Configuration Manual

MSc Research Project
Data Analytics

Gabriel Dada Ibukun
Student ID: x18176585

School of Computing
National College of Ireland

Supervisor: Dr. Vladimir Milosavljevic
Student Name: Gabriel Dada
Student ID: x18176585
Programme: Data Analytics
Year: 2019
Module: MSc Research Project
Supervisor: Dr. Vladimir Milosavljevic
Submission Due Date: 12th December 2019
Project Title: Configuration Manual

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author’s written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: 
Date: 

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).

Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).

You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only
Signature: 
Date: 
Penalty Applied (if applicable):
Configuration Manual
Gabriel Dada (x18176585)

1 Introduction
This configuration manual provides detailed documentation of the implementation of I.T solution deployed as part of the research thesis in Electric Load Forecasts using Machine Learning and Distributed Systems. The scope covers all steps taken to for solution deployment. The systems configuration requirements are as follows:

- Processor: intel core i5 1.8Ghz DDR3
- RAM: 8GB
- System: x64 processor

2 Integrated Development Environment
The project implementation was deployed in the Anaconda 2019.10 for mac operating system (with 64 bit graphic installer) environment. Python 3.7 accompanies it as both can be downloaded from here. Having installed Anaconda, Jupyter notebook was used for the data for data pre-processing, transformation, feature engineering and modeling.

3 Datasets
Datasets used for this project were downloaded as csv files in two categories namely electric load data and weather data. The load data was originally sourced from PJM open source repository online here. The historical hourly weather datasets were sourced directly from Kaggle containing weather measures of temperature, pressure, humidity, wind direction, and wind speed for 30 US cities here.

4 Assessing the datasets
The datasets were first loaded into R studio for preliminary checks after which all 6 datasets were loaded to the Jupyter python environment. First, all necessary libraries required for our analysis were loaded into python (even though Jupyter notebook has some of these libraries pre-installed). This is shown here:

```
In [1]: import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as mplot
import xgboost
from sklearn.ensemble import AdaBoostRegressor, BaggingRegressor, ExtraTreesRegressor
from xgboost import plot_importance, plot_tree
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
import warnings
warnings.filterwarnings('ignore')
mplot.style.use('ggplot')
```
After which the datasets are loaded accordingly to Jupyter notebook.

```python
In [3]: %time
#Load Power Datasets
power_dom = pd.read_csv('DOK_hourly.csv', parse_dates=[0], squeeze=True, index_col=[0])
power_dom = power_dom.loc[~power_dom.index.duplicated(keep='first')].sort_index().dropna()

CPU times: user 38.3 s, sys: 1.78 s, total: 40.1 s
Wall time: 2min 23s

In [5]: %time
#Load Weather Datasets
#Humidity
humidity = pd.read_csv('humidity.csv', parse_dates=[0], squeeze=True, index_col=[0])
humidity = humidity.loc[~humidity.index.duplicated(keep='first')].sort_index().dropna()

# Pressure
pressure = pd.read_csv('pressure.csv', parse_dates=[0], squeeze=True, index_col=[0])
pressure = pressure.loc[~pressure.index.duplicated(keep='first')].sort_index().dropna()

# Temperature
temperature = pd.read_csv('temperature.csv', parse_dates=[0], squeeze=True, index_col=[0])
temperature = temperature.loc[~temperature.index.duplicated(keep='first')].sort_index().dropna()

#Wind Direction
wind_direction = pd.read_csv('wind_direction.csv', parse_dates=[0], squeeze=True, index_col=[0])
wind_direction = wind_direction.loc[~wind_direction.index.duplicated(keep='first')].sort_index().dropna()

#Wind Speed
wind_speed = pd.read_csv('wind_speed.csv', parse_dates=[0], squeeze=True, index_col=[0])
wind_speed = wind_speed.loc[~wind_speed.index.duplicated(keep='first')].sort_index().dropna()

CPU times: user 38. s, sys: 898 ms, total: 38.9 s
Wall time: 50.9 s
```

5  Concatenation to create final project dataset

Since our analysis hinges on a single dataset that will be use electric load consumption as the dependent, and weather features such as temperature, pressure, etc as independent variables. The task will be to concatenate the various times series joining them by the date-time column common to all 6 csv files. Using the pandas library, first we deal with those of weather:

```python
In [12]: %time
#Concatenate weather Data
weather_data = pd.concat([temperature, humidity, pressure, wind_direction, wind_speed], axis=1).sort_index()

CPU times: user 27.3 ms, sys: 19 ms, total: 45.2 ms
Wall time: 165 ms
```

And then add power:
The final concatenated output looks like this:

```
In [15]:
project_data = pd.concat([power_data.loc[:9], weather_data.index[-1]], weather_data, axis=1).sort_index()

CPU times: user 57 ms, sys: 29.2 ms, total: 86.2 ms
Wall time: 174 ms
```

<table>
<thead>
<tr>
<th>POWER_MW</th>
<th>temperature</th>
<th>humidity</th>
<th>pressure</th>
<th>wind_direction</th>
<th>wind_speed</th>
<th>pressure_log</th>
</tr>
</thead>
<tbody>
<tr>
<td>987.0</td>
<td>288.650000</td>
<td>87.0</td>
<td>1012.0</td>
<td>70.0</td>
<td>4.0</td>
<td>6.919684</td>
</tr>
<tr>
<td>987.0</td>
<td>288.650172</td>
<td>87.0</td>
<td>1012.0</td>
<td>70.0</td>
<td>4.0</td>
<td>6.919684</td>
</tr>
<tr>
<td>987.0</td>
<td>288.650682</td>
<td>87.0</td>
<td>1012.0</td>
<td>71.0</td>
<td>4.0</td>
<td>6.919684</td>
</tr>
<tr>
<td>987.0</td>
<td>288.650911</td>
<td>87.0</td>
<td>1012.0</td>
<td>71.0</td>
<td>4.0</td>
<td>6.919684</td>
</tr>
<tr>
<td>987.0</td>
<td>288.651401</td>
<td>87.0</td>
<td>1012.0</td>
<td>72.0</td>
<td>4.0</td>
<td>6.919684</td>
</tr>
</tbody>
</table>

6 Feature Engineering

To further prepare the time series data for modeling, date-time features were expanded, also lag features created with this block of code. First date time features:

```
# Time Series Feature
project_final = (project_data.assign( day_of_week = project_data.index.dayofweek
    ,year = project_data.index.year
    ,month = project_data.index.month
    ,day = project_data.index.day
    ,day_of_year = project_data.index.dayofyear
    ,week = project_data.index.week
    ,week_day = project_data.index.weekday
    ,quarter = project_data.index.quarter
    ,hour = project_data.index.hour
    ,hour_x = np.sin(2.*np.pi/project_data.index.hour/24.)
    ,hour_y = np.cos(2.*np.pi/project_data.index.hour/24.)
    ,day_of_year_x = np.sin(2.*np.pi/project_data.index.dayofyear/365.)
    ,day_of_year_y = np.cos(2.*np.pi/project_data.index.dayofyear/365.)
    )
```

And then lag features with the below configuration:
This process increased the number of features to 54 in total. The output file is shown:

```

7 Feature Selection

Feature selection was achieved using ranking the contribution of all features in our model using the F-score. A plot of feature importance from an initial Xgboost regression model was used as a basis. The input blocks of codes and out are outline below.
```
```python
# Define the plot functions

def plot_prediction(actual, prediction, start_date, end_date, title, prediction_label):
    mplot.figure(figsize=(10, 5))
    mplot.title(title)
    mplot.plot(y_test.index, y_test, label='Actual')
    mplot.plot(y_test.index, prediction, label=prediction_label)
    mplot.ylabel('Power (MW)')
    mplot.xlabel('DateTime')
    mplot.legend()
    mplot.xlim(left=start_date, right=end_date)
    mplot.show()


def subplot_prediction(actual, prediction, prediction_label):
    fig, axes = mplot.subplots(nrows=2, ncols=1, figsize=(10, 12))
    con_df = pd.concat([actual.rename('Actual'), pd.DataFrame(index=actual.index, columns=[prediction_label])]
    axes[0].set_title('Actual vs Prediction - Day ahead')
    axes[0].set_ylabel('Power (MW)')
    axes[0].set_xlabel('DateTime')
    con_df.plot(ax=axes[0])
    axes[0].set_xlim(left=con_df.index[24+1], right=con_df.index[-1])
    axes[1].set_title('Actual vs Prediction - Week ahead')
    axes[1].set_ylabel('Power (MW)')
    axes[1].set_xlabel('DateTime')
    con_df.plot(ax=axes[1])
    axes[1].set_xlim(left=actual.index[24+7], right=actual.index[-1])
    axes[2].set_title('Actual vs Prediction - Month ahead')
    axes[2].set_ylabel('Power (MW)')
    axes[2].set_xlabel('DateTime')
    con_df.plot(ax=axes[2])
    axes[2].set_xlim(left=actual.index[24+30], right=actual.index[-1])
    mplot.tight_layout()
    mplot.show()


mplot.tight_layout()

mplot.show()


def plot_feature_importances(clf, X_train, y_train=None, top_n=10, figsize=(10, 10), print_table=False, title='Feature Importances'):
    feat_imp = pd.DataFrame({'Importance': clf.feature_importances_})
    feat_imp.sort_values(by='Importance', ascending=False, inplace=True)
    feat_imp = feat_imp.iloc[:top_n]
    feat_imp.sort_values(by='Importance', inplace=True)
    feat_imp.set_index('feature', drop=True)
    feat_imp.plot.barh(title=title, figsize=figsize)
    mplot.ylabel('Feature Importance Score')
    mplot.show()

    if print_table:
        from IPython.display import display
        print('Top {} features in descending order of importance'.format(top_n))
        display(feat_imp.loc[:, 'Importance'].sort_values(by='Importance', ascending=False))
        return feat_imp

m [68]: regression = xgboost.XGBRegressor()

m [69]: tscv = TimeSeriesSplit(n_splits=5)
scores = cross_val_score(regression, X.values, y.values, cv=tscv,
                   scoring='explained_variance')
p = print('Accuracy: %.2f +/- %.2f' % (scores.mean(), scores.std()))
p = print(scores)

[00:04:40] WARNING: src/objective/regression_obj.cu:152: reglinear is now deprecated in favor of reg_squarederror.
[00:04:45] WARNING: src/objective/regression_obj.cu:152: reglinear is now deprecated in favor of reg_squarederror.
[00:04:45] WARNING: src/objective/regression_obj.cu:152: reglinear is now deprecated in favor of reg_squarederror.
[00:04:45] WARNING: src/objective/regression_obj.cu:152: reglinear is now deprecated in favor of reg_squarederror.
[00:04:45] WARNING: src/objective/regression_obj.cu:152: reglinear is now deprecated in favor of reg_squarederror.

m [70]: regression.fit(X_train, y_train)
prediction = regression.predict(X_test)

[00:04:40] WARNING: src/objective/regression_obj.cu:152: reglinear is now deprecated in favor of reg_squarederror.

5
8 Modeling

Here, the machine learning models were implemented and necessary evaluation metric obtained. Xgboost, Extra Trees regressor, SARIMA and ARIMA were applied to the different lengths of

8.1 Modeling: XGBoost
```python
# Model validation using k-fold cross validation score
n = 10
scores = cross_val_score(regression, X.values, y.values, cv=n, scoring='explained_variance')
print("Accuracy: \$0.2f \pm \$0.2f\" % (scores.mean(), scores.std()))

# Plot importance of features
plot_importance(regression, height=0.9)
```

---

**Feature importance**

<table>
<thead>
<tr>
<th>Features</th>
<th>F score</th>
</tr>
</thead>
<tbody>
<tr>
<td>temperature</td>
<td>185</td>
</tr>
<tr>
<td>hour</td>
<td>121</td>
</tr>
<tr>
<td>day_of_year</td>
<td>99</td>
</tr>
<tr>
<td>humidity</td>
<td>39</td>
</tr>
<tr>
<td>hour_y</td>
<td>37</td>
</tr>
<tr>
<td>day_of_week</td>
<td>35</td>
</tr>
<tr>
<td>pressure</td>
<td>32</td>
</tr>
<tr>
<td>day_of_year_y</td>
<td>29</td>
</tr>
<tr>
<td>week</td>
<td>24</td>
</tr>
<tr>
<td>hour_x</td>
<td>21</td>
</tr>
<tr>
<td>year</td>
<td>21</td>
</tr>
<tr>
<td>day</td>
<td>10</td>
</tr>
<tr>
<td>wind_speed</td>
<td>9</td>
</tr>
<tr>
<td>wind_direction</td>
<td>6</td>
</tr>
<tr>
<td>month</td>
<td>1</td>
</tr>
</tbody>
</table>

8.2 Modeling: Exratrees Regressor

```python
In [132]:
    X = project_final.drop(columns = ['POWER_MW', 'week_day'])
y = project_final['POWER_MW']

In [133]:
    X.shape

Out[133]:
    (44655, 18)

In [134]:
    y.shape

Out[134]:
    (44655,)

In [135]:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle=False)

In [136]:
    regression = ExtraTreesRegressor()

In [137]:
    %time
    #MODEL VALIDATION USING K-FOLD CROSS VALIDATION SCORE
    tscv = TimeSeriesSplit(n_splits=10)
    scores = cross_val_score(regression, X.values, y.values, cv=tscv
    ,scoring='explained_variance'
    )
    print("Accuracy: 0.2f (+/- \%0.2f)" % (scores.mean(), scores.std()))
    print(scores)

    Accuracy: 0.71 (+/- 0.19)
    [0.21945934 0.53197164 0.71098586 0.79369168 0.82068105 0.85682173
    0.81769279 0.81277603 0.72259419 0.81744909]
    CPU times: user 12.2 s, sys: 572 ms, total: 12.8 s
    Wall time: 14.6 s

In [138]:
    regression.fit(X_train, y_train)
    prediction = regression.predict(X_test)

CPU times: user 1.8 s, sys: 71.4 ms, total: 1.87 s
Wall time: 2.29 s

In [139]:
    %time
    #RMSE
    rmse = np.sqrt(mean_squared_error(y_test, prediction))
    norm_rmse = rmse/np.std(y_test)
    print("RMSE: \%f" % (norm_rmse))

    RMSE: 0.460499
    CPU times: user 2.25 ms, sys: 1.57 ms, total: 3.82 ms
    Wall time: 4.43 ms

In [140]:
    %time
    def mean_absolute_percentage_error(y_true, y_pred):
        y_true, y_pred = np.array(y_true), np.array(y_pred)
        return np.mean(np.abs(y_true - y_pred) / y_true) * 100
    mean_absolute_percentage_error(y_train, prediction)

    CPU times: user 942 µs, sys: 432 µs, total: 1.37 ms
    Wall time: 1.11 ms

Out[140]:
    7.026220323155585
```
8.3 Modeling: SARIMA

First, the time series decompose plot is used to split the time series into its trend, seasonal, and residual elements using this block of codes:

```python
# Using sm.tsa.seasonal_decompose to show trend, seasonality and noise
from pylab import rcParams
cParams = {'figure.figsize': [12, 6]}
decomposition = sm.tsa.seasonal_decompose(y, model='additive')
fig = decomposition.plot()
mplot.show()
```

![Decomposition Plot](image)
Next is to find the Optimum \((p,d,q)(P, D, Q)\) parameters using grid search iteration

The combination of parameters with the lowest AIC score is selected and used for the forecast:
Similar procedure is carried out to grid search for best (p,d,q) parameters for ARIMA as shown below:
Then used for forecasts:

```python
In [ ]; #use it for the forecast
from matplotlib import pyplot
from pylab import rcParams
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_squared_error
from pandas import read_csv
from pandas import datetime
from math import sqrt
import warnings

# model = ARIMA(history, order=(0,1,1))
# output = model_fit.forecast()
# yhat = output[0]
# predictions.append(yhat)
# obs = test[t]
# history.append(obs)

X = y.values
size = int(len(X) * 0.70)
train, test = X[:size], X[size:]
history = [x for x in train]
predictions = list()

for t in range(len(test)):
    model = ARIMA(history, order=(0,0,1))
    model_fit = model_fit.forecast()
    yhat = output[0]
    predictions.append(yhat)
    obs = test[t]
    history.append(obs)
rmse = sqrt(mean_squared_error(test, predictions))
norm_rms = rmse/np.std(test)
print("RMSE: %f" % (norm_rms))

# MAPE
def mean_absolute_percentage_error(y_true, y_pred):
    Y_true, Y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

print("MAPE: %f" % (mean_absolute_percentage_error(test, predictions)))
```

References

[1] Hong T and Shahidehpour, M 2015. Load Forecasting Case Study for the Eastern Interconnection States’ Planning Council (EISPC) in response to the NARUC solicitation NARUC-2014-RFP042–DE0316. University of North Carolina at Charlotte (UNCC) teamed with Illinois Institute of Technology (IIT), ISO-New England, and North Carolina Electric Membership Corporation (NCEMC). The work was supported by the Department of Energy, National Energy Technology Laboratory, under Award Number DE-OE0000316
