Muvaffak Onus

Random Forest and Neural Network on Sales Data

Link to this blog entry for reference: http://52.58.179.173/random-forest-and-neural-network-model-on-sales-data/

Link to GitHub repository containing all the codes: [Forecasting sales of products in Rossmann Stores]

First, we need some cleaning on data such as removing Sundays because stores are not open on Sundays. Also for the sake of simplicity we will work on only 1 store's sales for now.

```
import pandas as pd
import statsmodels.api as sm
import datetime
from pandas.plotting import autocorrelation_plot
import matplotlib.pyplot
df = pd.read_csv("dataset/train.csv", sep=',', parse_dates=[2])

df = df[df['Store'] == 1.0][df['Date']>datetime.date(2013,1,6)].sort_values(by='Date')
df = df[df['DayOfWeek'] != 7]
df.head(10)
```

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
1009405	1	1	2013-01-07	7176	785	1	1	0	1
1008290	1	2	2013-01-08	5580	654	1	1	0	1
1007175	1	3	2013-01-09	5471	626	1	1	0	1
1006060	1	4	2013-01-10	4892	615	1	1	0	1
1004945	1	5	2013-01-11	4881	592	1	1	0	1
1003830	1	6	2013-01-12	4952	646	1	0	0	0
1001600	1	1	2013-01-14	4717	616	1	0	0	0
1000485	1	2	2013-01-15	3900	512	1	0	0	0
999370	1	3	2013-01-16	4008	530	1	0	0	0
998255	1	4	2013-01-17	4044	503	1	0	0	0

Taking a look at the summary statistics of the data.

```
df.describe()
```

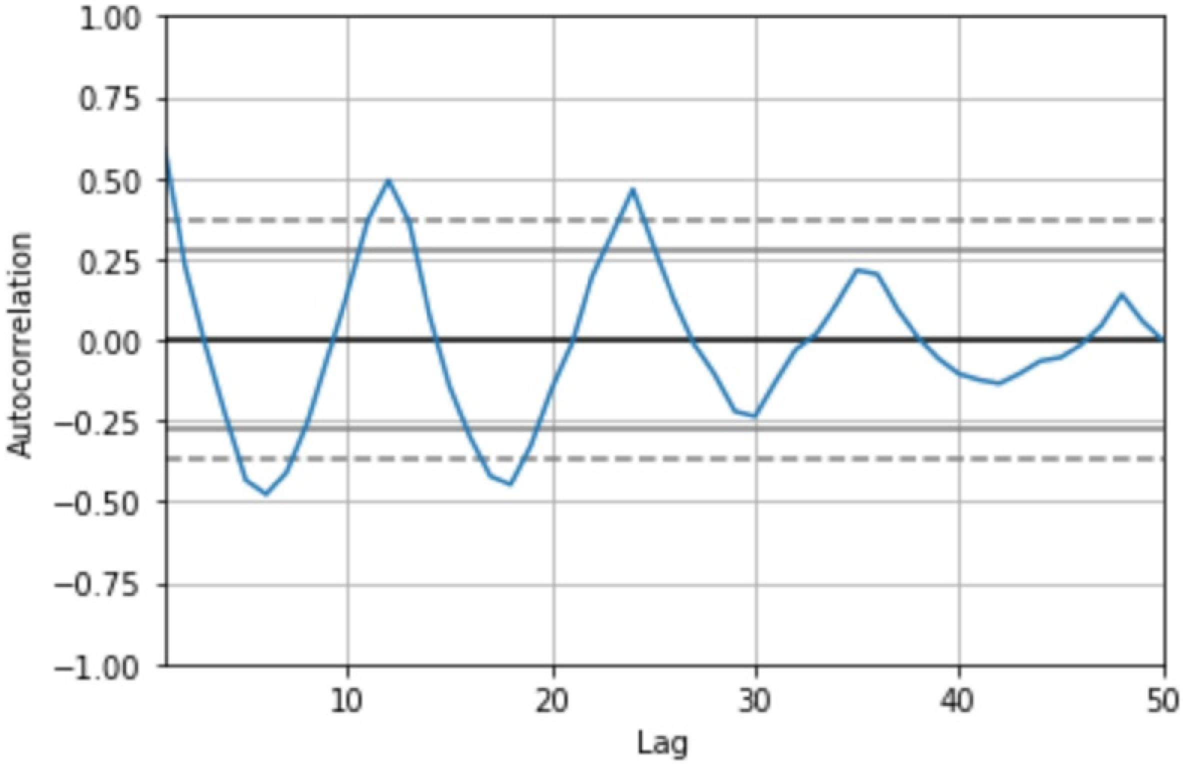
	Store	DayOfWeek	Sales	Customers	Open	Promo	SchoolHoliday
count	803.0	803.000000	803.000000	803.000000	803.000000	803.000000	803.000000
mean	1.0	3.496887	4604.625156	545.483188	0.967621	0.448319	0.209215
std	0.0	1.707670	1305.943349	135.961270	0.177114	0.497632	0.407002
min	1.0	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.0	2.000000	3935.000000	495.000000	1.000000	0.000000	0.000000
50%	1.0	3.000000	4602.000000	546.000000	1.000000	0.000000	0.000000
75%	1.0	5.000000	5327.000000	608.000000	1.000000	1.000000	0.000000
max	1.0	6.000000	9528.000000	1130.000000	1.000000	1.000000	1.000000

Correlation matrix would be beneficial to see the connections between the features that we already have. Use `df.corr()`

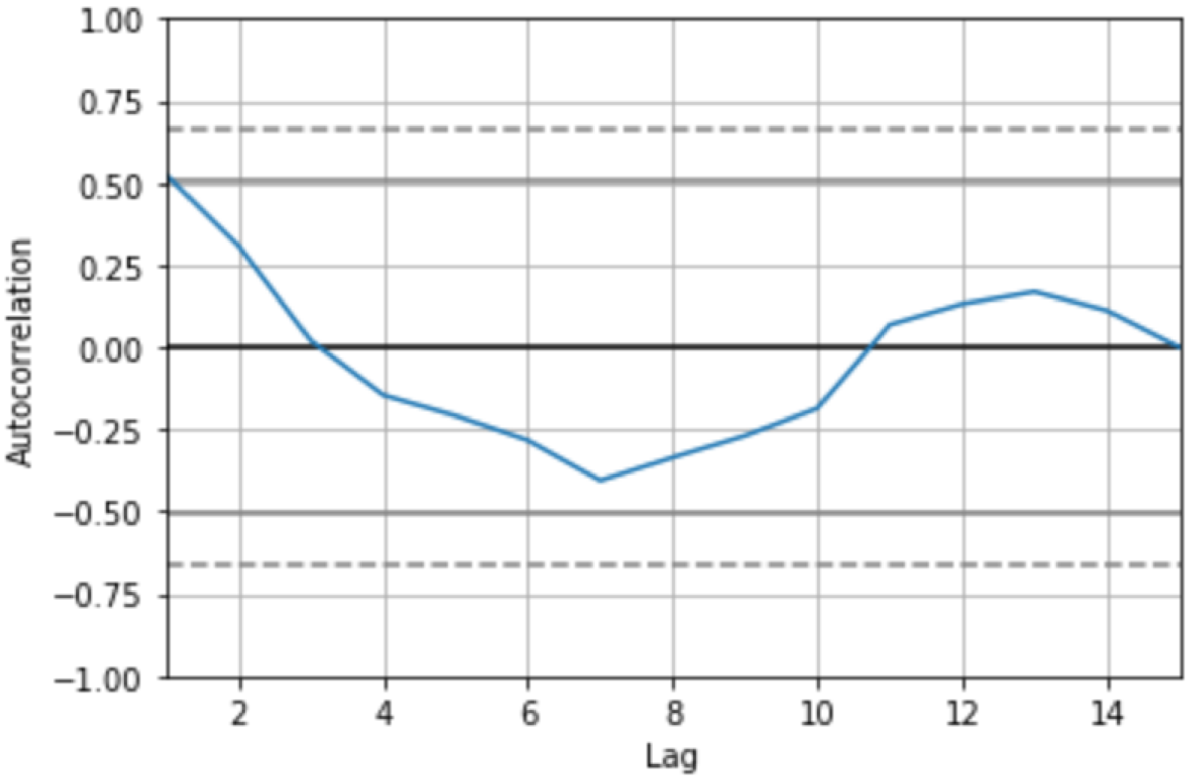
	Store	DayOfWeek	Sales	Customers	Open	Promo	SchoolHoliday
Store	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DayOfWeek	NaN	1.000000	-0.033374	0.025403	0.007911	-0.262466	-0.049294
Sales	NaN	-0.033374	1.000000	0.958332	0.645382	0.378689	-0.059832
Customers	NaN	0.025403	0.958332	1.000000	0.734367	0.212561	-0.087589
Open	NaN	0.007911	0.645382	0.734367	1.000000	0.023432	-0.113477
Promo	NaN	-0.262466	0.378689	0.212561	0.023432	1.000000	0.022670
SchoolHoliday	NaN	-0.049294	-0.059832	-0.087589	-0.113477	0.022670	1.000000

It is obvious that number of customers are highly correlated with Sales. However, we can’t have Customers as feature because that feature will be entered at the end of the day. So, it won’t be a forecast. But, we can use it as lagged variable.

In time series data, lagged variables are frequently used. As an example, yesterday’s sales have an effect on today’s sales, so that is t-1 lagged variable. We can check for all lagged variables using `autocorrelation_plot(df.head(50))` We limit for 50, otherwise it’d take too much time.



Taking a closer look with 15 limit:



We see that t-1 and t-2 are good candidates for correlation but others don't seem to be much correlated. As I said before, Customers feature is highly correlated with Sales, so we can add its lagged variables as well. We'll add them as features and see the correlation matrix.

```
df['SalesMinus1'] = df['Sales'].shift(1)
df['SalesMinus2'] = df['Sales'].shift(2)
df['CustomersMinus1'] = df['Customers'].shift(1)
df['CustomersMinus2'] = df['Customers'].shift(2)
df = df.dropna()
```

	Store	DayOfWeek	Sales	Customers	Open	Promo	SchoolHoliday	SalesMinus1	SalesMinus2	CustomersMinus1	CustomersMinus2
Store	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DayOfWeek	NaN	1.000000	-0.029092	0.029620	0.008462	-0.260097	-0.043901	-0.183720	-0.144556	-0.195466	-0.158561
Sales	NaN	-0.029092	1.000000	0.958147	0.646582	0.376548	-0.066929	0.303327	0.234385	0.246802	0.212661
Customers	NaN	0.029620	0.958147	1.000000	0.735533	0.209827	-0.094472	0.243688	0.194514	0.225318	0.196972
Open	NaN	0.008462	0.646582	0.735533	1.000000	0.022961	-0.114913	-0.019861	-0.038924	0.000077	-0.008428
Promo	NaN	-0.260097	0.376548	0.209827	0.022961	1.000000	0.017394	0.398975	0.350571	0.274336	0.261496
SchoolHoliday	NaN	-0.043901	-0.066929	-0.094472	-0.114913	0.017394	1.000000	-0.023455	-0.010176	-0.049033	-0.039577
SalesMinus1	NaN	-0.183720	0.303327	0.243688	-0.019861	0.398975	-0.023455	1.000000	0.304198	0.958288	0.247926
SalesMinus2	NaN	-0.144556	0.234385	0.194514	-0.038924	0.350571	-0.010176	0.304198	1.000000	0.244959	0.958502
CustomersMinus1	NaN	-0.195466	0.246802	0.225318	0.000077	0.274336	-0.049033	0.958288	0.244959	1.000000	0.226555
CustomersMinus2	NaN	-0.158561	0.212661	0.196972	-0.008428	0.261496	-0.039577	0.247926	0.958502	0.226555	1.000000

Random Forest

Now that we have incorporated some features from time series, we can train and test models. We'll start with Random Forest.

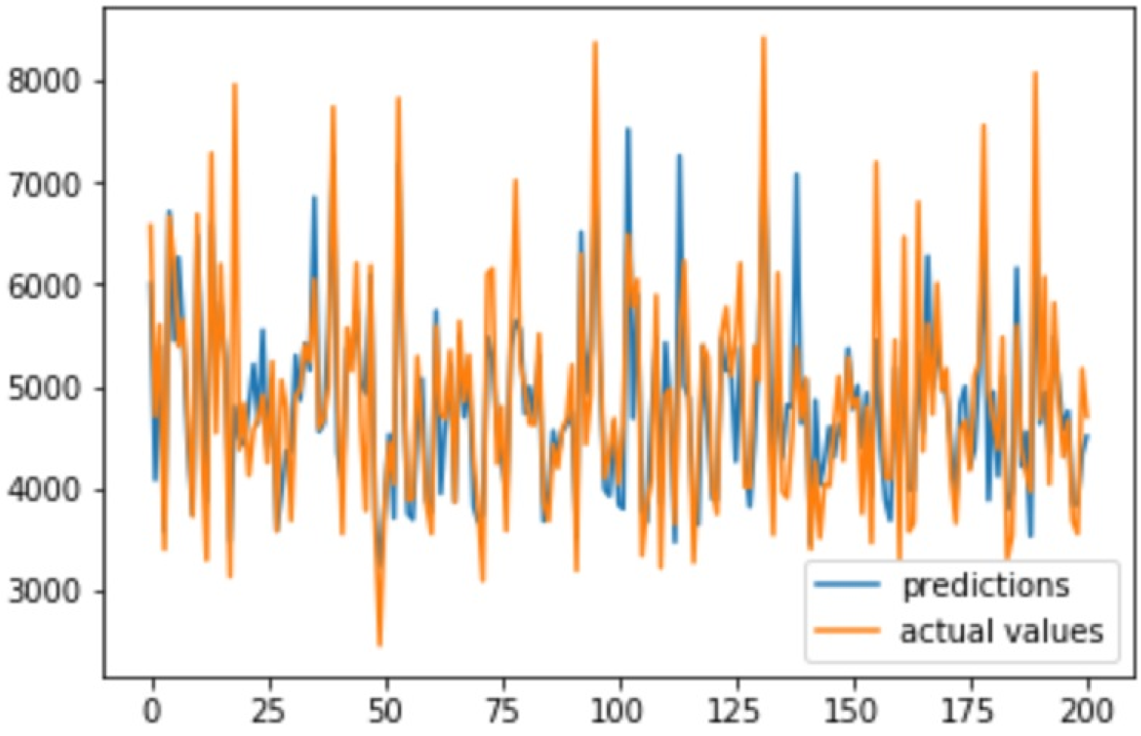
```
'''
Feature Extraction
'''
import pandas as pd
import statsmodels.api as sm
import datetime
df = pd.read_csv("dataset/train.csv", sep=',', parse_dates=[2])

df = df[df['Store'] == 1.0][df['Date']>datetime.date(2013,1,6)].sort_values(by='Date')
df = df[df['DayOfWeek'] != 7]

df['SalesMinus1'] = df['Sales'].shift(1)
df['SalesMinus2'] = df['Sales'].shift(2)
df['CustomersMinus1'] = df['Customers'].shift(1)
df['CustomersMinus2'] = df['Customers'].shift(2)
df = df.dropna()
df = df.drop('Customers', axis = 1)
df = pd.get_dummies(df, columns=['DayOfWeek'])
```

Here we converted DayOfWeek feature to one-hot encoding since it's a categorical feature and added lagged variables.

```
'''
Model Preperation
'''
import numpy as np
from sklearn.model_selection import train_test_split
labels = np.array(df['Sales'])
df_nosales = df.drop('Sales', axis = 1).drop('Date', axis = 1).drop('StateHoliday', axis = 1)
feature_list = list(df_nosales.columns)
np_data = np.array(df_nosales)

# Split the data into training and testing sets
train_features, test_features, train_labels, test_labels = train_test_split(np_data, labels, test_size = 0.25, ran

'''
Training
'''
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n_estimators = 10)

rf.fit(train_features, train_labels)

'''
Prediction
'''
from matplotlib import pyplot as plt

predictions = rf.predict(test_features)
# Fix for divide by 0 problem.
test_labels[test_labels == 0] = np.mean(test_labels)
predictions[predictions == 0] = np.mean(predictions)

errors = abs(predictions - test_labels)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
plt.plot(range(0, len(predictions)), predictions, label='predictions')
plt.plot(range(0, len(test_labels)), test_labels, label='actual values')
plt.legend()

# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / test_labels)
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

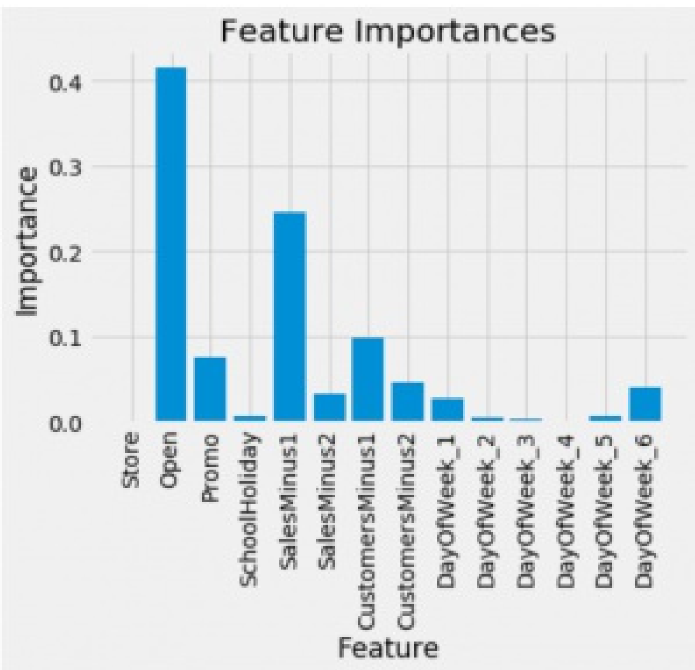
Output from the prediction is following:

```
Mean Absolute Error: 436.46 degrees.
Accuracy: 91.18 %.
```



We can also see how important each feature was in terms of success in prediction:

```
importances = list(rf.feature_importances_)
import matplotlib.pyplot as plt

%matplotlib inline
plt.style.use('fivethirtyeight')
x_values = list(range(len(importances)))

plt.bar(x_values, importances, orientation = 'vertical')
plt.xticks(x_values, feature_list, rotation='vertical')
plt.ylabel('Importance'); plt.xlabel('Feature'); plt.title('Feature Importances');
```



Neural Networks

We will use the same data and the feature set. Then compare the two models using the same scoring function.
Until the prediction step, most of the operations are same. But I'll add the whole code up until that point:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
import numpy as np
import pandas as pd
import statsmodels.api as sm
import datetime
np.random.seed(7)
df = pd.read_csv("dataset/train.csv", sep=',', parse_dates=[2])

df = df[df['Store'] == 1.0][df['Date']>datetime.date(2013,1,6)].sort_values(by='Date')
df = df[df['DayOfWeek'] != 7]

df['SalesMinus1'] = df['Sales'].shift(1)
df['SalesMinus2'] = df['Sales'].shift(2)
df['CustomersMinus1'] = df['Customers'].shift(1)
df['CustomersMinus2'] = df['Customers'].shift(2)
df = df.dropna()
df = df.drop(['Customers', 'StateHoliday', 'Store', 'Date'], axis = 1)
df = pd.get_dummies(df, columns=['DayOfWeek'])

'''
Data Preperation
'''
import numpy as np
from sklearn.model_selection import train_test_split
labels = np.array(df['Sales'])
df_nosales = df.drop('Sales', axis = 1)
feature_list = list(df_nosales.columns)
np_data = np.array(df_nosales)

# Split the data into training and testing sets
train_features, test_features, train_labels, test_labels = train_test_split(np_data, labels, test_size = 0.25, ran
```

At this point we have 4 datasets. 2 datasets are for training and 2 others are for testing, each pair having features and label as seperate datasets. Now, we'll define and compile our Neural Network:

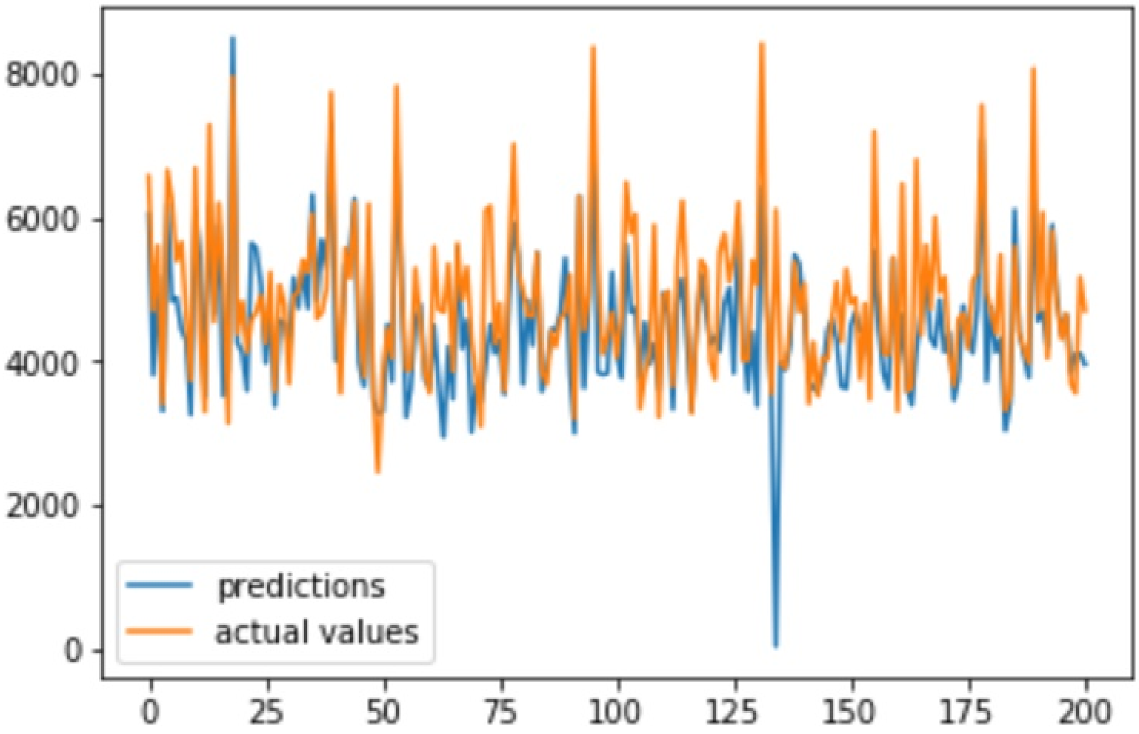
```
def sales_model():
# create model
model = Sequential()
model.add(Dense(13, input_dim=13, kernel_initializer='normal', activation='relu'))
model.add(Dense(1, kernel_initializer='normal'))
# Compile model
model.compile(loss='mean_squared_error', optimizer='adam')
return model
# evaluate model with standardized dataset
estimator = KerasRegressor(build_fn=sales_model, epochs=100, batch_size=5, verbose=0)
estimator.fit(train_features, train_labels)
```

At this point, our estimator is ready to provide predictions. Normally, one would run k-fold testing but since our data is timeseries data, it wouldn't be very beneficial. So, let's predict and run the tests and see the scores.

```
'''
Prediction
'''
from matplotlib import pyplot as plt

predictions = estimator.predict(test_features)
# Fix for divide by 0 problem.
test_labels[test_labels == 0] = np.mean(test_labels)
predictions[predictions == 0] = np.mean(predictions)

errors = abs(predictions - test_labels)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
plt.plot(range(0, len(predictions)), predictions, label='predictions')
plt.plot(range(0, len(test_labels)), test_labels, label='actual values')
plt.legend()

# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / test_labels)
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

Output is the following:

```
Mean Absolute Error: 583.82 degrees.
Accuracy: 88.54 %.
```

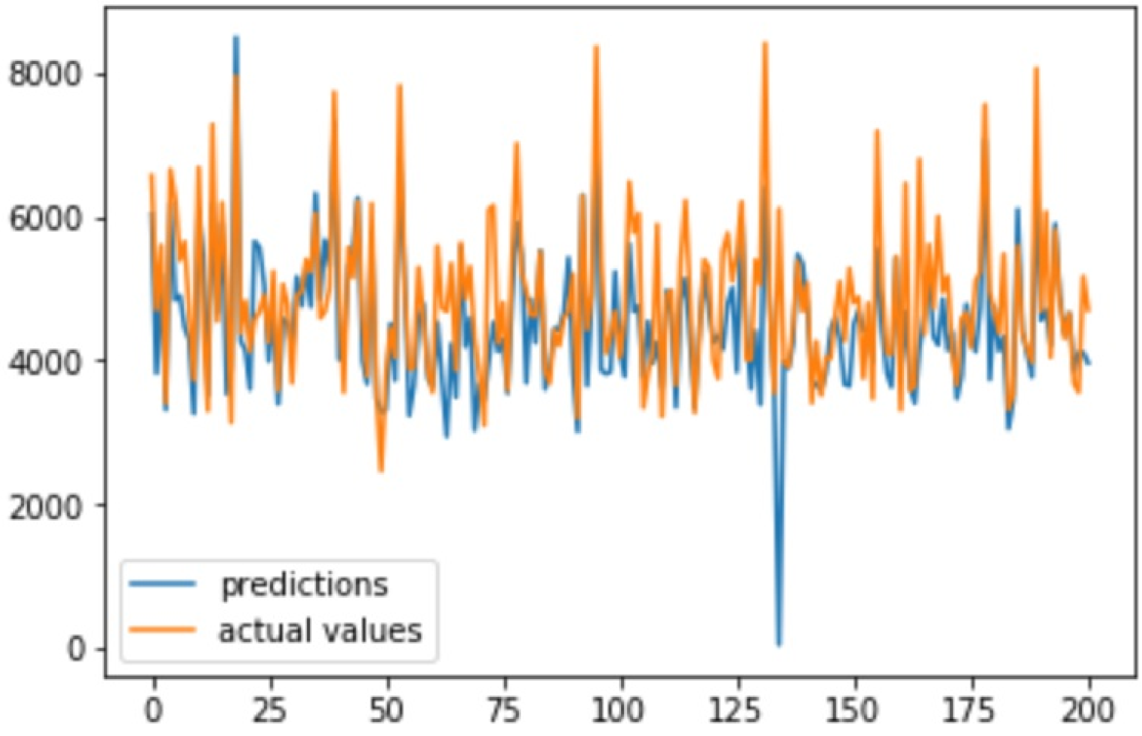


We see that overall, Random Forest model performs better on our data. We can tune both models to achieve better results. Let's see if we used a wider network in our model, what'd be the results.

```
def wider_model():
# create model
model = Sequential()
model.add(Dense(20, input_dim=13, kernel_initializer='normal', activation='relu'))
model.add(Dense(1, kernel_initializer='normal'))
# Compile model
model.compile(loss='mean_squared_error', optimizer='adam')
return model
```

Just change `sales_model` with `wider_model`. Following is the output:

```
Mean Absolute Error: 579.47 degrees.
Accuracy: 88.63 %.
```



As seen, there is not much of a difference in terms of error. One can tune both models, but generally Random Forest performs better than Neural Network in time series prediction.

You can find the whole code and the experiments that I did in [a public GitHub repository](). Most of them are in the form of Jupyter Notebooks. Neural Network is provided as standalone application as well as notebook. You can just run it `python NeuralNetworkStandalone.py` assuming needed packages are installed.

monus / May 22, 2018

Leave a Reply

Your email address will not be published. Required fields are marked *

COMMENT

NAME *

EMAIL *

WEBSITE

POST COMMENT

Search …

RECENT POSTS

- Random Forest and Neural Network on Sales Data
- How to do hypothesis testing on sales data?
- Forecasting Sales: Daily Demand Prediction of Products in Rossmann Stores
- How to detect emotion of the player in Unity?

RECENT COMMENTS

ARCHIVES

- May 2018
- March 2018
- February 2018
- January 2017

CATEGORIES

- Uncategorized

META

- Log in
- Entries RSS
- Comments RSS
- WordPress.org