Elile Part 2 Write Up

Sean Lee

June 2024

Abstract

We analyze our changes to the LSTM models created by then Stanford PHD students Adele Kuzmiakova, Gael Colas, and Alex McKeehan, and analyze how the changes affect their performance.

1 Introduction

Adele Kuzmiakova, Gael Colas, and Alex McKeehan worked on a project in 2017 to analyze the solar power output generated by UIUC over a time series. Through data cleaning, visualization, and feature selection, they then created models to help with power output prediction.

Model	**No. Parameters**	**Training Error**	**Test Error**
PCA-based Weighted Regression	5	0.866*10ˆ6	1.077*10ˆ6
Standard Weighted Regression	8	0.802*10ˆ6	1.011*10ˆ6
Generalized Boosting Method (Random Forest)	3	0.856*10ˆ6	1.240*10ˆ6
Neural Networks (without temporal data)	*	1.160*10ˆ6	1.183*10ˆ6
Neural Networks (with temporal data)	*	5.378*10ˆ5	5.439*10ˆ5

Figure 1: The authors' original models with their mean squared errors.

The authors' LSTM model was by far the most accurate model with a test error of $5.439 \cdot 10^5$, but they concluded with suggestions as to what changes they could have made. One of these was noting about the time series being sequential, and instead approaching the model as working with sequential data, which could improve accuracy for the LSTM (the other models do not benefit from this assumption: for example, linear regression assumes the data is independent and identically distributed).

2 Additions to the LSTM

For my project, I decided to modify the train-validation-test split from being shuffled and randomized to being chronological based on date and time. I also modified the proportion for each set from being an 80-10-10 split to a 90-5-5 split. There were bugs and issues that occur within the Grid Search section of the code that went unnoticed by the authors, so I fixed those issues. Due to changes in the packages or bugs in the authors' code, I made an attempt to fix the Adaboost model. I created an MSE (mean squared error) list file that stores and compares the MSE's of the LSTM models that were made, and compared my new LSTM model to the unchanged version. Finally, I considered adding lag features to reinforce previous outputs into the memory.

2.1 Sequential Split

The authors had already split the dataset into an 80-10-10 shuffled split. Therefore, to switch to an sequential split, I merged the datasets together and sorted their data by date and time (accomplished by sorting in the order: year, month, day, hour).

I initially split the dataset to have the first 80% of the data be the training set, the next 10% of the data be the validation set, and the last 10% be the testing set. However, my model overfitted, which gave a validation mean squared error that was about twice my training mean squared error. To compensate for this, I switched to have the first 90% of the data be the training set to allow my model to have more training on summer data, and gave the validation set the next 5% and the testing set the rest 5% of the data. This allowed my model to stop overfitting, with a slightly higher training error but much lower validation and testing error. With this, I sticked to this split for the rest of my model evaluation.

2.2 Bug fixing

2.2.1 Grid Search

Since the code was written in 2017, changes with the packages over time meant that some of the code was no longer functional. Some of the code included bugs. A big part of my work in part 2 was to identify and fix the errors that occurred.

To start off, the grid search initially considered the Keras optimizers: Adam, RMSprop, and Adaboost. However, as of Keras 3.0, the Adaboost is no longer a valid optimizer. Interestingly, it was not a valid optimizer in Keras 2.0, and Keras 1.0 does not have any documentation for what optimizers they contained. To mitigate this, I replaced the Adaboost with the Adadelta optimizer. Another issue was that the code had a typo:

```
if error < minimum_error:
                    error = minimum_error
                    optimal_params = [init_type, epoch, batch, optimizer]
```

The minimum error was set at $2e63$, which meant that if our model's error was less than this bug number, then instead of updating the minimum error to be our model's error, they wouldn't be updated at all. So this code was fixed so that the grid search would work properly.

2.2.2 Adaboost Model

The Adaboost classifer no longer worked at all, so I attempted to make this new model work.

My initial workaround was to implement an Adaboost Regressor with Keras and Scikitlearn:

```
adaboost_base_estimator = KerasRegressor(model = build_model, optimizer="adam", epochs=100,
                                         batch_size = 16, verbose=0)
adaboost = AdaBoostRegressor(estimator=adaboost_base_estimator, learning_rate=0.01)
adaboost.fit(X_train, train_labels)
```

This had no error messages, but the mean squared error was over a billion, which was clearly wrong. I made some other workarounds to try and use Keras and Scikitlearn to work with adaboost regression on the LSTM, but they didn't work.

If I had more time, I would instead consider coding adaboost by hand: it is interesting to see how an ensemble of LSTMs would perform, and if they can reduce the error even more.

2.3 Lag features

I also considered adding lag features, such as a feature containing the previous output from an hour ago, two hours ago, etc, on top of sequential data. This actually worked very well for my LSTM in part 1. In theory, LSTM has a memory preservation property which allows it to store in memory of previous outputs when dealing with the next output, so there wouldn't need to be lag features, but in practice, lag features may improve the accuracy of the model by reinforcing the importance of the previous outputs into the calculations.

3 Results

After storing the mean squared error of each model and storing them into a text file, we can compare each model's accuracy. Note that I'm removing the MSE's on Adaboost Models because they are wrong, so we remove them out of the model considerations.

```
Original LSTM without any changes:
Vanilla LSTM Train MSE 537911.0497106075
Vanilla LSTM Valid MSE 480173.31126397225
Vanilla LSTM Test MSE 537914.4461906437
(Optional) Optimal parameters ['normal', 150, 16, 'rmsprop']
```

```
Grid Search Optimal Train Error 508006.3148485781
Grid Seaerch Optimal Test Error 517839.8971436984
```

So we see that the mean squared error of the original LSTM was about 537,000, which is consistent with the authors' findings.

We then check our LSTM (without lag features) on sequential splits:

```
LSTM on sequential data:
LSTM Train MSE  551077.6690894169
LSTM Valid MSE  377717.14638681407
LSTM Test MSE  338101.65467025136
(Optional) Optimal parameters ['glorot_uniform', 100, 16, 'rmsprop']
Grid Search Optimal Train Error 460980.6485784626
Grid Search Optimal Test Error 343674.47067562304
```

We see that the sequential data consideration resulted in less overfitting, slightly increasing the bias of the model but reducing the variance. The test and validation mean squared error is greatly reduced compared to the original LSTM. Our validation error is about 22% less than the original model, while our test error is about 38% less than the original model.

Finally, for the LSTM including lag features, it unfortunately doesn't increase the accuracy of the model.

```
LSTM Train MSE 548390.0449724594
LSTM Valid MSE 380484.8380700148
LSTM Test MSE 323886.5179273499
(Optional) Optimal parameters ['normal', 50, 16, 'adam'] (Output above is using batch_size = 1)
Grid Search Optimal Train Error 482343.7087653852
Grid Search Optimal Test Error 271932.7353494599
```

Just to test, outside of the grid search, I tried out batch_size = 1 for all three models, and only the last model (LSTM with lag) benefitted from it, while the other models actually suffered with worse mean squared errors. Otherwise, it is about the same as the LSTM without lag features, but with longer training time. Therefore, the lag features weren't as helpful for this problem.

4 Conclusion

The change in the data split from shuffled to sequential resulted in a vast improvement in performance, as the time series' temporal data is sequential, which complements the LSTM. We saw a reduction of mean squared error by about 22% to about 38% for our LSTM model compared to the original. Future work would be to consider implementing an ensemble of LSTM networks.