SMOKE TEST

Analytics Zoo 0.7.0

Date Prepared: March 2020

Document Information

Project Name	**EPIC Accelerator Deployment & Integration Services**		
Project Owner		Document Version No	0.1
Quality Review Method			
Prepared By	Priyanka	Preparation Date	March 2020
Reviewed By		Review Date	

Table of Contents

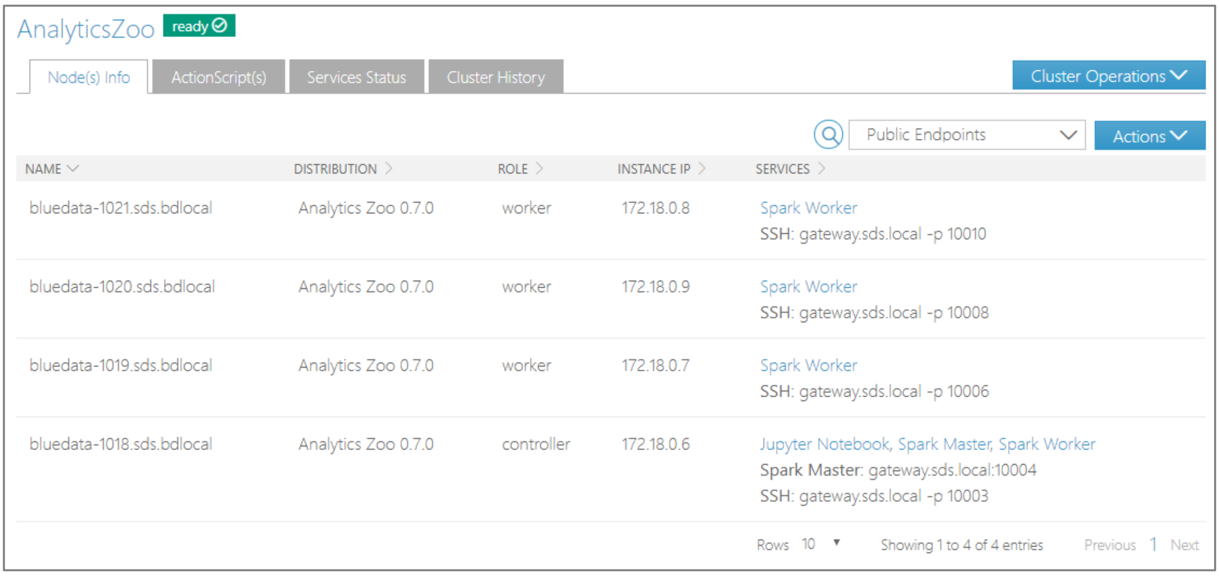
Table of Tables

NO TABLE OF FIGURES ENTRIES FOUND.

1 LOGIN TO JUPYTER NOTEBOOK

Login to Jupyter Notebook, using the following steps:

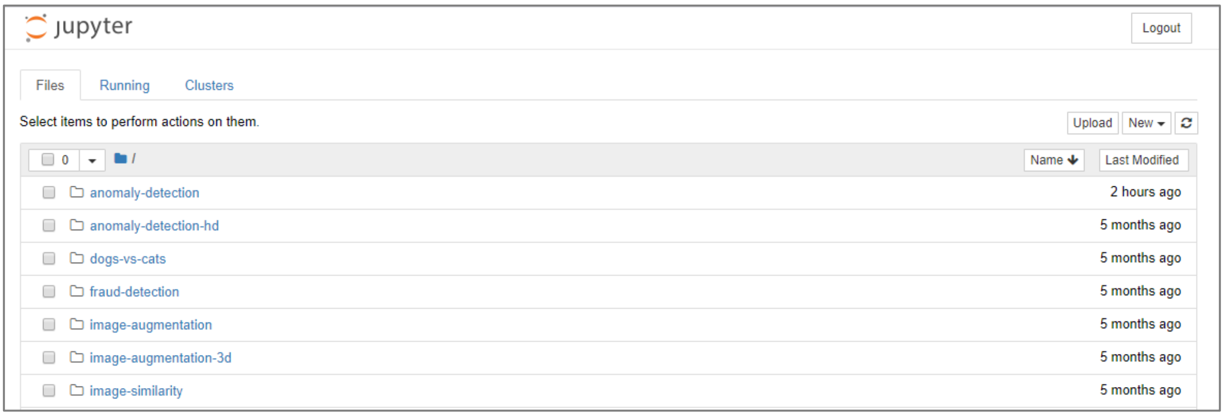
1. From the Cluster page under **SERVICES** column, click on **Jupyter Notebook**



2. It will open-up a new tab with Jupyter Notebook login page

3. Enter the password and click on **Log in**

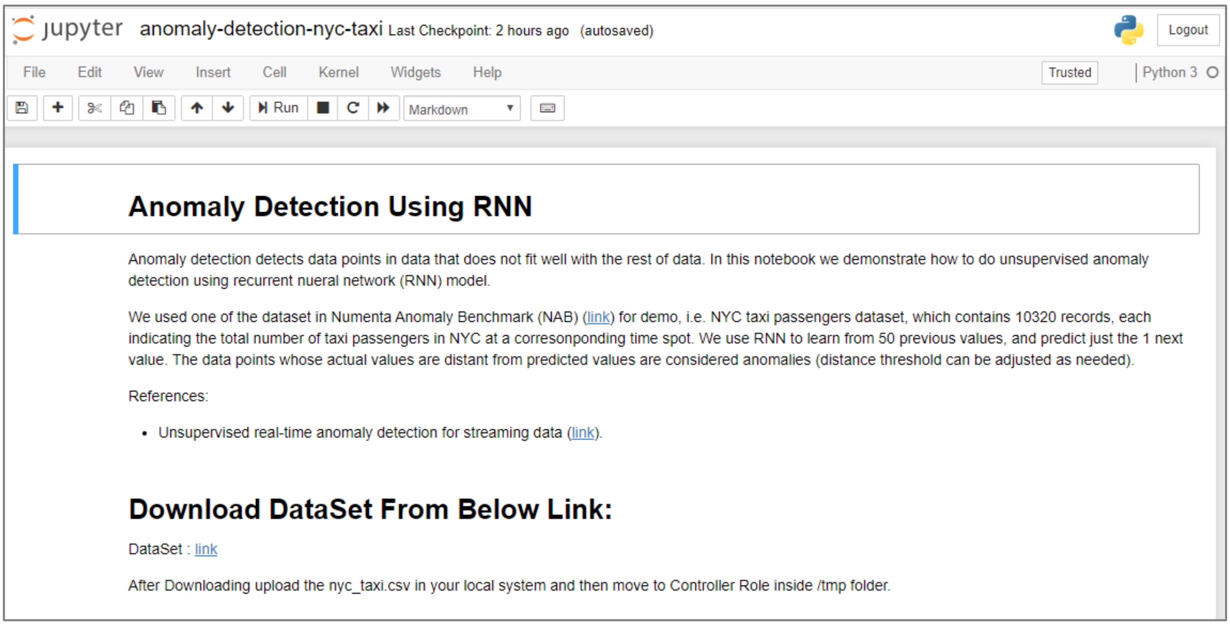
4. You will get Jupyter dashboard with some default files running


jupyter
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
0
/
Name
Last Modified
anomaly-detection
2 hours ago
anomaly-detection-hd
5 months ago
dogs-vs-cats
5 months ago
fraud-detection
5 months ago
image-augmentation
5 months ago
image-augmentation-3d
5 months ago
image-similarity
5 months ago

2 TESTING ANOMALY DETECTION EXAMPLE

Anomaly detection detects data points in data that does not fit well with the rest of data. In this notebook we demonstrate how to do unsupervised anomaly detection using recurrent neural network (RNN) model. This section is to test Anomaly detection example using the following procedure:

1. From the Jupyter dashboard, enter **anomaly-detection** folder and open the .ipynb file



2.1 Download DataSet

Note: Make sure to download the DataSet from the link and upload the downloaded file on the controller role in **/tmp** directory.

(https://raw.githubusercontent.com/numenta/NAB/master/data/realKnownCause/nyc_taxi.csv)

Here NYC taxi passenger's dataset is used. It consists of 10320 records, each indicating the total number of taxi passengers in NYC at a corresponding time spot. RNN is used to learn from 50 previous values, and predict just the 1 next value. The data points whose actual values are distant from predicted values are considered anomalies.

2.2 Initialization

1. To import the necessary libraries, execute the first cell by selecting the cell and clicking on **Run** from the toolbar

Run

Code

Intialization

- import necesary libraries

```
In [ ]: import os

        from zoo.common.nncontext import *

        """
        For Creating Custom Spark Context Create like below:

        from zoo.common.nncontext import *
        sparkConf = init_spark_conf().setAppName("Application Name").set(key="spark.executor.cores",value=4).set(key="spark.driver.cores"
        sc = init_nncontext(sparkConf)
        """
        sc = init_nncontext("Anomaly Detection Example")
        import pandas as pd
        import numpy as np

        import matplotlib
        matplotlib.use('Agg')
        %pylab inline
        import seaborn
        import matplotlib.dates as md
        from matplotlib import pyplot as plt

        from sklearn import preprocessing
```

2. The output will be like below

```
Using /usr/lib/spark/spark-2.4.3-bin-hadoop2.7
Prepending /usr/lib/spark/spark-2.4.3-bin-hadoop2.7/python/lib/py4j-0.10.7-src.zip to sys.path
Prepending /usr/lib/spark/spark-2.4.3-bin-hadoop2.7/python/lib/pyspark.zip to sys.path

WARNING:root:Trying to search from: /opt/work/analytics-zoo-0.7.0-SNAPSHOT/lib/analytics-zoo-bigdl_0.9.1-spark_2.4.3-0.7.0-SNAP
SHOT-python-api.zip/zoo, but can not find the jar for Analytics-Zoo

Populating the interactive namespace from numpy and matplotlib
```

3. Similarly import some necessary modules

- import necessary modules

```
In [2]: from zoo.pipeline.api.keras.layers import Dense, Dropout, LSTM
        from zoo.pipeline.api.keras.models import Sequential
```

2.3 Data Check

1. To read data, execute the below cell

Data Check

- read data

In [3]:
```
try:
    dataset_path = "/tmp/nyc_taxi.csv" #Please change the file name if you saved file with differnet name
    df = pd.read_csv(dataset_path)
except Exception as e:
    print("nyc_taxi.csv doesn't exist")
    print("File: {} doest not exist in Path:".format(nyc_taxi.csv, "/tmp"))
```

2. To understand the data better, execute the subsequent cells

- Understand the data.

Each record is in format of (timestamp, value). Timestamps range between 2014-07-01 and 2015-01-31.

In [4]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10320 entries, 0 to 10319
Data columns (total 2 columns):
timestamp    10320 non-null object
value        10320 non-null int64
dtypes: int64(1), object(1)
memory usage: 161.3+ KB
None
```

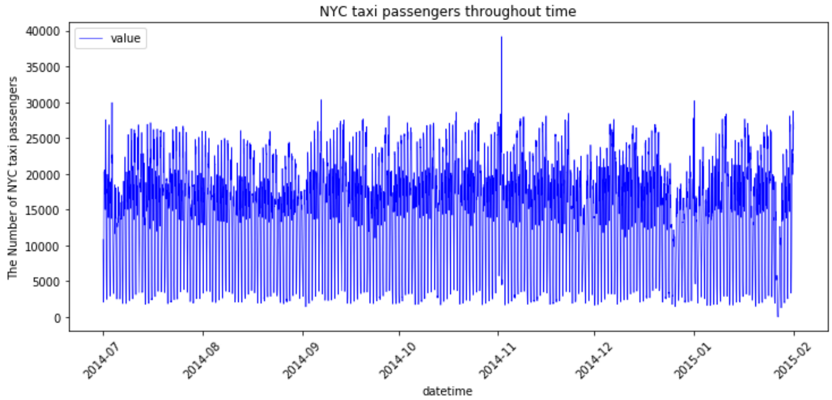
In [5]:
```
# check the timestamp format and frequence
print(df['timestamp'].head(10))
```

```
0    2014-07-01 00:00:00
1    2014-07-01 00:30:00
2    2014-07-01 01:00:00
3    2014-07-01 01:30:00
4    2014-07-01 02:00:00
5    2014-07-01 02:30:00
6    2014-07-01 03:00:00
7    2014-07-01 03:30:00
8    2014-07-01 04:00:00
9    2014-07-01 04:30:00
Name: timestamp, dtype: object
```

3. Below cell prints the mean value of passenger number and change the timestamp type for plotting along with the visualization of anomaly throughout time

```
In [6]: # check the mean of passenger number
        print(df['value'].mean())

15137.569379844961
```

```
In [7]: # change the type of timestamp column for plotting
        df['datetime'] = pd.to_datetime(df['timestamp'])

        # visualisation of anomaly throughout time (viz 1)
        fig, ax = plt.subplots(figsize=(12, 5))

        ax.plot(df['datetime'], df['value'], color='blue', linewidth=0.6)
        ax.set_title('NYC taxi passengers throughout time')

        plt.xlabel('datetime')
        plt.xticks(rotation=45)
        plt.ylabel('The Number of NYC taxi passengers')
        plt.legend(loc='upper left')
        plt.show()
```



2.4 Feature Engineering

1. To extract useful features, execute the below cell

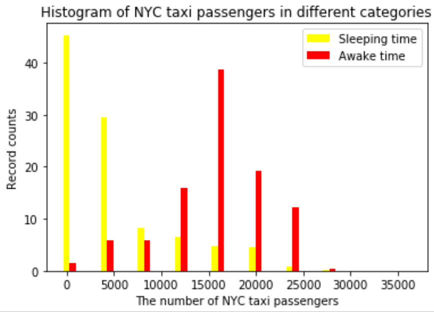
Feature engineering

- Extracting some useful features

```
In [8]: # the hours when people are awake (6:00-00:00)
        df['hours'] = df['datetime'].dt.hour
        df['awake'] = (((df['hours'] >= 6) & (df['hours'] <= 23)) | (df['hours'] == 0)).astype(int)
```

2. Below cell prints the histogram of NYC taxi passengers in different categories

```
In [9]: # creation of 2 distinct categories that seem useful (sleeping time and awake time)
df['categories'] = df['awake']

a = df.loc[df['categories'] == 0, 'value']
b = df.loc[df['categories'] == 1, 'value']

fig, ax = plt.subplots()
a_heights, a_bins = np.histogram(a)
b_heights, b_bins = np.histogram(b, bins=a_bins)

width = (a_bins[1] - a_bins[0])/6

ax.bar(a_bins[:-1], a_heights*100/a.count(), width=width, facecolor='yellow', label='Sleeping time')
ax.bar(b_bins[:-1]+width, (b_heights*100/b.count()), width=width, facecolor='red', label ='Awake time')
ax.set_title('Histogram of NYC taxi passengers in different categories')

plt.xlabel('The number of NYC taxi passengers')
plt.ylabel('Record counts')
plt.legend()
plt.show()
```




```
In [10]: df['awake'].head(4)
Out[10]: 0    1
         1    1
         2    0
         3    0
         Name: awake, dtype: int64

In [11]: df['timestamp'].head(4)
Out[11]: 0    2014-07-01 00:00:00
         1    2014-07-01 00:30:00
         2    2014-07-01 01:00:00
         3    2014-07-01 01:30:00
         Name: timestamp, dtype: object
```

From the above result, we can conclude:

- more people take taxi when they are awake

2.5 Data Preparation

1. Here data is standardized into test and train data

Data Preparation

- Standardizing data and spliting them into the train data and the test data

```
In [12]: #select and standardize data
data_n = df[['value', 'hours', 'awake']]
standard_scaler = preprocessing.StandardScaler()
np_scaled = standard_scaler.fit_transform(data_n)
data_n = pd.DataFrame(np_scaled)

#important parameters and train/test size
prediction_time = 1
testdatasize = 1000
unroll_length = 50
testdatacut = testdatasize + unroll_length  + 1

#train data
x_train = data_n[0:-prediction_time-testdatacut].as_matrix()
y_train = data_n[prediction_time:-testdatacut  ][0].as_matrix()

#test data
x_test = data_n[0-testdatacut:-prediction_time].as_matrix()
y_test = data_n[prediction_time-testdatacut:  ][0].as_matrix()
```

2. Here datasets are adapt for the sequence data shape

```
In [13]: #unroll: create sequence of 50 previous data points for each data points
def unroll(data,sequence_length=24):
    result = []
    for index in range(len(data) - sequence_length):
        result.append(data[index: index + sequence_length])
    return np.asarray(result)

# adapt the datasets for the sequence data shape
x_train = unroll(x_train,unroll_length)
x_test  = unroll(x_test,unroll_length)
y_train = y_train[-x_train.shape[0]:]
y_test  = y_test[-x_test.shape[0]:]

# see the shape
print("x_train", x_train.shape)
print("y_train", y_train.shape)
print("x_test", x_test.shape)
print("y_test", y_test.shape)

x_train (9218, 50, 3)
y_train (9218,)
x_test (1000, 50, 3)
y_test (1000,)
```

2.6 Build Data

1. Here RNN network is build using Analytics Zoo Keras-Style API

Build Model

- Here we show an example of building a RNN network using Analytics Zoo Keras-Style API. There are three LSTM layers and one Dense layer.

In [14]:
```
# Build the model
model = Sequential()

model.add(LSTM(
    input_shape=(x_train.shape[1], x_train.shape[-1]),
    output_dim=20,
    return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(
    10,
    return_sequences=False))
model.add(Dropout(0.2))

model.add(Dense(
    output_dim=1))

model.compile(loss='mse', optimizer='rmsprop')
```

```
creating: createZooKerasSequential
creating: createZooKerasLSTM
creating: createZooKerasDropout
creating: createZooKerasLSTM
creating: createZooKerasDropout
creating: createZooKerasDense
creating: createRMSprop
creating: createZooKerasMeanSquaredError
```

2.7 Train Model

1. To train the model, execute the below cell

Train the model

In [15]:
```
%%time
# Train the model
print("Training begins.")
model.fit(
    x_train,
    y_train,
    batch_size=1024,
    nb_epoch=2)
print("Training completed.")
```

```
Training begins.
Training completed.
CPU times: user 124 ms, sys: 31.9 ms, total: 156 ms
Wall time: 34.2 s
```

2.8 Prediction

1. Predicted points are determined here

Prediction ¶

- BigDL models make inferences based on the given data using model.predict(val_rdd) API. A result of RDD is returned. predict_class returns the predicted points.

In [16]:
```
# create the list of difference between prediction and test data
diff=[]
ratio=[]
predictions = model.predict(x_test)
p = predictions.collect()
for u in range(len(y_test)):
    pr = p[u][0]
    ratio.append((y_test[u]/pr)-1)
    diff.append(abs(y_test[u]- pr))
```

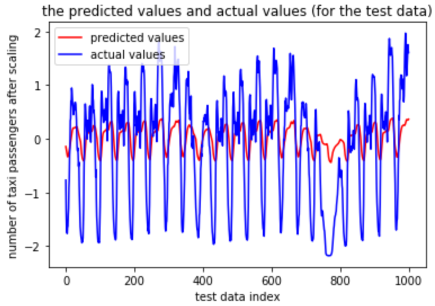
2.9 Evaluation

1. This cell plots prediction and reality

Evaluation

- plot the prediction and the reality

In [17]:
```
# plot the predicted values and actual values (for the test data)
fig, axs = plt.subplots()

axs.plot(p,color='red', label='predicted values')
axs.plot(y_test,color='blue', label='actual values')
axs.set_title('the predicted values and actual values (for the test data)')

plt.xlabel('test data index')
plt.ylabel('number of taxi passengers after scaling')
plt.legend(loc='upper left')
plt.show()
```



2. In the below cell, distance threshold is set for anomalies

- Set the distance thresold for anomalies. There're many ways to select this threshold. Here we set the expected proportion of anomalies among the entire set. Then we set the threshold as the minimum value of top N distances (here N is the total number of anomalies, i.e. anomaly fraction * total no. of samples)

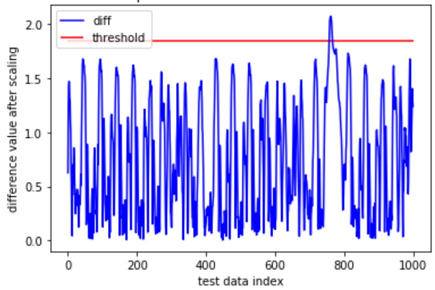
In [18]:
```
# An estimation of anomly population of the dataset
outliers_fraction = 0.01
# select the most distant prediction/reality data points as anomalies
diff = pd.Series(diff)
number_of_outliers = int(outliers_fraction*len(diff))
threshold = diff.nlargest(number_of_outliers).min()
```

3. Plot anomaly

- plot anomalies in the test data throughout time

In [19]:
```
# plot the difference and the threshold (for the test data)
fig, axs = plt.subplots()

axs.plot(diff,color='blue', label='diff')
axs.set_title('the difference between the predicted values and actual values with the threshold line')

plt.hlines(threshold, 0, 1000, color='red', label='threshold')
plt.xlabel('test data index')
plt.ylabel('difference value after scaling')
plt.legend(loc='upper left')
plt.show()
```

the difference between the predicted values and actual values with the threshold line

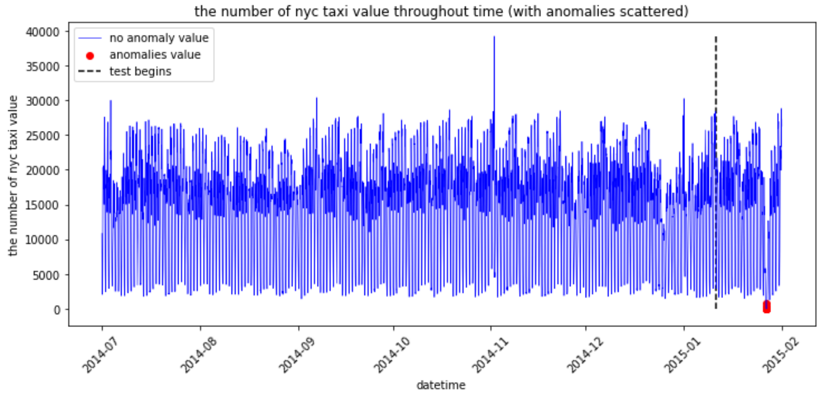


In [20]:
```
# data with anomaly label (test data part)
test = (diff >= threshold).astype(int)
# the training data part where we didn't predict anything (overfitting possible): no anomaly
complement = pd.Series(0, index=np.arange(len(data_n)-testdatasize))
last_train_data= (df['datetime'].tolist())[-testdatasize]
# add the data to the main
df['anomaly27'] = complement.append(test, ignore_index='True')
```

4. Below cell prints the visualization of anomaly throughout time

- plot anomalies in the test data throughout time

In [21]:
```
# visualisation of anomaly throughout time (viz 1)
fig, ax = plt.subplots(figsize=(12, 5))

a = df.loc[df['anomaly27'] == 1, ['datetime', 'value']] #anomaly
ax.plot(df['datetime'], df['value'], color='blue', label='no anomaly value', linewidth=0.6)
ax.scatter(a['datetime'].tolist(),a['value'], color='red', label='anomalies value')
ax.set_title('the number of nyc taxi value throughout time (with anomalies scattered)')

max_value = df['value'].max()
min_value = df['value'].min()
plt.vlines(last_train_data, min_value, max_value, color='black', linestyles = "dashed", label='test begins')
plt.xlabel('datetime')
plt.xticks(rotation=45)
plt.ylabel('the number of nyc taxi value')
plt.legend(loc='upper left')
plt.show()
```



In [22]:
```
sc.stop()
```

In [ ]: