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 (basically replace the parts marked with $$) and finally answer a quiz on the Coursera platform.

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/

In [ ]:
```
# delete files from previous runs
!rm -f jfk_weather*

# download the file containing the data in CSV format
!wget http://max-training-data.s3-api.us-geo.objectstorage.softlayer.
net/noaa-weather/jfk_weather.tar.gz

# extract the data
!tar xvfz jfk_weather.tar.gz

# create a dataframe out of it by using the first row as field names a
nd trying to infer a schema based on contents
df = spark.read.option("header", "true").option("inferSchema","true").
csv('jfk_weather.csv')

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

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 [ ]:
```
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('do
uble')) \
    .withColumn("HOURLYStationPressure", translate(col("HOURLYStationP
ressure"), "s,", "")) \
    .withColumn("HOURLYPrecip", translate(col("HOURLYPrecip"), "s,",
"")) \
    .withColumn("HOURLYRelativeHumidity", translate(col("HOURLYRelativ
eHumidity"), "*", "")) \
    .withColumn("HOURLYDRYBULBTEMPC", translate(col("HOURLYDRYBULBTEMP
C"), "*", "")) \

df_cleaned =   df_cleaned \
                    .withColumn("HOURLYStationPressure", df_cleaned.HO
URLYStationPressure.cast('double')) \
                    .withColumn("HOURLYPrecip", df_cleaned.HOURLYPreci
p.cast('double')) \
                    .withColumn("HOURLYRelativeHumidity", df_cleaned.H
OURLYRelativeHumidity.cast('double')) \
                    .withColumn("HOURLYDRYBULBTEMPC", df_cleaned.HOURL
YDRYBULBTEMPC.cast('double')) \

df_filtered = df_cleaned.filter("""
    HOURLYWindSpeed <> 0
    and HOURLYWindDirection <> 0
    and HOURLYStationPressure <> 0
    and HOURLYPressureTendency <> 0
    and HOURLYPressureTendency <> 0
    and HOURLYPrecip <> 0
    and HOURLYRelativeHumidity <> 0
    and HOURLYDRYBULBTEMPC <> 0
""")
```

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

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

As we can see, HOURLYWindSpeed and HOURLYWindDirection correlate with 0.06306013 whereas HOURLYWindSpeed and HOURLYStationPressure correlate with -0.4204518, this is a good sign if we want to predict HOURLYWindSpeed from HOURLYWindDirection and HOURLYStationPressure. Since this is supervised learning, let's split our data into train (80%) and test (20%) set.

In [ ]:
```
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 [ ]:
```
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 [ ]:
```
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 [ ]:
```
#LR1

from pyspark.ml.regression import LinearRegression


lr = LinearRegression(labelCol="HOURLYWindSpeed", featuresCol='feature
s', 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)
```

Now we'll try a Gradient Boosted Tree Regressor

In [ ]:
```
#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)
```

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 [ ]:
```
from pyspark.ml.feature import Bucketizer, OneHotEncoder
bucketizer = Bucketizer(splits=[ 0, 180, float('Inf') ],inputCol="HOUR
LYWindDirection", outputCol="HOURLYWindDirectionBucketized")
encoder = OneHotEncoder(inputCol="HOURLYWindDirectionBucketized", outp
utCol="HOURLYWindDirectionOHE")
```

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 [ ]:
```
def classification_metrics(prediction):
    from pyspark.ml.evaluation import MulticlassClassificationEvaluato
r
    mcEval = MulticlassClassificationEvaluator().setMetricName("accura
cy") .setPredictionCol("prediction").setLabelCol("HOURLYWindDirectionB
ucketized")
    accuracy = mcEval.evaluate(prediction)
    print("Accuracy on test data = %g" % accuracy)
```

Again, for baselining we use LogisticRegression.

In [ ]:
```
#LGReg1

from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(labelCol="HOURLYWindDirectionBucketized", maxI
ter=10)
#,"ELEVATION","HOURLYStationPressure","HOURLYPressureTendency","HOURLY
Precip"

vectorAssembler = VectorAssembler(inputCols=["HOURLYWindSpeed","HOURLY
DRYBULBTEMPC"],
                                  outputCol="features")

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

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 [ ]:

```
#RF1

from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(labelCol="HOURLYWindDirectionBucketized",
numTrees=30)

vectorAssembler = VectorAssembler(inputCols=["HOURLYWindSpeed","HOURLY
DRYBULBTEMPC","ELEVATION","HOURLYStationPressure","HOURLYPressureTende
ncy","HOURLYPrecip"],
                                  outputCol="features")

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

In [ ]:

```
#GBT2

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

vectorAssembler = VectorAssembler(inputCols=["HOURLYWindSpeed","HOURLY
DRYBULBTEMPC","ELEVATION","HOURLYStationPressure","HOURLYPressureTende
ncy","HOURLYPrecip"],
                                  outputCol="features")

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