Welcome to the final project of “Apache Spark for Scalable Machine Learning on BigData”. In this assignment you’ll analyze a real-world dataset and apply machine learning on it using Apache Spark.

In order to pass, you need to implement some code (as described in the instruction section on Coursera) and finally answer a quiz on the Coursera platform.

This notebook is designed to run in a IBM Watson Studio default runtime (NOT the Watson Studio Apache Spark Runtime as the default runtime with 1 vCPU is free of charge). Therefore, we install Apache Spark in local mode for test purposes only. Please don't use it in production.

In case you are facing issues, please read the following two documents first:

https://github.com/IBM/skillsnetwork/wiki/Environment-Setup (https://github.com/IBM/skillsnetwork/wiki/Environment-Setup)

https://github.com/IBM/skillsnetwork/wiki/FAQ (https://github.com/IBM/skillsnetwork/wiki/FAQ)

Then, please feel free to ask:

https://coursera.org/learn/machine-learning-big-data-apache-spark/discussions/all (https://coursera.org/learn/machine-learning-big-data-apache-spark/discussions/all)

Please make sure to follow the guidelines before asking a question:

https://github.com/IBM/skillsnetwork/wiki/FAQ#im-feeling-lost-and-confused-please-help-me (https://github.com/IBM/skillsnetwork/wiki/FAQ#im-feeling-lost-and-confused-please-help-me)

If running outside Watson Studio, this should work as well. In case you are running in an Apache Spark context outside Watson Studio, please remove the Apache Spark setup in the first notebook cells.

In [1]:

```
from IPython.display import Markdown, display
def printmd(string):
    display(Markdown('# <span style="color:red">'+string+'</span>'))


if ('sc' in locals() or 'sc' in globals()):
    printmd('<<<<<!!!!! It seems that you are running in a IBM Watson Studio Apache Spa
rk Notebook. Please run it in an IBM Watson Studio Default Runtime (without Apache Spar
k) !!!!!>>>>>')
```

In [2]:

```
!pip install pyspark==2.4.5
```

```
/opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages/secretstorage/
dhcrypto.py:16: CryptographyDeprecationWarning: int_from_bytes is deprecat
ed, use int.from_bytes instead
  from cryptography.utils import int_from_bytes
/opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages/secretstorage/
util.py:25: CryptographyDeprecationWarning: int_from_bytes is deprecated,
use int.from_bytes instead
  from cryptography.utils import int_from_bytes
Collecting pyspark==2.4.5
  Downloading pyspark-2.4.5.tar.gz (217.8 MB)
     |████████████████████████████████| 217.8 MB 12 kB/s s eta 0:00:01
| 52.8 MB 11.6 MB/s eta 0:00:15
Collecting py4j==0.10.7
  Downloading py4j-0.10.7-py2.py3-none-any.whl (197 kB)
     |████████████████████████████████| 197 kB 58.3 MB/s eta 0:00:01
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-2.4.5-py2.py3-none-any.whl s
ize=218257927 sha256=b01df607f2457e4375dd31a6be83200238645b410e82aed07d0a9
f57e08fba7b
  Stored in directory: /tmp/wsuser/.cache/pip/wheels/01/c0/03/1c241c9c482b
647d4d99412a98a5c7f87472728ad41ae55e1e
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.7 pyspark-2.4.5
```

In [3]:

```
try:
    from pyspark import SparkContext, SparkConf
    from pyspark.sql import SparkSession
except ImportError as e:
    printmd('<<<<<!!!!! Please restart your kernel after installing Apache Spark !!!!!>
>>>>')
```

In [4]:

```
sc = SparkContext.getOrCreate(SparkConf().setMaster("local[*]"))

spark = SparkSession \
    .builder \
    .getOrCreate()
```

Let's start by downloading the dataset and creating a dataframe. This dataset can be found on DAX, the IBM Data Asset Exchange and can be downloaded for free.

[https://developer.ibm.com/exchanges/data/all/jfk-weather-data/ (https://developer.ibm.com/exchanges/data/all/jfk-weather-data/)](https://developer.ibm.com/exchanges/data/all/jfk-weather-data/)

In [5]:

```
# delete files from previous runs
!rm -rf noaa-weather-data-jfk-airport*

# download the file containing the data in CSV format
!wget https://dax-cdn.cdn.appdomain.cloud/dax-noaa-weather-data-jfk-airport/1.1.4/noaa-
weather-data-jfk-airport.tar.gz

# extract the data
!tar xvfz noaa-weather-data-jfk-airport.tar.gz noaa-weather-data-jfk-airport/jfk_weathe
r.csv

# create a dataframe out of it by using the first row as field names and trying to infe
r a schema based on contents
df = spark.read.option("header", "true").option("inferSchema","true").csv('noaa-weather
-data-jfk-airport/jfk_weather.csv')

# register a corresponding query table
df.createOrReplaceTempView('df')
```

```
--2021-04-13 05:49:28--  https://dax-cdn.cdn.appdomain.cloud/dax-noaa-weat
her-data-jfk-airport/1.1.4/noaa-weather-data-jfk-airport.tar.gz
Resolving dax-cdn.cdn.appdomain.cloud (dax-cdn.cdn.appdomain.cloud)... 23.
198.7.75, 23.198.7.68, 2600:1404:1400:1::687c:395b, ...
Connecting to dax-cdn.cdn.appdomain.cloud (dax-cdn.cdn.appdomain.cloud)|2
3.198.7.75|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3509993 (3.3M) [application/x-tar]
Saving to: ‘noaa-weather-data-jfk-airport.tar.gz’

noaa-weather-data-j 100%[===================>]   3.35M  --.-KB/s    in 0.0
2s

2021-04-13 05:49:28 (158 MB/s) - ‘noaa-weather-data-jfk-airport.tar.gz’ sa
ved [3509993/3509993]

noaa-weather-data-jfk-airport/jfk_weather.csv
```

The dataset contains some null values, therefore schema inference didn’t work properly for all columns, in addition, a column contained trailing characters, so we need to clean up the data set first. This is a normal task in any data science project since your data is never clean, don’t worry if you don’t understand all code, you won’t be asked about it.

In [6]:

```
import random
random.seed(42)

from pyspark.sql.functions import translate, col

df_cleaned = df \
    .withColumn("HOURLYWindSpeed", df.HOURLYWindSpeed.cast('double')) \
    .withColumn("HOURLYWindDirection", df.HOURLYWindDirection.cast('double')) \
    .withColumn("HOURLYStationPressure", translate(col("HOURLYStationPressure"), "s,",
"")) \
    .withColumn("HOURLYPrecip", translate(col("HOURLYPrecip"), "s,", "")) \
    .withColumn("HOURLYRelativeHumidity", translate(col("HOURLYRelativeHumidity"), "*",
"")) \
    .withColumn("HOURLYDRYBULBTEMPC", translate(col("HOURLYDRYBULBTEMPC"), "*", "")) \

df_cleaned =   df_cleaned \
                    .withColumn("HOURLYStationPressure", df_cleaned.HOURLYStationPressu
re.cast('double')) \
                    .withColumn("HOURLYPrecip", df_cleaned.HOURLYPrecip.cast('double'))
\
                    .withColumn("HOURLYRelativeHumidity", df_cleaned.HOURLYRelativeHumi
dity.cast('double')) \
                    .withColumn("HOURLYDRYBULBTEMPC", df_cleaned.HOURLYDRYBULBTEMPC.cas
t('double')) \

df_filtered = df_cleaned.filter("""
    HOURLYWindSpeed <> 0
    and HOURLYWindSpeed IS NOT NULL
    and HOURLYWindDirection IS NOT NULL
    and HOURLYStationPressure IS NOT NULL
    and HOURLYPressureTendency IS NOT NULL
    and HOURLYPrecip IS NOT NULL
    and HOURLYRelativeHumidity IS NOT NULL
    and HOURLYDRYBULBTEMPC IS NOT NULL
""")
```

We want to predict the value of one column based of some others. It is sometimes helpful to print a correlation matrix.

In [8]:

```
from pyspark.ml.feature import VectorAssembler
vectorAssembler = VectorAssembler(inputCols=["HOURLYWindSpeed","HOURLYWindDirection",
"HOURLYStationPressure", "HOURLYPressureTendency"],
                                  outputCol="features")
df_pipeline = vectorAssembler.transform(df_filtered)
from pyspark.ml.stat import Correlation
Correlation.corr(df_pipeline,"features").head()[0].toArray()
```

Out[8]:

```
array([[ 1.        ,  0.25478863, -0.26171147, -0.01324305],
       [ 0.25478863,  1.        , -0.13377466, -0.18145395],
       [-0.26171147, -0.13377466,  1.        , -0.05821663],
       [-0.01324305, -0.18145395, -0.05821663,  1.        ]])
```

As we can see, HOURLYWindSpeed and HOURLYWindDirection correlate with 0.25478863 whereas HOURLYWindSpeed and HOURLYStationPressure correlate with -0.26171147, this is a good sign if we want to predict HOURLYWindSpeed from HOURLYWindDirection and HOURLYStationPressure. Note that the numbers can change over time as we are always working with the latest data. Since this is supervised learning, let’s split our data into train (80%) and test (20%) set.

In [9]:

```
splits = df_filtered.randomSplit([0.8, 0.2])
df_train = splits[0]
df_test = splits[1]
```

Again, we can re-use our feature engineering pipeline

In [10]:

```
from pyspark.ml.feature import StringIndexer, OneHotEncoder
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import Normalizer
from pyspark.ml import Pipeline

vectorAssembler = VectorAssembler(inputCols=[
                                    "HOURLYWindDirection",
                                    "ELEVATION",
                                    "HOURLYStationPressure"],
                                  outputCol="features")

normalizer = Normalizer(inputCol="features", outputCol="features_norm", p=1.0)
```

Now we define a function for evaluating our regression prediction performance. We’re using RMSE (Root Mean Squared Error) here , the smaller the better…

In [11]:

```
def regression_metrics(prediction):
    from pyspark.ml.evaluation import RegressionEvaluator
    evaluator = RegressionEvaluator(
    labelCol="HOURLYWindSpeed", predictionCol="prediction", metricName="rmse")
    rmse = evaluator.evaluate(prediction)
    print("RMSE on test data = %g" % rmse)
```

Let’s run a linear regression model first for building a baseline.

In [12]:

```
#LR1

from pyspark.ml.regression import LinearRegression

lr = LinearRegression(labelCol="HOURLYWindSpeed", featuresCol='features', maxIter=100,
regParam=0.0, elasticNetParam=0.0)
pipeline = Pipeline(stages=[vectorAssembler, normalizer,lr])
model = pipeline.fit(df_train)
prediction = model.transform(df_test)
regression_metrics(prediction)
```

```
RMSE on test data = 5.30775
```

Run linear regression model with features_norm

In [13]:

```
#LR2 with features normalized
from pyspark.ml.regression import LinearRegression

lr2 = LinearRegression(labelCol="HOURLYWindSpeed", featuresCol='features_norm', maxIter
=100, regParam=0.0, elasticNetParam=0.0)
pipeline2 = Pipeline(stages=[vectorAssembler, normalizer,lr2])
model2 = pipeline2.fit(df_train)
prediction2 = model2.transform(df_test)
regression_metrics(prediction2)
```

```
RMSE on test data = 5.53835
```

Now we’ll try a Gradient Boosted Tree Regressor

In [14]:

```
#GBT1

from pyspark.ml.regression import GBTRegressor
gbt = GBTRegressor(labelCol="HOURLYWindSpeed", maxIter=100)
pipeline = Pipeline(stages=[vectorAssembler, normalizer,gbt])
model = pipeline.fit(df_train)
prediction = model.transform(df_test)
regression_metrics(prediction)
```

```
RMSE on test data = 5.12168
```

Now let’s switch gears. Previously, we tried to predict HOURLYWindSpeed, but now we predict HOURLYWindDirection. In order to turn this into a classification problem we discretize the value using the Bucketizer. The new feature is called HOURLYWindDirectionBucketized.

In [15]:

```
from pyspark.ml.feature import Bucketizer, OneHotEncoder
bucketizer = Bucketizer(splits=[ 0, 180, float('Inf') ],inputCol="HOURLYWindDirection",
outputCol="HOURLYWindDirectionBucketized")
encoder = OneHotEncoder(inputCol="HOURLYWindDirectionBucketized", outputCol="HOURLYWind
DirectionOHE")
```

Again, we define a function in order to assess how we perform. Here we just use the accuracy measure which gives us the fraction of correctly classified examples. Again, 0 is bad, 1 is good.

In [16]:

```
def classification_metrics(prediction):
    from pyspark.ml.evaluation import MulticlassClassificationEvaluator
    mcEval = MulticlassClassificationEvaluator().setMetricName("accuracy") .setPredicti
onCol("prediction").setLabelCol("HOURLYWindDirectionBucketized")
    accuracy = mcEval.evaluate(prediction)
    print("Accuracy on test data = %g" % accuracy)
```

Again, for baselining we use LogisticRegression.

In [17]:

```
#LGReg1

from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(labelCol="HOURLYWindDirectionBucketized", maxIter=10)

vectorAssembler = VectorAssembler(inputCols=["HOURLYWindSpeed","HOURLYDRYBULBTEMPC", "E
LEVATION","HOURLYStationPressure",
                                             "HOURLYPressureTendency","HOURLYPrecip"],
                                  outputCol="features")

pipeline = Pipeline(stages=[bucketizer,vectorAssembler,normalizer,lr])
model = pipeline.fit(df_train)
prediction = model.transform(df_test)
classification_metrics(prediction)
```

```
Accuracy on test data = 0.687443
```

Let’s try some other Algorithms and see if model performance increases. It’s also important to tweak other parameters like parameters of individual algorithms (e.g. number of trees for RandomForest) or parameters in the feature engineering pipeline, e.g. train/test split ratio, normalization, bucketing, …

In [20]:

```
#RF1

from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(labelCol="HOURLYWindDirectionBucketized", numTrees=10)

vectorAssembler = VectorAssembler(inputCols=["HOURLYWindSpeed","HOURLYDRYBULBTEMPC","EL
EVATION","HOURLYStationPressure",
                                             "HOURLYPressureTendency","HOURLYPrecip"],
                                  outputCol="features")

pipeline = Pipeline(stages=[bucketizer,vectorAssembler,normalizer,rf])
model = pipeline.fit(df_train)
prediction = model.transform(df_test)
classification_metrics(prediction)
```

```
Accuracy on test data = 0.722831
```

In [19]:

```
#GBT2

from pyspark.ml.classification import GBTClassifier
gbt = GBTClassifier(labelCol="HOURLYWindDirectionBucketized", maxIter=100)

vectorAssembler = VectorAssembler(inputCols=["HOURLYWindSpeed","HOURLYDRYBULBTEMPC","EL
EVATION","HOURLYStationPressure",
                                             "HOURLYPressureTendency","HOURLYPrecip"],
                                  outputCol="features")

pipeline = Pipeline(stages=[bucketizer,vectorAssembler,normalizer,gbt])
model = pipeline.fit(df_train)
prediction = model.transform(df_test)
classification_metrics(prediction)
```

```
Accuracy on test data = 0.730822
```