

Big Data Systems – Spark Lab Sheet:4

Spark MLlib

1. Objective

Students should be able to

- A. Get familiarity with the Spark MLlib module
- B. Get hands-on experience with Machine Learning programme development and execution

This lab sheet provides a quick introduction of using Spark for Machine Learning applications. This exercise will introduce the API through MLlib package for development of classical regression model.

MLlib is Spark's machine learning (ML) library. Its goal is to make practical machine learning scalable and easy. At a high level, it provides tools such as:

- ML Algorithms: common learning algorithms such as classification, regression, clustering, and collaborative filtering
- Featurization: feature extraction, transformation, dimensionality reduction, and selection
- Pipelines: tools for constructing, evaluating, and tuning ML Pipelines
- Persistence: saving and load algorithms, models, and Pipelines
- Utilities: linear algebra, statistics, data handling, etc.

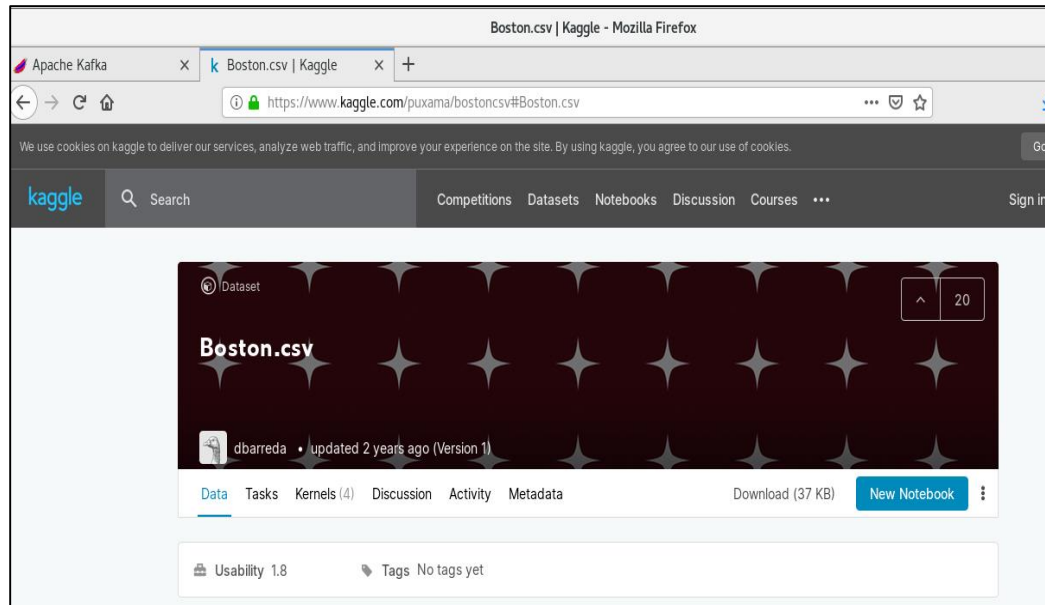
2. Steps to be performed

Preparations -

- a) Open the terminal by clicking on icon on desktop



- b) Using the web browser , download the Boston.csv file and save it in the local file system.
<https://www.kaggle.com/puxama/bostoncsv#Boston.csv>



Installing pySpark

For the execution of python programmes on the Spark, a package named pyspark is required. Using the sudo privileges, install the packages with pip command.

```
pip install pyspark
```

Writing Linear Regression Machine Learning program

- c) Open up the text editor and copy the code written in the attached linear_regression.py file.

```
[centos@master ~]$ gedit linear_regression.py
```

- d) Execute the linear_regression.py file using the spark-submit command.

```
[centos@master ~]$ spark-submit linear_regression.py
```

Look at the outcome printed while the program is getting executed on the Spark cluster. It shows actual and predicted house prices.

prediction	medv	features
27.927270908006726	22.0	[0.01096,55.0,2.2...
30.768510069596182	29.1	[0.01439,60.0,2.9...
26.497762769897413	23.1	[0.0187,85.0,4.15...
40.29308710051661	50.0	[0.02009,95.0,2.6...
27.087495861316143	16.5	[0.02498,0.0,1.89...
28.147245589564633	23.9	[0.02543,55.0,3.7...
22.219573905760136	20.6	[0.03306,0.0,5.19...
32.57075672757935	34.9	[0.03359,75.0,2.9...
20.383617525016447	19.5	[0.03427,0.0,5.19...
30.645810541006668	28.5	[0.03502,80.0,4.9...
27.895366964633475	22.0	[0.03537,34.0,6.0...
24.551339355753125	22.9	[0.03551,25.0,4.8...
28.756201579966294	27.9	[0.03615,80.0,4.9...
23.549490273145608	20.7	[0.03738,0.0,5.19...
35.0073271643155	34.6	[0.03768,80.0,1.5...
36.411263074979686	33.3	[0.04011,80.0,1.5...
26.88971216116669	22.9	[0.04203,28.0,15...
24.69531323293066	20.6	[0.04294,28.0,15...
26.554206975416243	24.8	[0.04297,52.5,5.3...
17.14499460979746	18.2	[0.04301,80.0,1.9...

only showing top 20 rows

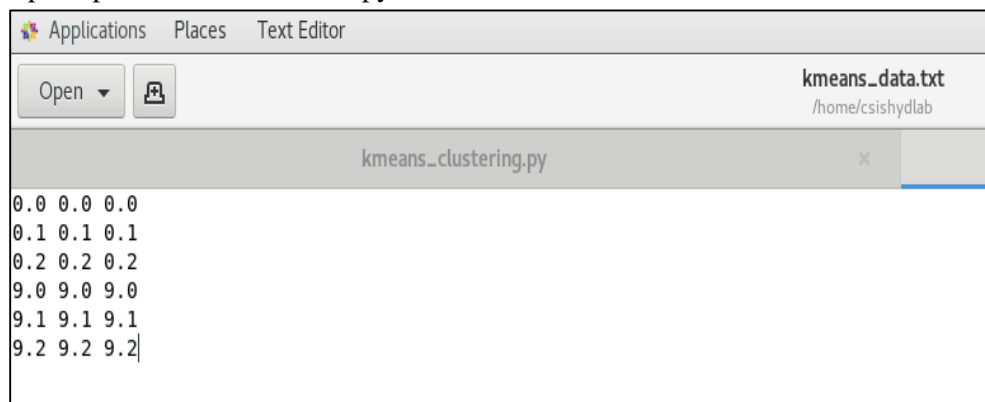
Installing numpy

- e) For the execution of python programmes on the Spark, a package named numpy is required. Using the sudo privileges, install the packages with pip command.

```
pip install numpy
```

Writing k-means clustering Machine Learning program

- f) Open up the text editor and copy the data written in the attached kmeans_data.txt file.



```

Applications  Places  Text Editor
Open  [icon]  kmeans_data.txt
/home/csishydlab

kmeans_clustering.py  x

0.0 0.0 0.0
0.1 0.1 0.1
0.2 0.2 0.2
9.0 9.0 9.0
9.1 9.1 9.1
9.2 9.2 9.2

```

- g) Open up the text editor and copy the code written in the attached kmeans_clustering.py file.

```

kmeans_clustering.py
"""
A K-means clustering program using MLlib.

This example requires NumPy (http://www.numpy.org/).
"""
from __future__ import print_function

import sys

import numpy as np
from pyspark import SparkContext
from pyspark.mllib.clustering import KMeans

def parseVector(line):
    return np.array([float(x) for x in line.split(' ')])

sc = SparkContext(appName="KMeans")
lines = sc.textFile("kmeans_data.txt")
data = lines.map(parseVector)
k = 2
model = KMeans.train(data, k)
print("*****Final centers: " + str(model.clusterCenters))
print("*****Total Cost: " + str(model.computeCost(data)))
sc.stop()

```

- h) Execute the kmeans-clustering.py file using the spark-submit command.

```
[centos@master ~]$ spark-submit kmeans_clustering.py
```

- i) Look at the outcome printed while the program is getting executed on the Spark cluster. It shows the centroids based on the number of clusters mentioned in the python code (k value) .

```

20/01/26 22:40:33 INFO ContextCleaner: Cleaned accumulator 120
20/01/26 22:40:33 INFO MapPartitionsRDD: Removing RDD 3 from persistence list
20/01/26 22:40:33 INFO BlockManagerInfo: Removed broadcast 8 piece0 on apache-spark:34724 in memory (size: 5.5
*****Final centers: [array([0.1, 0.1, 0.1]), array([9.1, 9.1, 9.1])]
20/01/26 22:40:33 INFO BlockManager: Removing RDD 3

```

- j) Repeat the execution by changing the value of k in the python code.

Classification using PySpark-Machine Learning

Given a wine data-set with following attributes

- a) fixed acidity
- b) volatile acidity
- c) citric acid
- d) residual sugar
- e) chlorides
- f) free sulfur dioxide
- g) total sulfur dioxide
- h) density
- i) pH
- j) sulphates
- k) alcohol

The objective is to predict the quality of wine

Steps

- a) Use wine.csv as input file.
- b) Copy-paste the following code in wine-classifier.py file (file also given with lab sheet)

```
import operator
from pyspark import SparkConf, SparkContext
from pyspark.sql import SQLContext
from pyspark.mllib.clustering import KMeans
from numpy import array
from math import sqrt
import matplotlib.pyplot as plt

# data load

datafile = "/user/spark/wine.csv"
sc = SparkContext("local", "wine app")
sqlContext = SQLContext(sc)

df = sqlContext.read.format("csv").options(header="true", inferschema="true").load(datafile)
df.show(n=2)
df.printSchema()

# data prep
from pyspark.ml.feature import VectorAssembler
vectorAssembler = VectorAssembler(inputCols = ["fixed acidity","volatile acidity","citric acid","residual sugar","chlorides",
        "free sulfur dioxide","total sulfur dioxide","density","pH","sulphates","alcohol"], outputCol = 'features')

df2 = vectorAssembler.transform(df)
df2 = df2.select(['features', 'quality'])
df2.show(3)

train, test = df2.randomSplit([0.7, 0.3], seed = 2018)

# train on 70% of the data
from pyspark.ml.classification import DecisionTreeClassifier
dt = DecisionTreeClassifier(featuresCol = 'features', labelCol = 'quality', maxDepth = 3)
dtModel = dt.fit(train)

# test classifier with 30% of the data
predictions = dtModel.transform(test)
predictions.select("features", "quality", "prediction").show(20)
```

- c) Execute wine-classifier.py using spark-submit command and observe the output

```
[centos@master ~]$ spark-submit wine-classifier.py
```

features	quality	prediction
[3.8,0.31,0.02,11...	6	7.0
[4.6,0.445,0.0,1...	5	6.0
[4.7,0.455,0.18,1...	7	7.0
[4.7,0.67,0.09,1...	5	7.0
[4.8,0.21,0.21,10...	7	7.0
[4.8,0.29,0.23,1...	6	7.0
[4.9,0.235,0.27,1...	6	5.0
[5.0,0.17,0.56,1...	7	6.0
[5.0,0.235,0.27,1...	6	5.0
[5.0,0.24,0.19,5...	5	6.0
[5.0,0.24,0.34,1...	7	7.0
[5.0,0.27,0.4,1.2...	6	6.0
[5.0,0.31,0.0,6.4...	6	6.0
[5.0,0.33,0.16,1...	6	6.0
[5.0,0.35,0.25,7...	6	6.0
[5.1,0.26,0.34,6...	6	5.0
[5.1,0.3,0.3,2.3,...	6	7.0
[5.1,0.33,0.22,1...	7	7.0
[5.1,0.33,0.22,1...	7	7.0
[5.1,0.39,0.21,1...	6	7.0

Recommendation System using PySpark-Machine Learning

Given the books and ratings datasets, the objective is to generate recommendations for a user. The Books and Ratings have following attributes.

Books.csv

ISBN	Title	Author	Year	Publisher
------	-------	--------	------	-----------

Ratings.csv

User-ID	ISBN	Rating
---------	------	--------

Copy the following code in recommendation.py

```
from pyspark.sql.types import *

from pyspark import SparkConf, SparkContext
from pyspark.sql import SQLContext
from pyspark.sql import SparkSession
from pyspark.ml.recommendation import ALS
from pyspark.sql.functions import import *
from pyspark.sql.types import import *

# define the configurations for this Spark program
'''
conf = SparkConf().setMaster("local[*]").setAppName("Books")
conf.set("spark.executor.memory", "6G")
conf.set("spark.driver.memory", "2G")
conf.set("spark.executor.cores", "4")
conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
conf.set("spark.default.parallelism", "4")
'''
spark = SparkContext("local", "recommender")
sqlContext = SQLContext(spark)
ss = SparkSession.builder.getOrCreate()
# load data - books
books_df = sqlContext.read.format("csv").options(header="true", inferSchema="true").load("/user/spark/Books.csv")

'''
books_df.show()
books_df.select('Title').show()
books_df.createOrReplaceTempView('Table')
df = sqlContext.sql("SELECT Publisher from Table")
df.show()
'''

# load data - user-ratings
user_ratings_df = sqlContext.read.format("csv").options(header="true", inferSchema="true").load("/user/spark/Ratings.csv")

# Columns User-ID, ISBN and Book-Rating were in string format, which we convert to int
ratings_df = user_ratings_df.withColumn("User-ID", user_ratings_df['User-ID'].\
                                         cast(IntegerType())).\
                             withColumn("ISBN", user_ratings_df['ISBN'].\
                                         cast(IntegerType())).\
                             withColumn("Rating",\
                                         user_ratings_df['Rating'].\
                                         cast(IntegerType())).\
                             na.drop()

ratings_df.show()

# use ALS to build a model
als = ALS(maxIter=5, regParam=0.01, userCol="User-ID", itemCol="ISBN", ratingCol="Rating", coldStartStrategy="drop")
# fit the model to the ratings
dataframemodel = als.fit(ratings_df)

# pick a user and ratings
ratings = ratings_df.filter(col('User-ID')==17)
books_df.join(ratings, ratings.ISBN==books_df.ISBN).select(col('User-ID'), col('Title'), col('Author'), col('Year'), col('Rating')).show()

# now for this user show recommendations

user_id = [17]
# convert this into a dataframe so that it can be passed into the recommendForUserSubset
df = spark.parallelize(user_id).toDF(['User-ID'])
num_rec = 5
recommendations = dataframemodel.recommendForUserSubset(df, num_rec)
recommendations.collect()

# pick only the ISBN of the books, ignore other fields
recommended_ISBN = [recommendations.collect()[0]['recommendations'][x]['ISBN'] for x in range(0,num_rec)]
print(recommended_ISBN)
```

Execute the recommendation.py using spark-submit command and observe the output.

```
[centos@master ~]$ spark-submit recommendation.py
```

Output

Title	Author	Year
Dog Talk: Trainin...	John Ross	1995
River Rising	Merline Lovelace	1999
I'M Not Anti-Busi...	I'M Anti-Idiot-D...	Scott Adams
Bubbly	Jonathan Ray	2001



3. Outputs/Results

Students should be able to

- Execute the linear regression program on Spark cluster
- See the accuracy of the linear regression models in terms of error
- See the predictions done by the model for the test records
- Execute the k-means program on Spark cluster
- Look at the centroid generated by the program with different values of k
- Execute the decision tree classifier on spark cluster
- Look at the class predicted for different test records of wine data-set
- Execute recommendation system on spark cluster
- Look at the recommendations generated for a particular user_id.

4. Observations

Students carefully needs to observe

- Error statistics associated with the linear regression model
- Centroids change when the number of clusters changed
- Accuracy of wine classifier
- Recommendations generated for a particular user
- Interpret the profiling information generated as result of executing different spark programs on spark cluster



References

- A. [Spark ML Guide](#)
- B. [Spark Documentation](#)
- C. [Linear Regression](#)
- D. [K-means clustering](#)