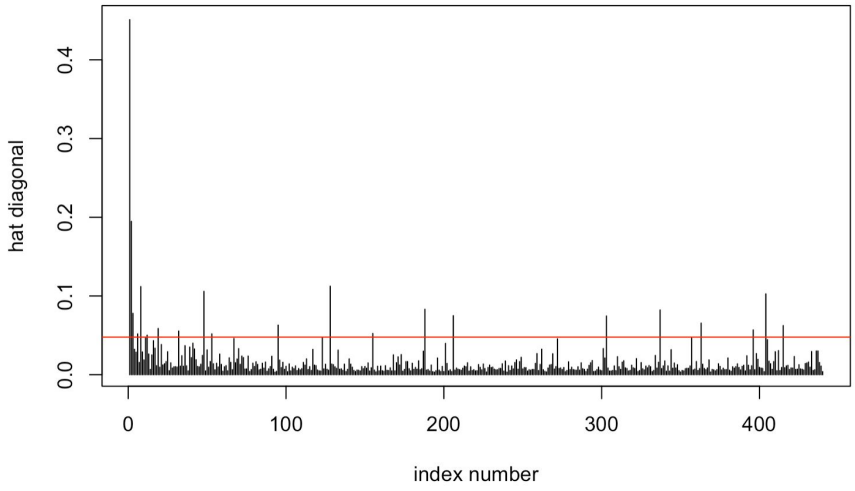
Studentized deleted residuals were used to detect outliers in the data. This metric is computed by dividing the residual of the deleted data point by its estimated standard deviation. We call a point an outlier when its absolute value of studentized deleted residual is larger than 3. Observation 350 was found to be an outlier.



Using Cook's Distance, DFBETAS, and DFFITS plots we found observation **229** to be an influential point.

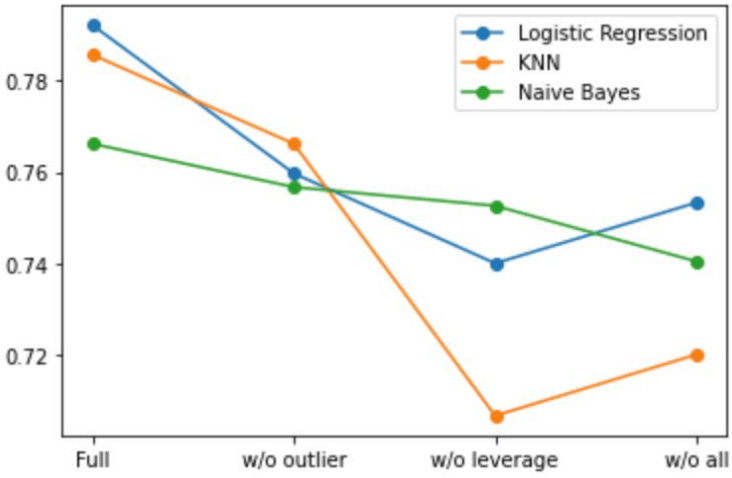


For leverage detection, we obtained the diagonal elements of the hat matrix. Using the rule of thumb, 20 extreme X values were identified.



10.5. Model fitting and validation

Logistic regression, KNN, and Naive Bayes classifiers were trained on the data set using all predictors and all observations, excluding outliers, excluding high leverage points, and excluding both outliers and high leverages. The validation accuracy is summarized in the following figure. The best performing classifier was logistic regression using all data points, with an accuracy of 79.22%



10.6. Conclusions

We have found that the performance of the logistic regression is better overall. However, deleting outliers or leverage points actually made the performance worse overall. This suggests that even though these points are outside the common ranges of the data, they are still informative to help classify the data.