Time Series Analysis: Weather Forecasting

Yi Lin

April 27, 2021

1 Introduction

This project aims to build up a weather predictive system in Python, predicting the temperature for the next 24 hours in the Los Angeles Department of Water and Power. Since the historical records from Los Angeles are about 5 years which is a time-series dataset, using TensorFlow of time series forecasting would be truly changed the way sequential data. TensorFlow could build different styles of models including Convolutional Neural Network (CNN) and Recurrent Neural Networks (RNN).

In this project, I would start by constructing the baseline model with shifting, which exploits the seasonality of the temperature. Then I would train advanced models including LSTM and GRUs that predict timesteps at once. Based on the performance on either baseline model or advanced models, I would compare and contrast the mean squared error between models, and check whether RNNs models make progress or not.

2 Exploratory Data Analysis

There are two data files. The first data file, **ps6_trainvalid.csv**, includes labeled training and validation data. The second data file, **ps6_test.csv**, contains labeled testing data to test the final version of models. Train_val dataset contains 45013 rows while testing dataset contains 240 rows. Both datasets contain 7 different features including temperature, humidity, pressure, weather, wind_direction, and wind_speed. These were collected every hour, beginning in 2012 and ending in 2017.

2.1 Raw Data Visualization

Since there are not too many features, I use plot feature to visualize the evolution of features over time. Figure 1 is the evolution of train_val dataset while Figure 2 is the evolution of testing dataset. As you can see below, it is reasonable that evolution of train_val dataset is much noisy than that of testing dataset due to its data size with all daily temperatures. If we look carefully into the data points, we could notice that there is only small temperature change between current date and the next date. Take an insightful look, I notice that roughly every 24 hours would a period of temperature.

Figure 1: The Evolution of Train_Val Set

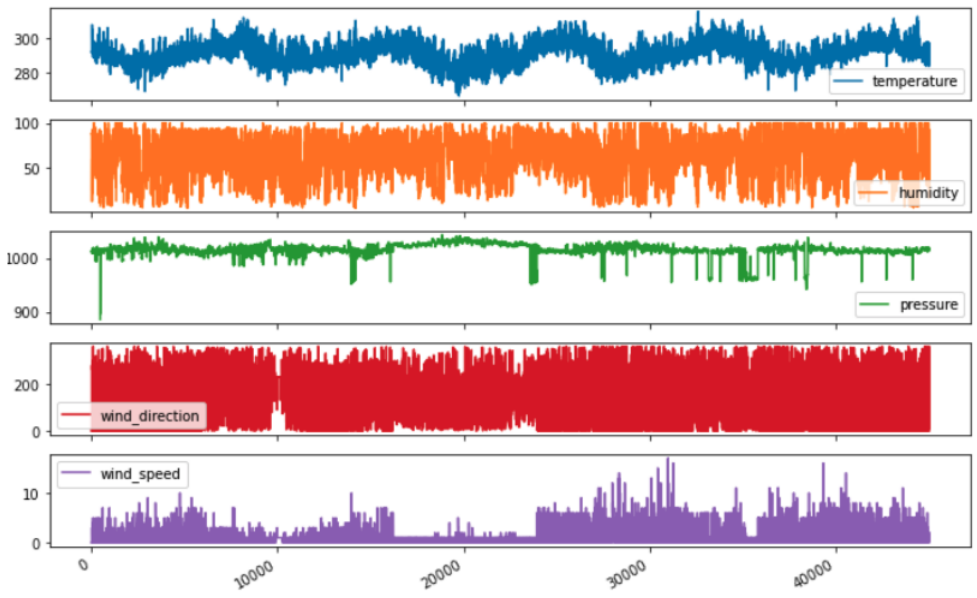
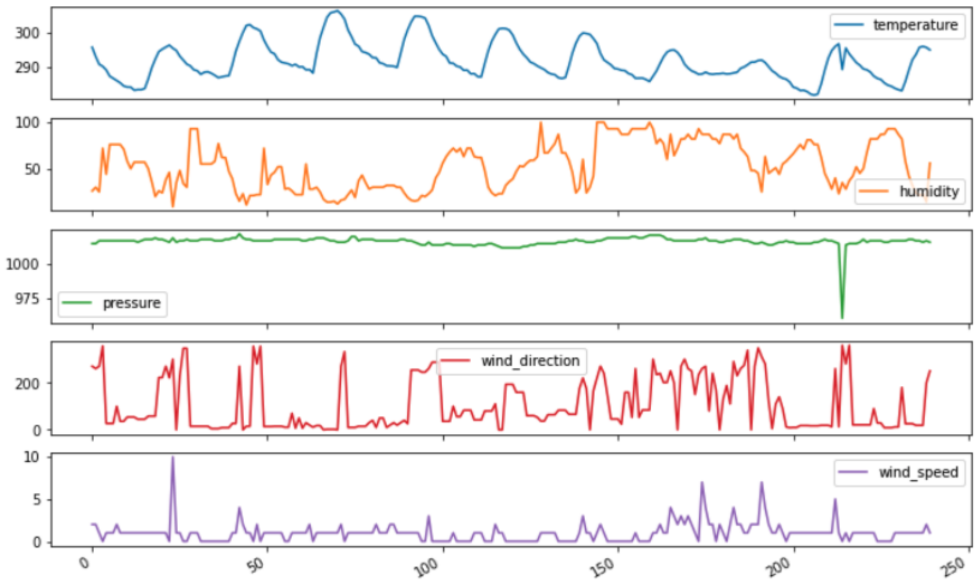


Figure 2: The Evolution of Testing Set

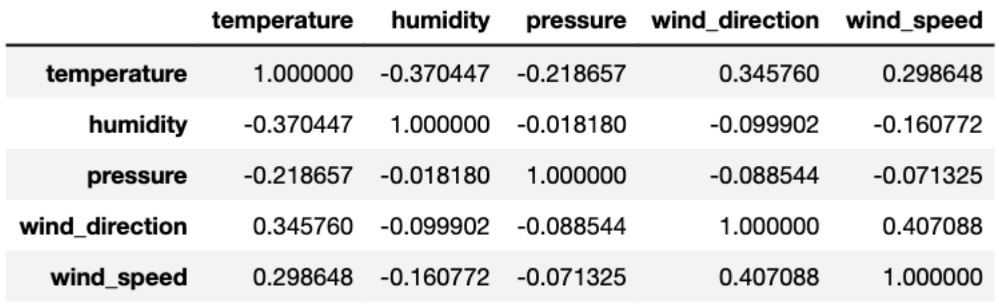


2.2 Correlation Between Raw Numerical Features

To determine how important those different features are, I applied plot correlation matrix to see the relationship between features and target features (temperature). Notice that both humidity (corr = -0.371) and pressure (corr = -0.219) have negative correlation with respect to temperature whereas wind_direction (corr = 0.346) and wind_speed (corr =

0.299) have positive correlation with respect to temperature. Here is Figure 3 of plot correlation matrix:

Figure 3: Plot Correlation Matrix



	temperature	humidity	pressure	wind_direction	wind_speed
temperature	1.000000	-0.370447	-0.218657	0.345760	0.298648
humidity	-0.370447	1.000000	-0.018180	-0.099902	-0.160772
pressure	-0.218657	-0.018180	1.000000	-0.088544	-0.071325
wind_direction	0.345760	-0.099902	-0.088544	1.000000	0.407088
wind_speed	0.298648	-0.160772	-0.071325	0.407088	1.000000

3 Data Cleaning

I checked the statistics of the datasets and did not see any unreasonable statistics number crossing each feature. There are no obvious outlier and erroneous such as negative values of pressure, humidity or temperature. But there are some missing values in the columns.

To replace and interpolate missing values, I first converted the read date to datetime format and set 'datetime' as index of the dataframe. The 'datetime' format as YYYY-MM-DD H:M:S. Then, I check the number of missing value in both datasets. Notice that there is no missing values in testing set. There are missing values in train_val set, including 3 temperature, 152 humidity, 252 pressure, 1 weather, 1 wind_direction, 1 wind_speed.

3.1 Replacing Missing Value with Backfill

I applied backward-filling to fill null values by calling *fillna()* method on the read-value column (weather, wind_direction, wind_speed).

3.2 Interpolate Missing Value

I applied linear method to interpolate null value by calling *interpolate()* method on the read-value column (humidity, pressure).

4 Feature Engineering

4.1 Encoding 'weather'

Since there is one categorical column 'weather', I applied *pd.get_dummies()* to convert categorical variables into dummy/indicator variables. Then I dictionary mapping columns names to prefixes. After encoding 'weather', it generates additional 26 columns in the train_val set while it creates additional 11 columns in the testing set. By using *pd.concat()* to merging the cleaned data, train_val set contains 45013 rows * 29 columns, and testing set contains 240 rows * 14 columns.

4.2 Normalized Dataset

To normalize the dataset, I applied *MinMaxScaler* from *sklearn.preprocessing* to standardized both datasets. I used s*caler.fit_ transform()* to train_val set, and used *scaler.transform()* to testing set.

4.3 Data Windowing

Smoothing is technique applied to time series, removing noise and exposing the signal of underlying causal processes between time steps [1]. Using smoothing function will automatically group observations into a window. In this case, I specify the window size with 24 hours in terms of 1 day, and by default, a trailing window is created. Once the window is created, we can take the window, and this is our transformed dataset. The transforming the dataset into simply moving with window size of 1 days, chosen arbitrarily [1]. Xtrain contains 44989 time series with shape of [44989, 24, 1], while Xtest contains 216 time series with shape of [216, 24, 1]. Since we want to forecast a single value for each series, the targets are columns vectors such that Ytrain has a shape of [44989, 1].

5 Baseline Model

Because the change of temperature is not obvious between any 1-2 days, it is reasonable to generate a basic model in which it uses the current temperature as prediction for the next few days [1]. Therefore, I would yield the prediction of weather based on the assumption: the temperature today depends on the temperature 10 days ago, the temperature yesterday depends on the 10 days before yesterday.

To validate how well our model is, I would look at the Mean Squared Error (MSE) among the actual and predicted temperature. Notice that the MSE of baseline model is small (MSE =5.109). This indicates that baseline model to predict the upcoming day's temperature with the average error of 5.5 degrees Kelvin, which is acceptable.

6 Weather Forecast using RNN Model

As we know, the time series forecasting is the use of models to predict future values based on the previously observed values. In the RNN, we store the output activations from more than one of layers of the neural network. These are usually as hidden later activations. I applied both LSTM and GRU cells in this project. Since LSTM and GRU cells enable to tackle much longer sequence than simple RNNs, these cells are one of the majority reasons behind the success of RNNs.

6.1 LSTM

The *Long-Short-Term Memory* (LSTM) cell was generated in 1997, and improve over the years by several researchers. According to the textbook, the author mentions 'If you consider the LSTM cell as a black box, it can be used very much like a basic cell, except it will perform much better; training will converge faster, and it will detect long-term dependencies in the data [2].'

In the LSTM model turning into a sequence-to-sequence model, it is necessary for us to set *return_ sequences = True* in all recurrent layers (except for the last one). I first applied three Keras recurrent layers, then last layer is Dense layer. The first recurrent layer has batch size of 32, activation of 'relu', return_sequence = True, and input dimension of (24,1). The second recurrent layer is quite similar with the first layer but without input dimension. The third recurrent layer has batch size of 32, activation of 'sigmoid', and return_sequence = False. Finally, it runs the Dense layer with 1 unit.

To compile the LSTM model, I applied optimizer with 'adam'. Then I used Adam optimizer to keep track of accuracy and loss during the training. Then the model is trained for batch size of 32, epochs 20 through the training set after data cleaning and normalize. *model.fit(Xtrain,Ytrain, epochs=20, batch_ size = 32, callbacks=[early_ stop]).*

Once the LSTM model is done with the fit, I draw a plot about Model Loss in Figure 4. Then I use *mean_ squared_ error()* function to evaluate the testing set and calculate the MSE of LSTM model. Notice that the MSE of LSTM is 2.366 which is much better than the baseline model (MSE = 5.109). In the Figure 5, I draw a plot of temperature prediction based on the testing set. Notice that the blue line is actual data point, and red line is predicted data point.

Figure 4: LSTM Model Loss

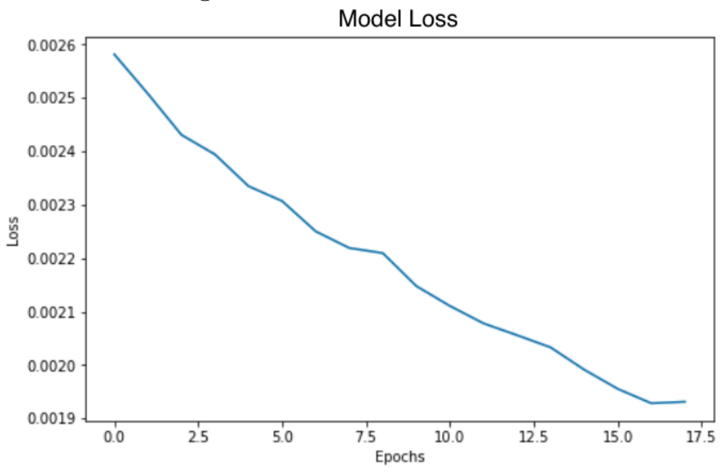
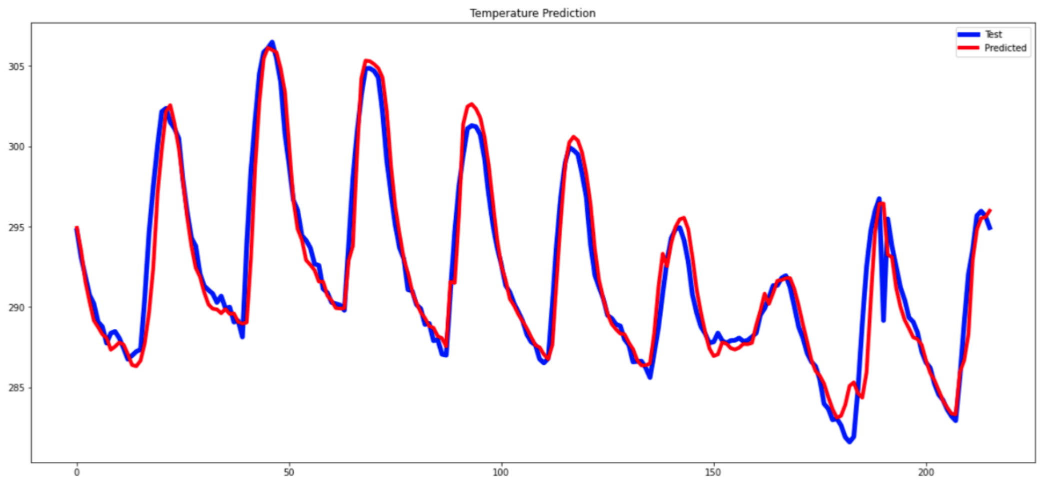


Figure 5: LSTM: Tempreature Prediction



6.2 GRU

The *Gated Recurrent Unit* (GRU) cell was proposed in 2014, which was introduced by the Encoder-Decoder network. According to the textbook, the write mentions 'the GRU cell is a simplified version of the LSTM cell, and it seems to perform just as well (which explains its growing popularity) [2].'

Similar with LSTM model, I first applied three *Keras* recurrent layers, then last layer is Dense layer. The first recurrent layer has batch size of 32, activation of 'relu', return_sequence = True, and input dimension of (24,1). The second recurrent layer is quite similar with the first layer but without input dimension. The third recurrent layer has batch size of 32, activation of 'sigmoid', and return_sequence = False. Finally, it runs the Dense layer with 1 unit.

To compile the GRU model, I applied optimizer with 'adam'. Then I applied Adam optimizer to keep track of both loss and accuracy during the training. Then, almost similar with LSTM model, the model is trained for batch size of 32, epochs 20 through the training set after data cleaning and normalize. *model.fit(Xtrain,Ytrain, epochs=20, batch_ size = 32, callbacks=[early_ stop]).*

Once the GRU model is done with the fit, I draw a plot about Model Loss in Figure 6. Then I use *mean_ squared_ error()* function to evaluate the testing set and calculate the MSE of LSTM model. Notice that the MSE of LSTM is 1.918 which is much better than the both baseline model (MSE = 5.109) and LSTM model (MSE =2.366). In the Figure 7, I draw a plot of temperature prediction based on the testing set. Notice that the blue line is actual data point, and red line is predicted data point.

Figure 6: GRU Model Loss

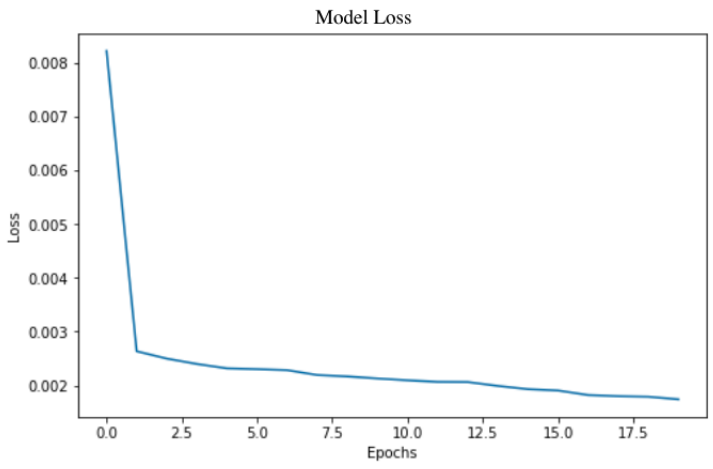
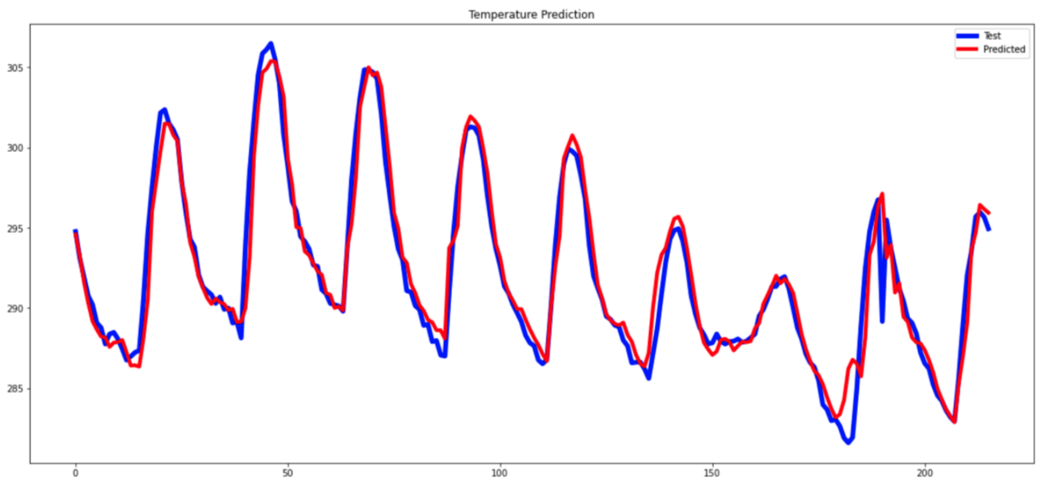


Figure 7: GRU: Temperature Loss



7 Conclusion

In this project, I first did data preprocessing and data cleaning, and then start building the baseline model with shifting to explore the seasonality of the temperature. Then I train recurrent advances models (LSTM and GRU) to predict timesteps with 24 as one day at once. Among the performance of these models, the GRU model yields the best performance with the smallest MSE at 1.918. If an MSE of one model is 0, it means that the estimator is predicting observations of the parameter with perfect accuracy, which would be an ideal scenario but it not typically possible [1]. In addition, the second better performance model is the LSTM model with MSE at 2.366. It is obvious to show that RNNs models did make progress. Since the RNNs model is designed to avoid the long-term dependency issue, it is reasonable that the RNNs model does yield relatively better performance in time series analysis and weather forecasting.

References

[1] Lai, Khoa. “Time Series Analysis and Weather Forecast in Python.” *Medium*, Medium, 4 Jan. 2021

[2] GeÌron, Aureìlien. *Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 2nd ed., O’Reilly, 2019.