Taxi

November 29, 2019

1 Apache Spark - Taxi Fare

1.1 Set Up Libraries

There are several libraries imported according to what is needed to build up the models.

[1]:
```
from pyspark import SparkContext
import pyspark as pyspark
from pyspark.sql import SparkSession

from pyspark.ml.feature import VectorAssembler
from pyspark.ml.linalg import Vectors

from pyspark.ml.regression import LinearRegression
from pyspark.sql.types import FloatType
import pandas as pd

from datetime import datetime
```

2 Native Clustering

The APIs for Machine Learning and DataBases purposes will be used. Native Clustering means using libraries that implicitly parallelize the data and fault tolerance. It's important to take it into account because later it's used to solve the same problem but with another way of clustering.

2.1 Set a Spark Session

The purpose of this method is to create an interface between the Jupyter Notebook and the Spark-shell.

[2]:
```
spark = SparkSession.builder.appName('taxis_fare').getOrCreate()
```

2.2 Load Dataset

Loading the data set with the spark method.

[3]:
```
# Load training data
df = spark.read.format("csv").option("header", "true").load("data/train.csv")
```

2.3 Preparing Data

Computing the LinearRegression model

[4]:
```
lr = LinearRegression()
```

It's crutial to see the type of the attributes to check if it's needed to process some categorical attributes, but not in this case.

[5]:
```
df.dtypes
```

[5]:
```
[('key', 'string'),
 ('fare_amount', 'string'),
 ('pickup_datetime', 'string'),
 ('pickup_longitude', 'string'),
 ('pickup_latitude', 'string'),
 ('dropoff_longitude', 'string'),
 ('dropoff_latitude', 'string'),
 ('passenger_count', 'string')]
```

It's necessary to cast all the attributes to float because it's one of the datatypes that Spark works with, because Spark doesn't work with strings.

[6]:
```
df = df.select(df['fare_amount'].cast("float").alias('fare_amount'),
               df['pickup_longitude'].cast("float").alias('pickup_longitude'),
               df['pickup_latitude'].cast("float").alias('pickup_latitude'),
               df['dropoff_longitude'].cast("float").alias('dropoff_longitude'),
               df['dropoff_latitude'].cast("float").alias('dropoff_latitude'),
               df['passenger_count'].cast("float").alias('passenger_count'))
```

Selection of the most important features to work with.

[7]:
```
df = df.selectExpr("fare_amount as label",'pickup_longitude','pickup_latitude',
                   'dropoff_longitude','dropoff_latitude','passenger_count')
```

Checking the correctness of the features, by taking a look at the current data.

[ ]:
```
df.show()
```

Gathering all the features into one colum called "features" because Spark needs to work with all the features in one column.

[9]:
```
#colum features
vecAssembler = VectorAssembler(inputCols=["pickup_longitude", "pickup_latitude",
                                          "dropoff_longitude",␣
 ↪"dropoff_latitude",
                                          "passenger_count"],␣
 ↪outputCol="features")
new_df = vecAssembler.transform(df)
new_df.count()
```

[9]:
```
55423856
```

It's vital to delete null rows because Spark doesn't accept them.

[ ]:
```
#Delete null rows
new_df = vecAssembler.setHandleInvalid("skip").transform(df)
```

```
new_df.show()
```

2.4 Train Model

We use a Linear Regression algorithms to train the model. It's necessary to measure the time the model spends to train the data, which is one of the main purposes of this project.

[11]:
```
# Fit the model
start_time = datetime.now()

lrModel = lr.fit(new_df.select('label','features'))

time_elapsed = datetime.now() - start_time
print('TIME OF LINEAR REGRESSION TRAINING (hh:mm:ss.ms) {}'.
 ↪format(time_elapsed))
```

```
TIME OF LINEAR REGRESSION TRAINING (hh:mm:ss.ms) 0:08:17.698225
```

2.5 Measures

2.5.1 Root Mean Squared Error

[12]:
```
#Summarize the model over the training set and print out some metrics
trainingSummary = lrModel.summary
print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
```

```
RMSE: 20.710237
```

[13]:
```
spark.stop()
```

3 Decision tree regression

Computing Decision Tree with Local Clustering.

[14]:
```
from pyspark.ml import Pipeline
from pyspark.ml.regression import DecisionTreeRegressor
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator

spark = SparkSession.builder.appName('taxis_fare').getOrCreate()

# Load training data
df = spark.read.format("csv").option("header", "true").load("data/train.csv")

df = df.select(df['fare_amount'].cast("float").alias('fare_amount'),
               df['pickup_longitude'].cast("float").alias('pickup_longitude'),
               df['pickup_latitude'].cast("float").alias('pickup_latitude'),
```

```
                df['dropoff_longitude'].cast("float").alias('dropoff_longitude'),
                df['dropoff_latitude'].cast("float").alias('dropoff_latitude'),
                df['passenger_count'].cast("float").alias('passenger_count'))

df = df.selectExpr("fare_amount as label",'pickup_longitude','pickup_latitude',
                   'dropoff_longitude','dropoff_latitude','passenger_count')

new_df = vecAssembler.setHandleInvalid("skip").transform(df)

# Split the data into training and test sets (30% held out for testing)
(trainingData, testData) = new_df.randomSplit([0.7, 0.3])

# Train a DecisionTree model.
dt = DecisionTreeRegressor()

start_time = datetime.now()

# Train model.  This also runs the indexer.
model = dt.fit(trainingData)

time_elapsed = datetime.now() - start_time
print('TIME OF DECISION TREE REGRESSION TRAINING (hh:mm:ss.ms) {}'.
 ↪format(time_elapsed))

# Make predictions.
predictions = model.transform(testData)

# Select example rows to display.
predictions.select("prediction", "label", "features").show(5)

# Select (prediction, true label) and compute test error
evaluator = RegressionEvaluator(
    labelCol="label", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

```
TIME OF DECISION TREE REGRESSION TRAINING (hh:mm:ss.ms) 0:20:05.413627
+------------------+------+--------------------+
|        prediction| label|            features|
+------------------+------+--------------------+
| 34.02801889571103|-29.87|[-73.863159179687...|
|11.850275013996376| -20.0|       (5,[4],[5.0])|
| 8.896463517049114|  -6.5|[-73.984352111816...|
| 8.896463517049114|  -6.0|[-73.987518310546...|
| 8.896463517049114|  -3.0|[-73.995063781738...|
+------------------+------+--------------------+
only showing top 5 rows
```

```
Root Mean Squared Error (RMSE) on test data = 23.9768
```

4 Random forest regression

Computing Random Forest Regression with Local Clustering.

[15]:
```
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator

spark = SparkSession.builder.appName('taxis_fare').getOrCreate()

# Load training data
df = spark.read.format("csv").option("header", "true").load("data/train.csv")

df = df.select(df['fare_amount'].cast("float").alias('fare_amount'),
               df['pickup_longitude'].cast("float").alias('pickup_longitude'),
               df['pickup_latitude'].cast("float").alias('pickup_latitude'),
               df['dropoff_longitude'].cast("float").alias('dropoff_longitude'),
               df['dropoff_latitude'].cast("float").alias('dropoff_latitude'),
               df['passenger_count'].cast("float").alias('passenger_count'))

df = df.selectExpr("fare_amount as label",'pickup_longitude','pickup_latitude',
                   'dropoff_longitude','dropoff_latitude','passenger_count')

new_df = vecAssembler.setHandleInvalid("skip").transform(df)

# Split the data into training and test sets (30% held out for testing)
(trainingData, testData) = new_df.randomSplit([0.7, 0.3])

# Train a RandomForest model.
rf = RandomForestRegressor()

# Train model.
start_time = datetime.now()

model = rf.fit(trainingData)

time_elapsed = datetime.now() - start_time
print('TIME OF RANDOM FOREST TRAINING (hh:mm:ss.ms) {}'.format(time_elapsed))

# Make predictions.
predictions = model.transform(testData)

# Select example rows to display.
predictions.select("prediction", "label", "features").show(5)
```

```
# Select (prediction, true label) and compute test error
evaluator = RegressionEvaluator(
    labelCol="label", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

```
TIME OF RANDOM FOREST TRAINING (hh:mm:ss.ms) 0:35:53.671012
+------------------+------+--------------------+
|        prediction| label|            features|
+------------------+------+--------------------+
|18.195694139170584| -44.9|[-73.871116638183...|
|30.771499981119245|-29.87|[-73.863159179687...|
|13.290360749502483| -18.1|[-73.958274841308...|
| 8.933319739165913| -10.1|[-73.983306884765...|
| 8.933319739165913|  -6.5|[-73.984352111816...|
+------------------+------+--------------------+
only showing top 5 rows

Root Mean Squared Error (RMSE) on test data = 24.8237
```

5 With Local Cluster

Spark implicitly split his own data structures to parallelize them into the clusters, so we set a master to manage the others slaves and then we run as many slaves as logical cores our own computer has.

The local[*] string is a special string that denots you're using a local cluster, which is another way of saying you're running in single-machine mode with several cores working as clusters. The * tells Spark to create as many worker threads as logical cores on your machine.

[16]:
```
spark = SparkSession.builder.master('local[*]').
 ↪appName('taxi_fare_local_cluster').config('spark.driver.memory', '8g').
 ↪getOrCreate()
```

5.0.1 Preparing data

Preparing the dataset as before to be trained in a local cluster.

[17]:
```
# Load training data
df = spark.read.csv('data/train.csv', header=True,
                    inferSchema=True, nullValue=' ')

lr = LinearRegression()

df = df.select(df['fare_amount'].cast("float").alias('fare_amount'),
               df['pickup_longitude'].cast("float").alias('pickup_longitude'),
               df['pickup_latitude'].cast("float").alias('pickup_latitude'),
               df['dropoff_longitude'].cast("float").alias('dropoff_longitude'),
```

```
                df['dropoff_latitude'].cast("float").alias('dropoff_latitude'),
                df['passenger_count'].cast("float").alias('passenger_count'))


df = df.selectExpr("fare_amount as label",'pickup_longitude','pickup_latitude',
                   'dropoff_longitude','dropoff_latitude','passenger_count')

#colum features
vecAssembler = VectorAssembler(inputCols=["pickup_longitude", "pickup_latitude",
                                          "dropoff_longitude",␣
 ↪"dropoff_latitude",
                                          "passenger_count"],␣
 ↪outputCol="features")

#Delete null rows
df_local_clust = vecAssembler.setHandleInvalid("skip").transform(df)
```

5.1 Training Linear Regression

[18]:
```
# Fit the model
start_time = datetime.now()

lrModel = lr.fit(df_local_clust.select('label','features'))

time_elapsed = datetime.now() - start_time
print('TIME OF LINEAR REGRESSION TRAINING (hh:mm:ss.ms) {}'.
 ↪format(time_elapsed))
```

```
TIME OF LINEAR REGRESSION TRAINING (hh:mm:ss.ms) 0:07:36.215578
```

5.2 Measures

5.2.1 Root Mean Squared Error

[19]:
```
#Summarize the model over the training set and print out some metrics
trainingSummary = lrModel.summary
print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
```

```
RMSE: 20.710237
```

6 Decision tree regression local cluster

[20]:
```
from pyspark.mllib.tree import DecisionTree, DecisionTreeModel
from pyspark.mllib.util import MLUtils
```

```
spark = SparkSession.builder.master('local[*]').
 ↪appName('taxi_fare_local_cluster').config('spark.driver.memory', '8g').
 ↪getOrCreate()

# Load training data
df = spark.read.csv('data/train.csv', header=True,
                    inferSchema=True, nullValue=' ')

df = df.select(df['fare_amount'].cast("float").alias('fare_amount'),
               df['pickup_longitude'].cast("float").alias('pickup_longitude'),
               df['pickup_latitude'].cast("float").alias('pickup_latitude'),
               df['dropoff_longitude'].cast("float").alias('dropoff_longitude'),
               df['dropoff_latitude'].cast("float").alias('dropoff_latitude'),
               df['passenger_count'].cast("float").alias('passenger_count'))

df = df.selectExpr("fare_amount as label",'pickup_longitude','pickup_latitude',
                   'dropoff_longitude','dropoff_latitude','passenger_count')

#colum features
vecAssembler = VectorAssembler(inputCols=["pickup_longitude", "pickup_latitude",
                                          "dropoff_longitude",␣
 ↪"dropoff_latitude",
                                          "passenger_count"],␣
 ↪outputCol="features")

#Delete null rows
df_local_clust = vecAssembler.setHandleInvalid("skip").transform(df)

# Split the data into training and test sets (30% held out for testing)
(trainingData, testData) = df_local_clust.randomSplit([0.7, 0.3])

# Train a DecisionTree model.
dt = DecisionTreeRegressor()

start_time = datetime.now()

# Train model.  This also runs the indexer.
model = dt.fit(trainingData.select('label','features'))

time_elapsed = datetime.now() - start_time
print('TIME OF DECISION TREE ON LOCAL CLUSTER TRAINING (hh:mm:ss.ms) {}'.
 ↪format(time_elapsed))

# Make predictions.
predictions = model.transform(testData)

# Select example rows to display.
```

```
predictions.select("prediction", "label", "features").show(5)

# Select (prediction, true label) and compute test error
evaluator = RegressionEvaluator(
    labelCol="label", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

```
TIME OF DECISION TREE ON LOCAL CLUSTER TRAINING (hh:mm:ss.ms) 0:13:05.248696
+------------------+-----+--------------------+
|        prediction|label|            features|
+------------------+-----+--------------------+
|  8.93128770001487|-45.0|[-73.980125427246...|
|13.792550010457832|-18.1|[-73.958274841308...|
|  8.93128770001487|-10.1|[-73.983306884765...|
|  8.93128770001487| -6.5|[-73.984352111816...|
|  8.93128770001487| -5.3|[-73.984802246093...|
+------------------+-----+--------------------+
only showing top 5 rows

Root Mean Squared Error (RMSE) on test data = 24.0373
```

7 Random forest regression local cluster

[21]:
```
from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.mllib.util import MLUtils

spark = SparkSession.builder.master('local[*]').
 ↪appName('taxi_fare_local_cluster').config('spark.driver.memory', '8g').
 ↪getOrCreate()

# Load training data
df = spark.read.csv('data/train.csv', header=True,
                    inferSchema=True, nullValue=' ')

df = df.select(df['fare_amount'].cast("float").alias('fare_amount'),
               df['pickup_longitude'].cast("float").alias('pickup_longitude'),
               df['pickup_latitude'].cast("float").alias('pickup_latitude'),
               df['dropoff_longitude'].cast("float").alias('dropoff_longitude'),
               df['dropoff_latitude'].cast("float").alias('dropoff_latitude'),
               df['passenger_count'].cast("float").alias('passenger_count'))


df = df.selectExpr("fare_amount as label",'pickup_longitude','pickup_latitude',
                   'dropoff_longitude','dropoff_latitude','passenger_count')
```

```
#colum features
vecAssembler = VectorAssembler(inputCols=["pickup_longitude", "pickup_latitude",
                                          "dropoff_longitude",␣
 ↪"dropoff_latitude",
                                          "passenger_count"],␣
 ↪outputCol="features")

#Delete null rows
df_local_clust = vecAssembler.setHandleInvalid("skip").transform(df)

# Split the data into training and test sets (30% held out for testing)
(trainingData, testData) = df_local_clust.randomSplit([0.7, 0.3])

# Train a RandomForest model.
#  Note: Use larger numTrees in practice.
#  Setting featureSubsetStrategy="auto" lets the algorithm choose.
rf = RandomForestRegressor()

start_time = datetime.now()

# Train model.  This also runs the indexer.
model = rf.fit(trainingData.select('label','features'))

time_elapsed = datetime.now() - start_time
print('TIME OF RANDOM FOREST ON LOCAL CLUSTER TRAINING (hh:mm:ss.ms) {}'.
 ↪format(time_elapsed))

# Make predictions.
predictions = model.transform(testData)

# Select example rows to display.
predictions.select("prediction", "label", "features").show(5)

# Select (prediction, true label) and compute test error
evaluator = RegressionEvaluator(
    labelCol="label", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

```
TIME OF RANDOM FOREST ON LOCAL CLUSTER TRAINING (hh:mm:ss.ms) 0:26:28.543485
+------------------+-----+--------------------+
|        prediction|label|            features|
+------------------+-----+--------------------+
| 8.887903540270049|-45.0|[-73.980125427246...|
|13.359459177697568|-18.1|[-73.958274841308...|
| 8.887903540270049|-10.1|[-73.983306884765...|
|11.238500888096038| -4.5|[-74.006141662597...|
```

```
| 8.934682006302216| -3.0|[-74.004997253417...|
+------------------+-----+--------------------+
only showing top 5 rows

Root Mean Squared Error (RMSE) on test data = 6.8643
```

7.1 Conclusion

As we can observe the Root Mean Squared Error is the same because we use the same algorithm in the same dataset, something relevant to be able to compare them with different approaches. As we see the time is 58% faster while working with the cluster.