Analysis of Diabetes Dataset

Stephen Valente

IT-460

3-3-2022

**Section I**

**Introduction**

The dataset that will be analyzed is the AIM 1994 Diabetes Dataset. This dataset contains 4 feature columns spread over 70 patient records, with each documenting anywhere from weeks to months of observations. There is no specified algorithm in the README, Domain-Description, or Data-Codes files. The 4 feature columns are the date, time, code (to denote different actions) and resulting blood glucose level. The patients in the data suffer from IDDM (Insulin-dependent diabetes mellitus), where, to summarize, their pancreases don’t produce enough insulin. Insulin therapy helps to keep BG levels closer to a normal range, but there’s the risk of hyperglycemia, which can lead to micro and macrovascular problems. The purpose of this dataset is to better understand how certain actions affect an individual’s blood glucose levels over time. This can be used to assess how particular actions may affect BG levels.

Although there isn’t a specified algorithm in the README, Domain-Description, or Data-Codes files, this dataset could make great use of a time-series model. The reasoning behind this is that one insulin level reading is certainly dependent on those before it, showing a need for the model to be stateful. The 20 data codes in the data can be one-hot encoded, and the date and time columns would have to be numerically encoded as well. The BG level would be the dependent variable. Given that the data would contain many columns as a result of the data code column being one-hot encoded, multiple linear regression would probably be the best model for this.

This model will add value to any medical organization by allowing medical practitioners to predict BG levels based upon action taken and previous readings, hopefully with a high degree of accuracy. This would prevent the possibility of hyperglycemia and resulting side effects. Furthermore, with less side effects, patients would more than likely spend less on medical expenses. This would lead to an overall better experience for patients, which would increase the medical organization’s reputation.

The tool I will use in this analysis will be R. I’ll start by encoding any data necessary, which in this case would be the date, time and code columns. The BG level column can be kept the same, since it is a continuous value. When thinking about how to encode the date and time columns, I reasoned that it is necessary to track the time difference between and time from the beginning without giving any importance to the objective time frame. In other words, time is only important relative to the last reading taken. That being said, I think it would be best to encode this as an integer, representing the number of seconds between readings. This can be done by using the difftime function in R, with a unit of “seconds.” The only downside is I would have to loop through the data and do this for every observation, which could be performance intensive. Next, I would want to encode the “code” column in this data by one hot encoding. The reason for this is that there’s no “order” to the codes (they’re not continuous values), but they’re merely representations of some action taken. This can be done by using the function dummyVars() from the “caret” library. Next would be to run the lm function using the newly structured data as a preliminary test. Once results have been assessed by analyzing the R-squared score, I would probably try adding interaction effects (mostly with time on the one-hot encoded code columns), as well as adding a certain experimented-upon previous number of observations in each row to account for the importance of time.

I believe that for this analysis, and probably many others using machine learning, R would be the tool of choice. Some other competitors that exist are Python and MATLAB. Python is a sort of “Jack of all Trades,” having packages that span web development, machine learning, mobile development, and much more. It is an incredibly popular open-source language, with plenty of people having contributed to it since it’s conception. It has packages like Tensorflow, PyTorch, OpenCV, and many others that make expansive machine learning tasks easier, and it also happens to be completely free to use. All of this being said, there isn’t much of a need of a wide variety of tools and packages to help with this project, and given that this is mostly a statistical analysis utilizing regression, along with the fact that R was created for statistical analysis explicitly, R outshines Python. MATLAB is another contender that makes machine learning easy, with it’s easy to use functions and emphasis on matrix programming. However, MATLAB isn’t free, and much of what can be done in MATLAB can be done in R.

**Section II**

**How To**

The first step in getting the data set up properly for this model is to download the tar from the proper source, https://archive.ics.uci.edu/ml/machine-learning-databases/diabetes/. Once this is downloaded, you can open the source by either double tapping on MacOS or using the winzip tool on windows. Once done downloading the data, be sure to read at least briefly through the README, Data-Codes, and Domain-Description files to understand more about/review how blood glucose can cause problems in diabetes patients if gone untreated. The next step would be to review the data itself, which includes a date/time stamp, data-code which depicts an action taken, and resulting blood glucose level; where all records are split into 70 different patient files.

The ultimate goal of this model is to predict a future reading to tell how a certain action might affect said future reading. This being said, the way the data will be restructured is so that there will be a 5-record span contained within one new record. This will mean a data-code, blood glucose level of 5 records ago; a time between the 5th and 4th record prior, 4th record data; and so on. I will make a tool that will restructure the data so that it can capture a certain number of records back into single rows to give the model “memory”. With the data restructured this way, this depersonalizes the data, allowing for a more general model that can be used by anyone.

From this you can either re-train the model created (more than likely a neural network, due to its ability to handle more than non-linear dependencies) with a different amount of prior records reference, or you can try utilizing any other method of your choice.

**Ethical Considerations**

The main ethical consideration is to not use this data/model to discriminate against people. This should not be used by medical insurance companies by people to determine the likelihood of whether someone is going to develop complications in a greedy attempt to charge higher premiums ahead of time. This should not be used in any other capacity than to help people who need it. Insulin is already charged very high in the US (and healthcare alone). This work isn’t groundbreaking, but people working on these types of projects should be doing it to help others, not exploit them.

**Ethical Use**

A large potential risk of misusing the data/model used/created is creating bias during acceptance of coverage. According to the HHS.gov website, “Health insurers can no longer charge more or deny coverage to you or your child because of a pre-existing health condition like asthma, diabetes, or cancer, as well as pregnancy. They cannot limit benefits for that condition either. Once you have insurance, they can't refuse to cover treatment for your pre-existing condition.” This means that if you ran an insurance company and decided to start doing this, you could very quickly find yourself knee-deep in settlements. Expanding off this, any entity (not just medical) that used this project to exploit others in any amount of a similar capacity could potentially find themselves in similar legal troubles.

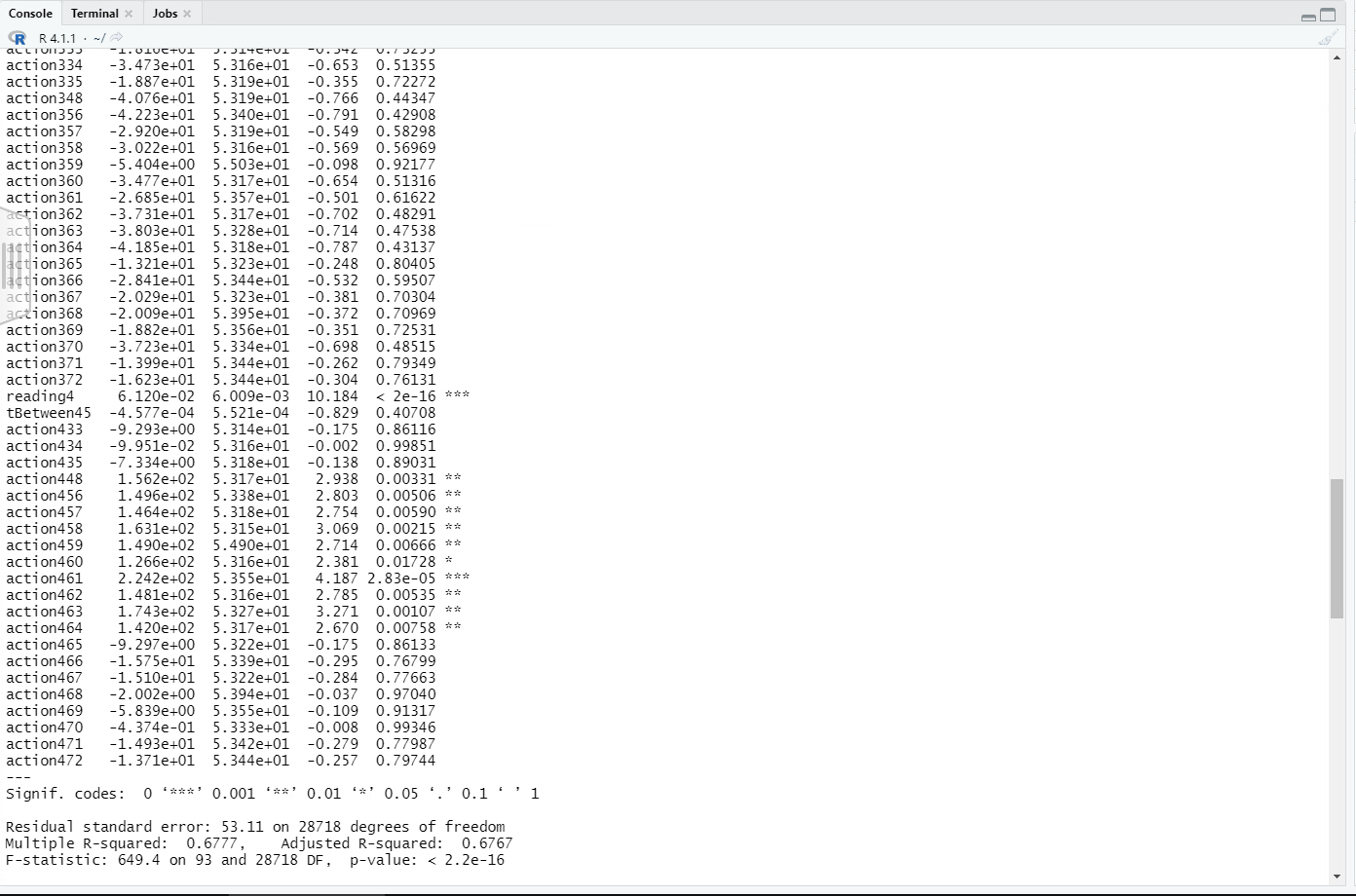
**Section III**

**Initial Execution**

The initial execution was with a basic multiple linear regression algorithm, using the lm() command. RNNs can be quite lengthy to train, and often times can suffer from vanishing or exploding gradients, so I opted to simply restructure the data in a way that resembled time series data, wherein it started with a first reading, time between first and second, action code, resulting BG reading, and etc. The original lm() command used the last reading as a target variable, with all others being feature variables. Feature variables originally went five readings back, and included the aforementioned columns. The script has been attached for review.

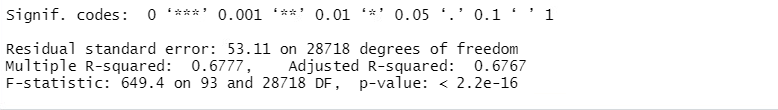
**Initial Evaluation**

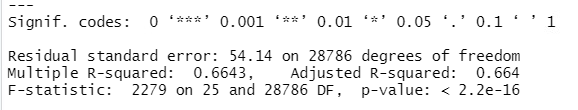
Running this model as is with all variables included resulted in an R-Squared score of .67, seen below. Given that this figure shows that the model is able to explain 67% of variance in the data, it’s a fairly decent score, but not great. We can also see that the p-values for many of the categorical “action” variables, as well as the “timebetween” variables were quite high (indicating lack of statistical significance), with the exception of the respective columns before the 5th reading.



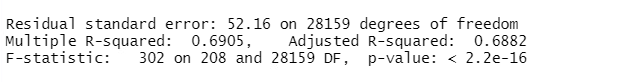
**Parameter Changes**

The first modification was attempting to normalize all numeric data using min-max normalization, but this resulted in the same score:

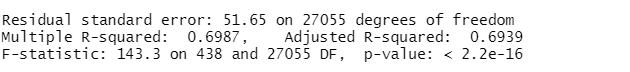


The next modification made was excluding columns that had a higher p-value, indicating less statistical significance. Most of these were all action and time between columns before the last reading. This resulted in a lesser score than all variables being considered

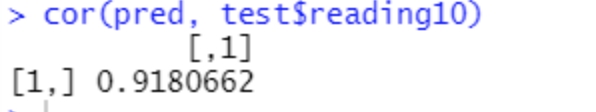
The last set of modifications to the linear model was adjusting the number of records taken into account to predict the final reading. This was started by doubling the number of records from 5 to 10, which resulted in an even better R-squared value:

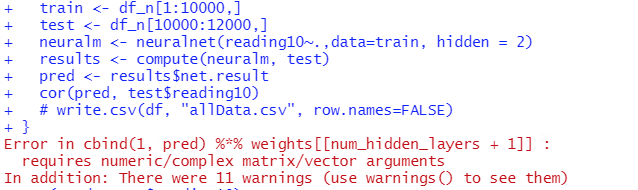


Although increasing from 10 to 20 examples prior didn’t result in much better of an improvement:

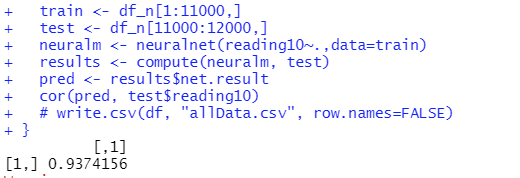


The next major modification to be made was the model itself. Maybe the data has non-linear relationships that the prior method couldn’t map out given the complexity of the data and the number of columns. With that, the decision to utilize a neural network was made. I started with one-hot encoding the action variables by using the dummyVars command, and then performed normalization on the dataset. There were a few column names that popped up when running this command, so I had to trim those from the dataset. I then split the data into a train and test set. For this part, I didn’t utilize the full dataset, opting to only use 12000 records, with 10000 being used for training. This decision was due to the fact that the amount of columns present and the environment which training took place was non-performant, resulting in longer training times. However, even with one hidden layer (the default for the neuralnet command) and after standard min-max normalization, the results were significantly better than the multiple linear regression model, with a 91.8% correlation to the original data.

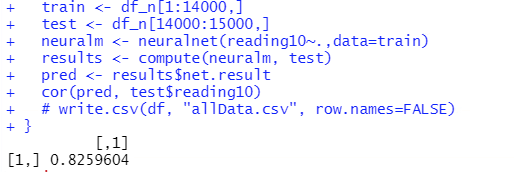


When attempting to utilize 2 hidden layers, the model took very long to train (hours) and when it finally completed, there was an error being thrown:  


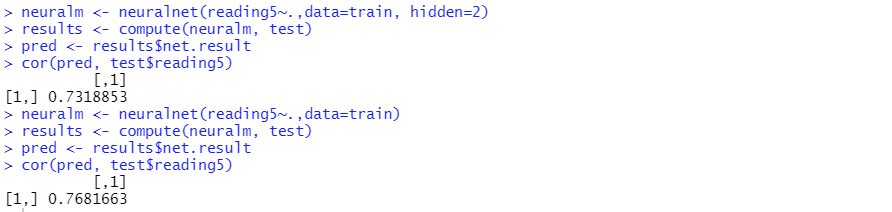
Because of this, it would be best to stick to a hidden parameter of one for now. However, another parameter that can be tuned is the train/test split. When running a split of 11000/12000, the results were even better:



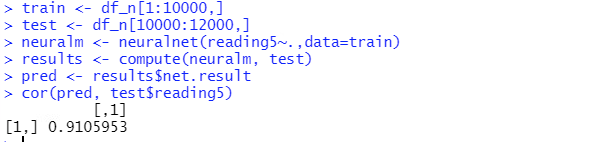
Although attempting to boost this even further resulted in more poor results:



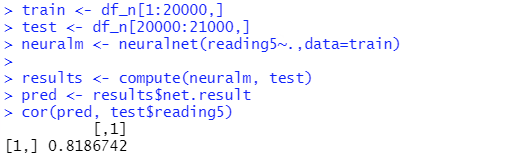
I then got curious about using less prior examples. Both of the models below were trained using 5000/1000 train test splits respectively. Oddly enough, the model with less hidden layers performed better:



When increasing the number of examples, the correlation continued to go up:

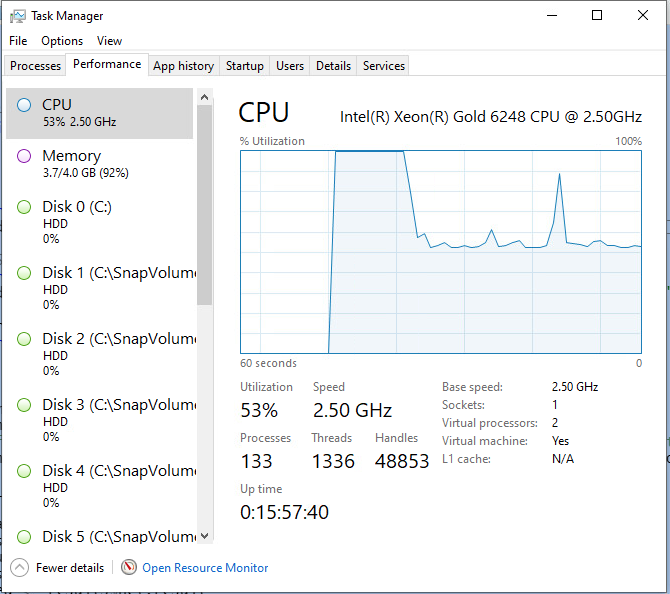


But again, up to a point:



**Parameter Confirmation**

The confirmed best model will be the neural network, as seen by the very high correlation between the target and the predicted variables. The number of hidden layers should remain at 1, at least until a better/more performant computing environment can be used. The number of examples used will also remain lower at 12,000 total examples, due to the length of time that a higher number of examples took to train. The performance issues are easily explained when looking at the specs of the computer that this model was trained on (4GB RAM, presumably no graphics card/GPU Acceleration). Then again, being able to train as well as this model has on a computer with these specs is encouraging, given that a proper PC with a high end graphics card to take care of parallelization/GPU acceleration and better/more RAM would undoubtedly allow for a much larger, more robust model to be trained. Due to evidence in terms of the correlation performance, the train/test split of 11000/1000 is recommended.



**Organizational Impact**

With a correlation this high, and given just ten previous records of a patient, the organization can fairly confidently predict the resulting blood glucose levels from a particular action taken. This means that doctors and patients can more confidently assess which actions will have what potential consequence on the patient, allowing them to take actions that will keep their blood glucose levels at a reasonable, safe level. This will mitigate the risk of emergency situations, ultimately saving the patient more money.

**References**

*Pre-Existing Conditions*. Retrieved March 26, 2022 from https://www.hhs.gov/healthcare/about-the-aca/pre-existing-conditions/index.html.