**Contract Deliverables and Progress Report: PLEXOS Insights**

Contractor: Allyson Tom

July 31, 2019

**Overview**

The goal for this portion of the PLEXOS Insights project is to be able to take fundamental model results (for example, solution data from a PLEXOS simulation), feed those results into some statistical model, and to quickly create good predictions for variables of interest using machine learning. This involves several intermediate steps. The first step was to automate the data gathering process, as an effective model will need lots of data to train on in order to generate meaningful results. I would consider this phase of the project complete. The second step is to create some machine learning model. The third step, then is to tune that model with lots of data so that it can generate meaningful, predictive results. The second and third steps have results ready, as will be discussed below, but would be improved with further work.

**Deliverables**

This report categorizes the following deliverables as either “Complete” or “In Progress” and discusses each in relation to the others.

* get\_data.py – a Python script that pulls data from PLEXOS solution files in a specified folder system to a SQLite3 tuning database
* PLEXOS Test.XML – a PLEXOS file that sets up 27 test scenarios (variations on high, normal, and low cost to load and fuel prices) used to initially fill the tuning database and create machine learning models
* PLEXOS Data Solution Files (folder) – PLEXOS solution files generated by PLEXOS Test.XML, whose results are pulled and processed by get\_data.py and so on
* jsontest.json – a sample .json file which supplies metadata to get\_data.py
* TuningDB.db – a SQLite3 database which holds all data made available by get\_data.py
* process\_sql\_data.py – a Python script which grabs data stored in TuningDB.db and prepares it for use by modeling.py, including processing it into a Pandas DataFrame and writing it to a .csv file
* modeling.py – a Python file for creating and tuning a model using TensorFlow and Keras, and for generating results with the model

NOTE: These files are all made available on the GitHub repository “shiny-rotary-phone” hosted by Dr. Steven Broad, and also in the .zip file “Deliverables”.

**Complete**

At the time of this report, the deliverables discussed in this section are considered complete for the purposes of this phase of the PLEXOS Insights project.

* get\_data.py
  + This Python script automates the data collecting process and allows the user to specify which data to query from PLEXOS solution files. A folder system containing several PLEXOS solution files and folders can be traversed easily and data of interested collected with the use of this script. A .json file should be specified in the script to provide metadata, such as which variables are of interest to the user, the root folder of the PLEXOS files, the eventual prediction variable(s), and a key for the tuning database.
* PLEXOS Test.xml
  + This file sets up a simple PLEXOS simulation with two generators, two fuels, and 27 scenario combinations of high, normal, and low load and fuel prices. The results generated by this file is captured in the folder “PLEXOS Data Solution Files”. This is the current test case for generating results.
* PLEXOS Data Solution Files (folder)
  + As discussed in the bullet point above, this folder holds solution data from the 27 scenarios set up by PLEXOS Test.xml. Currently, get\_data.py pulls from this folder, and this data is fed into both the tuning database and the machine learning model.
* jsontest.json
  + This .json file serves to provide metadata to get\_data.py. At the time of this report, it serves its current purpose. However, it should absolutely be modified by any user who uses the rest of this project. The basic structure need not change, though.
* proces\_sql\_data.py
  + A user would use this Python script to extract data from the tuning database for use by the machine learning model. (The data as it is stored in the tuning database cannot be fed directly to modeling.py.) This file makes it possible for the tuning database to store lots of raw solution data and makes it easy to grab relevant data from the tuning database for use by a machine learning model.

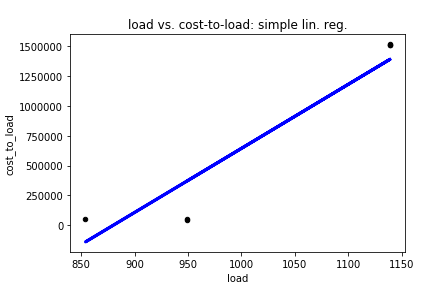
**In-Progress**

At the time of this report, the deliverables discussed in this section are considered in-progress.

* TuningDB.db
  + While this SQLite3 database is currently ready to use and holds useful data in it, the goal is that it will serve as an ever-growing source of data that can better inform the predictive model. For that reason, it is listed as in-progress. Hopefully, it will be “in progress” (constantly updated and added to) if meaningful data is being generated by PLEXOS, gathered historically, and the made available by the industry.
* modeling.py
  + This Python script currently serves as a catch-all for modeling and generating results. At the time of this report, two machine learning models are set up for use: neural network and linear regression. The script predominantly uses TensorFlow (via Keras) to create, train, and use machine learning models. There are two options for scaling data prior to training a model: min-max scaling and standard scaling, after which the data is split into training and testing sets. A function called “run\_model” trains a neural network and a linear regression model and validates results against the test set. A GridSearch function provided by SciKitLearn determines the optimal hyperparameters for each model.

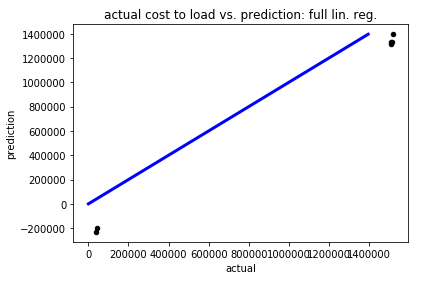
The current models are set up to predict “cost-to-load” based on system load and the fuel prices (Fuel A & Fuel B) for two generators (one y-variable, three x-variables). There are only 27 samples in the dataset, which is generated by PLEXOS Test.xml and found in the folder entitled “PLEXOS Data Solution Files”. Cost-to-load values range from $47,450,000 to $1,518,867,000; 33% of those values fall within the range $1,504,150,000 to $1,518,867,000, and the remaining 66% within $47,450,000 to $51,246,000. While the prices of fuels A and B were not at all strongly correlated with the cost-to-load, load and cost-to-load share a correlation coefficient of 0.95, indicating that they are very strongly related.

Before the data is fed to any model, it is split into a test set and a train set. This allows for cross-validation of results. Then, the training set is scaled. This can speed up calculations in larger datasets, but most importantly it makes the relative importance of each variable the same from the perspective of the model. The most effective scaling seen thus far on this data set is a min-max scaling. Three models and their results are discussed below.

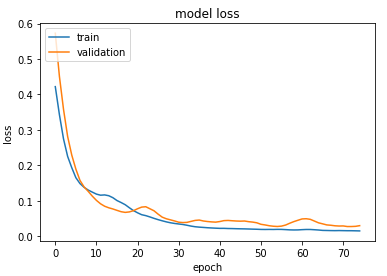
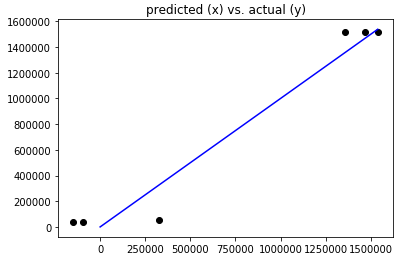
1. Linear regression: cost-to-load on load

The blue line in the graph to the right shows the linear regression’s predictions. The black dots (of which there are six, but several overlap) show actual data points generated by PLEXOS Test.xml. The variance score for this model was .93; a perfect variance score would be 1.

1. Linear regression: cost-to-load on load, Fuel A price, and Fuel B price

This model does the same as the model in (1), with the addition of two explanatory variables. The variance score for this model is around .90, which is slightly lower than the simple linear regression in (1). With more data, this could be much better and make more use of a relationship between cost-to-load and Fuel A and Fuel B prices, assuming a relationship does exist. The graph shown here shows how close the predictions are (black dots) to the correct amounts (would appear on blue line). They appear close, but “close” is relative as the cost-to-load is in thousands of dollars.

1. Neural network

The neural network results are not quite as strong as the linear regression results in this case. A neural network for this dataset tends to overcomplicate what is happening; however, with more data, the neural network may prove more useful. The predicted costs-to-load aren’t that far off of the actual values, relatively, but they are indeed wrong: notice some are predicted to be negative (see second graph). We do find that with more iterations over the data and more training passes, the error decreases (see first graph), suggesting that again, a larger dataset could achieve better results.

In summary, a simple linear regression with one explanatory variable did well enough to outperform the other two more complicated models in this 27-scenario case. However, more accurate results could likely be obtained from a larger dataset, even with more variation. More importantly, the model structure to test this theory is already in place.

**Future**

While the above discussed results have been achieved, the project would benefit from some more work on this portion before it is considered complete. At the very least, modeling.py needs some simple restructuring, and some of the data prepping work should be moved to another file or function which automates the process for other x-y scenarios. An option for the user to specify which data-scaling method (min-max or standard) needs to be added, along with a default handling if none is provided. The current neural network and linear regression models would likely produce more accurate results if they had access to a larger sample size of data. One way to achieve this is to add a stochastic element to the current scenarios found in PLEXOS Test.xml. The type of model (neural network, linear regression, or otherwise) should be specified by the .json file and read into modeling.py. Other models, including classification vs. regression models, should be explored to see if better results can be achieved by something other than a neural net or linear regression model. As models are compared, there needs to be a way to save the results of which is best, along with the optimal hyperparameters for the best model. That way, when new data comes in, a “best” model for that scenario will be ready to use without much extra training. Finally, providing prediction intervals to any of the produced point estimates would make the results of any model more useful. This is a relatively easy add for a linear regression model, and much less straightforward for a neural network.

PROOFREAD REPORT

SEND AS DOC and PDF to DR BROAD with .zip file

PUSH TO GITHUB