Model Documentation

This document provides background information and describes methods used in the Power Laws Forecasting Challenge.

1. Who are you (mini-bio) and what do you do professionally?

I am a Master's student in mechanical engineering with a research focus in applied data science at Case Western Reserve University. I have an undergraduate degree in mechanical engineering and am almost entirely self-taught (using Udacity, Coursera, and books) when it comes to data science. I work on a research project called Energy Diagnostic Investigator for Efficiency Savings (EDIFES) that uses large-scale data analytics to improve building energy efficiency. My long term goals are to graduate with my Masters next year (May 2019) and work full time as a data scientist in the clean technology sector, preferably for a start-up working to enable the transition to a renewable energy economy.

2. High level summary of your approach: what did you do and why?

My approach treats the problem as a standard supervised regression task. Given a set of features – the time and weather information – we want to build a model that can predict the continuous target, energy consumption. The model is trained on the past historical energy consumption using the features and the target and then can be used to make predictions for future dates where only the features are known.

I first engineered the original training and testing data to extract as much information as possible. This involves first breaking out the timestamp into minutes, hours, weekday, day of the year, day of the month, month, and year. The time variables are then converted to cyclical features to preserve the proper relationships between times (such as hour 0 being closer to hour 23 than to hour 5). Then I joined the weather data to the building energy-use data because the temperature can significantly influence the energy consumption and including this information helps the model learn to predict energy use. I rounded the index of the weather data to the nearest fifteen minutes and then joined the weather data to the building energy-use data based on the timestamp. The final aspect of feature engineering was finding the days off for each building using the metadata. This is crucial because the energy use differs significantly between working and non-working days. The same engineering that was applied to the training data was applied to the testing data because the model needs to see the same features at test time as during training. I did not include the holidays in the data because I found the information was sparse.

The final machine learning model I built is an ensemble of two ensemble methods (a meta-ensemble I guess you could call it), the random forest and the extra trees algorithms. The final model is made up of six random forest regressors and six extra trees regressors, each of which uses a different number of decision trees (from 50 to 175 spaced by 25). (All of the algorithms are implemented using the Scikit-learn Python library). The final predictions are made by averaging the results of these 12 individual estimators.

During training, I train the complete model on a single building at a time, which I found to work best rather than trying to make one model that can be trained on all buildings at once. I first select the engineered training and testing data for the building, making sure to use only **past data for training**. Then, I impute the missing temperature values in the training and testing data using median imputation. I impute the missing energy values only in the training data using median imputation (this is not done for the testing data because the energy value is what we want to predict). I convert the timestamp to a numeric value so the machine learning models can understand this feature, and make sure to drop the energy use value from the training data. The random forest and extra trees models do not require features

to be scaled, but for other machine learning models, scaling the features between 0 and 1 is a best practice so this step is implemented before the models are trained.

For each building, I then train the 12 separate models on the training data and make predictions on the testing data. The final predictions are made by averaging the predictions from the 12 individual models. Once predictions have been made for all buildings, they are saved to the specified file.

3. Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

Cyclical Variable Transformation

Time variables are cyclical features! What this means is that to preserve the proper relationships between time values, we need to transform them into sines and cosines representations. For example, during the day, hour 0 is closer to hour 23 than it is to hour 10. However, using a simple 0-24 coding scheme for hours will place 0 closer to 5. To address this, we convert the time to a representation using sines and cosines. With time of the day, this means taking the sine or cosine of the time multiplied by 2π and divided by 24. A great introduction to the topic is provided in this article. Cyclical variable transformation is implemented in `data_format.py`.

```
# Cyclical variable transformations

# time has period of 24
df['time_sin'] = np.sin(2 * np.pi * df['time'] / 24)
df['time_cos'] = np.cos(2 * np.pi * df['time'] / 24)

# wday has period of 6
df['wday_sin'] = np.sin(2 * np.pi * df['wday'] / 6)
df['wday_cos'] = np.cos(2 * np.pi * df['wday'] / 6)

# yday has period of 365
df['yday_sin'] = np.sin(2 * np.pi * df['yday'] / 365)
df['yday_cos'] = np.cos(2 * np.pi * df['yday'] / 365)

# month has period of 12
df['month_sin'] = np.sin(2 * np.pi * df['month'] / 12)
df['month_cos'] = np.cos(2 * np.pi * df['month'] / 12)
```

Median Imputation of Missing Temperature and Energy Use

A classic problem in machine learning is dealing with missing data. While there are many ways to fill in values that are missing, median imputation is one of the simplest and also fairly effective. This entails filling in any missing value with the median value in the column. For example, if we are missing 10 temperature measurements for one building and the median temperature measurement was 20.5, then all the missing temperatures will be filled in as 20.5. If all the temperatures were missing from the training or testing data, then the temperature column was not used because it does not contain any information. This process was carried out for temperature on the training and testing data, and then for the energy only on the training data. One important note is that before we perform imputation, we must first select only the **past training data**, otherwise we would be incorporating information from the future into our training

data. Other methods of imputation were tried, but the simplest method turned out to work the best! Here's a good scikit-learn reference on imputation. Median imputation was carried out in `predict.py`

```
# Only use past training data

train_df = train_df[train_df['Timestamp'] < test_df['Timestamp'].min()]

# If all training temperatures are missing, drop temperatures from both
training and testing

if (np.all(np.isnan(train_df['Temperature']))) or
(np.all(np.isnan(test_df['Temperature']))):

            train_df = train_df.drop(labels = 'Temperature', axis=1)

            test_df = test_df.drop(labels= 'Temperature', axis=1)


# Otherwise impute the missing temperatures

else:

      temp_median_imputer = Imputer(missing_values='NaN', strategy='median',
axis = 0)

      temp_median_imputer.fit(train_df[['Temperature']])

      train_df['Temperature'] =
temp_median_imputer.transform(train_df[['Temperature']])

      test_df['Temperature'] =
temp_median_imputer.transform(test_df[['Temperature']])


# Impute the missing energy values

value_median_imputer = Imputer(missing_values='NaN', strategy='median',
axis=0)

value_median_imputer.fit(train_df[['Value']])

train_df['Value'] = value_median_imputer.transform(train_df[['Value']])
```

Ensembling Many Ensemble Models

The final model is an ensemble of 12 models using two different ensembling algorithms: the random forest and the extra trees regressor. For each algorithm, the code creates 6 different models, using a different number of trees for each model (from 50 to 175 spaced by 25). The random forest and extra trees models are so effective because they average the predictions from many weak learners, and we can extend this idea further by then taking the average of multiple random forest and extra trees models. This works because it reduces the variance that can arise when we are only using one decision tree (or one single random forest). Averaging the predictions from many different models can also serve as a regularization technique to prevent overfitting by trying to average out the bias from each model. The original article on

Random Forests by Leo Breiman is a nice read on the subject and for information on ensembling, here is a solid article for the basics. The ensembling was carried out in `predict.py`

```
# List of trees to use in the random forest and extra trees model
trees_list = list(range(50, 176, 25))

# Initialize list of predictions for site
_predictions = np.array([0. for _ in range(len(test_x))])


# Iterate through the number of trees
for tree in trees_list:

	# Create a random forest and extra trees model with the number
of trees
	model1 = RandomForestRegressor(n_estimators=tree, n_jobs=-1)
	model2 = ExtraTreesRegressor(n_estimators=tree, n_jobs=-1)

	# Fitting the model
	model1.fit(train_x, train_y)
	model2.fit(train_x, train_y)

	# Make predictions with each model
	_predictions += np.array(model1.predict(test_x))
	_predictions += np.array(model2.predict(test_x))

# Average the predictions
predictions = _predictions / (len(trees_list) * 2)
```

4. **What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?**

There are far too many attempts to elaborate, but here is a brief listing:

- Other methods of imputation: mean, mode, k-nearest neighbors
- Training on all the buildings at once rather than a single building
- Training on a per-forecast id basis rather than on a single building
- Using Auto-Regressive Integrated Moving Average (ARIMA) models to predict energy use as a univariate time-series problem
- Trying out many different models: deep neural networks (using Keras), support vector machines, gradient boosting algorithms, generalized linear models
- Engineering different features and using holiday information (which did not help)!

5. **Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?**

Yes, my workflow involved a dozen different Jupyter Notebooks that were used for data exploration and model development. The final submission includes only the final two scripts that can be used to go from the raw data to the final predictions. I made plenty of exploratory plots using the matplotlib and seaborn library to try and gain insights into the data and tried out many different methods in Jupyter Notebooks. If there is interest, I would be willing to document and display the Jupyter Notebooks to show my progress.

My general workflow is to do all data exploration and initial modeling in Jupyter Notebooks and then move on to Python scripts when I am only making small changes to the machine learning models. Scripts are easier to run on the high-powered cloud computing resources I use (Google Cloud and Amazon Web Services), and after the data has been cleaned, engineering, and saved in Jupyter Notebooks and the basic model has been developed, I move to scripts. This lets me make only minor changes to the model, such as adjusting the number of iterations used for training a deep neural network, run the script, and then judge the effect based on the performance metric. My workflow for this challenge followed exactly the same pattern, with a dozen Jupyter Notebooks used for data exploration and initial modeling and the final two scripts (although I mostly changed `predict.py)` used to adjust the model and run it on high-powered cloud computers.

6. **How did you evaluate performance of the model other than the provided metric, if at all?**

I used cross-validation to improve my models by taking a random subset of 100 buildings to serve as a testing set and calculating the root mean-squared error for predictions on these buildings. This generally underestimated the final error on the actual test sets. I was never able to get a working implementation of the provided metric because I found the documentation lacking. While the cross-validation was helpful, in the end, I relied more on the score from the submissions to gauge the performance of models.

7. **Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?**

Both the random forest regressor and the extra trees regressor are highly non-deterministic algorithms. Therefore, predictions made on the test set will vary significantly across runs. This can be addressed using a random state in the model building, but I did not use a random state for any of my submissions because then it seems like you can get carried away trying to find the “best” random state.

8. **Do you have any useful charts, graphs, or visualizations from the process?**

I have quite a few visualizations in the development Jupyter Notebooks that I can retrieve if there is interest. The most useful charts are those that show the relationships between energy and the time and weather features because these allowed me to focus my feature engineering on extracting the most relevant variables. In addition, the feature importances returned by the random forest and extra trees regressors can be used to determine the most relevant features for predicting energy use. These exploratory and diagnostic plots played a large role in the project by informing feature engineering decisions.

9. **If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?**

If I had more time to work on this problem, I would focus on acquiring more data. I tried nearly every model possible, including deep neural networks, support vector machines, gradient boosted ensembles, and generalized linear models and had the most success with the final implementation of random forests and extra trees. I do not think further modeling and hyperparameter tuning will improve the predictions much beyond the final results. This is often the case in machine learning: getting more data will significantly improve performance while tuning models has a smaller effect with a higher development cost. Here is a great article titled "The Unreasonable Effectiveness of Data" which demonstrates the choice of model matters less and less as the amount of training data increases.

In terms of the actual data to acquire, I would like to have access to expanded weather data. Ideally, the weather information would completely cover the time periods involved for each building and would have more variables such as precipitation, global horizontal irradiance, relative humidity, etc. It's no guarantee these would improve the model, but generally, the weather is a large influence on energy consumption and more weather information could not hurt performance. Moreover, simply increasing the length of the training data for each building would improve performance. For many of the buildings, there was a relatively limited amount of historical data, and performance would significantly improve with access to several years of past data. If it were possible to get other information about buildings, such as the occupancy schedule or use-types of the building, that could also improve performance.

Overall, the largest return on investment will come from acquiring more high-quality data rather than from tuning machine learning algorithms.

Thanks for a great competition!