TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

LAB – 04:

Lập trình Spark với Python

Lớp: 19_21

Môn: Nhập môn dữ liệu lớn

Niên khóa: 2021-2022

Mục lục

I. Thông tin sinh viên

1. Thông tin nhóm**:**

Tên nhóm: Gaming House.

Danh sách thành viên:

STT	Họ tên	MSSV
1	Đỗ Thái Duy	19120492
2	Huỳnh Quốc Duy	19120494
3	Phạm Đức Huy	19120534
4	Lê Thành Lộc	19120562

2. Bảng phân công công việc:

MSSV	Họ tên	Công việc
19120492	Đỗ Thái Duy	Viết báo cáo và tham gia code mục 4, 5 của bài 2.
19120494	Huỳnh Quốc Duy	Code bài 1 và quay video demo.
19120534	Phạm Đức Huy	Code bài 3 và quay video demo.
19120562	Lê Thành Lộc	Code mục 1, 2, 3 của bài 2 và quay video demo.

3. Đánh giá mức độ hoàn thành:

STT	Họ tên	MSSV	Mức độ hoàn thành
1	Đỗ Thái Duy	19120492	100%
2	Huỳnh Quốc Duy	19120494	100%
3	Phạm Đức Huy	19120534	100%
4	Lê Thành Lộc	19120562	100%

II. Nội dung tìm hiểu

1. Quá trình thực hiện bài 1 và video demo:

Bước 1: Cài đặt các thư viện cần thiết và khởi động Spark.

```
import pyspark.sql.functions as f

# Thư viện FPGrowth để áp dụng giải thuật khai thác mẫu phổ biến và luật kết hợp
from pyspark.ml.fpm import FPGrowth

import findspark
findspark.init()

import pyspark # only run after findspark.init()
from pyspark.sql import SparkSession

# Tạo SparkSession, chỉ chạy sau khi findspark.init()
spark = SparkSession.builder.getOrCreate()
```

Bước 2: Đọc hai tập tin (orders.csv và products.csv) vào PySpark, lưu dưới dạng 2 Dataframe orders và products để tiện cho tiền xử lý.

```
orders = spark.read.load("../../data/Bai1/orders.csv", format="csv", header=True, delimiter=",")
products = spark.read.load("../../data/Bai1/products.csv", format="csv", header=True, delimiter=",")
```

Bước 3: Thực hiện tiền xử lý.

```
# Dùng lệnh join để nhóm 2 dataframe dựa vào cột product_id của cả 2 dataframe
# Sau đó nhóm theo cột order_id bằng lệnh groupby
# sau đó tạo cột products chứa danh sách các product_name thuộc dòng có order_id tương ứng bằng hàm collect_list của pyspark.sql
# Đưa cột order_id từ chuỗi về loại số thực sau đó sắp xếp cột này từ bé đến lớn (lệnh orderBy) để dễ theo dõi kết quả

df = orders.join(products,
                 products.product_id == orders.product_id,
                 "inner")\
.groupby('order_id').agg(f.collect_list("product_name").alias('products'))\
.withColumn("order_id", orders["order_id"] - 0)\
.orderBy("order_id")
df.show()
print(f'Dataframe kết quả có tổng cộng {df.count()} dòng')
```

Bước 4: Áp dụng giải thuật khai thác mẫu phổ biến và luật kết hợp trong gói pyspark.ml.fpm..

- Thử nghiệm với bộ giá trị ngưỡng support và confidence lần lượt là: (0.2, 0.2).

```
fpGrowth = FPGrowth(itemsCol="products", minSupport=0.2, minConfidence=0.2)

model = fpGrowth.fit(df)
# Tập mẫu phổ biến patternsDF
patternsDF = model.freqItemsets
# Tập luật kết hợp associationRules
rulesDF = model.associationRules

print(f'Có {patternsDF.count()} mẫu phổ biến và {rulesDF.count()} luật kết hợp
đối với bộ giá trị này')

patternsDF.show(patternsDF.count(), False)

rulesDF.show(rulesDF.count(), False)
```

- Thử nghiệm với bộ giá trị ngưỡng support và confidence lần lượt là: (0.1, 0.1).

```
fpGrowth = FPGrowth(itemsCol="products", minSupport=0.1, minConfidence=0.1)

model = fpGrowth.fit(df)
# Tập mẫu phổ biến patternsDF
patternsDF = model.freqItemsets
# Tập luật kết hợp associationRules
rulesDF = model.associationRules

print(f'Có {patternsDF.count()} mẫu phổ biến và {rulesDF.count()} luật kết hợp
đối với bộ giá trị này')

patternsDF.show(patternsDF.count(), False)

rulesDF.show(rulesDF.count(), False)
```

- Thử nghiệm với bộ giá trị ngưỡng support và confidence lần lượt là: (0.01, 0.005).

```
fpGrowth = FPGrowth(itemsCol="products", minSupport=0.01, minConfidence=0.005)
model = fpGrowth.fit(df)
# get the frequent itemsets
patternsDF = model.freqItemsets
# get the association rules
rulesDF = model.associationRules

print(f'Có {patternsDF.count()} mẫu phổ biến và {rulesDF.count()} luật kết hợp
đối với bộ giá trị này')
```

```
patternsDF.show(patternsDF.count(), False)

rulesDF.show(rulesDF.count(), False)
```

- Thử nghiệm với bộ giá trị ngưỡng support và confidence lần lượt là: (0.05, 0.001).

```
fpGrowth = FPGrowth(itemsCol="products", minSupport=0.005,
minConfidence=0.001)
model = fpGrowth.fit(df)
# get the frequent itemsets
patternsDF = model.freqItemsets
# get the association rules
rulesDF = model.associationRules

print(f'Có {patternsDF.count()} mẫu phổ biến và {rulesDF.count()} luật kết hợp
đối với bộ giá trị này')

patternsDF.show(patternsDF.count(), False)

rulesDF.show(rulesDF.count(), False)
```

**** Vấn đề về hình thức của các luật được tìm thấy và cách khắc phục:*

Vấn đề 1:

- Với minsupport và minconfidence lớn, cụ thể là từ 0.2 trở lên.

Vấn đề 2:

- Ta có thể thấy ở phần antecedent và consequent đều chỉ có 1 sản phẩm ở tất cả các luật.

- Để khắc phục điều này, ta có thể giảm minSupport xuống 0.005 và giảm minConfindence xuống ít nhất là 0.001 để ở phần antecedent xuất hiện các luật có từ 2 sản phẩm trở lên như Organic Strawberries, Bag of Organic Bananas -> Organic Hass Avocado hay Organic Hass Avocado, Bag of Organic Bananas -> Organic Strawberries.

***** Link video demo:**

https://drive.google.com/file/d/18iYRox11kfOS0LMEQmnyA1gG7ECkuxjd/view?usp=sharing

2. Quá trình thực hiện bài 2 và video demo:

Bước 1: Cài đặt spark và khởi động **SparkContext**, **SparkSession**.

```
!pip install pyspark

from pyspark import SparkContext
sc = SparkContext(master='local[2]')

# Load Pkgs
from pyspark.sql import SparkSession
# Spark
spark = SparkSession.builder.appName("MLwithSpark").getOrCreate()
```

Bước 2: Tải dữ liệu từ file *mushroom.csv* và cài đặt các module cần thiết.

```
# Load our dataset
df =
spark.read.csv("../../data/Bai2/mushrooms.csv",header=True,inferSchema=True)

# Import modules
from pyspark import SparkContext
from pyspark.sql import SparkSession, Row
from pyspark.ml.feature import VectorAssembler, StringIndexer, OneHotEncoder
from pyspark.ml import Pipeline
from pyspark.ml.classification import DecisionTreeClassifier,
RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

Bước 3: Tiền xử lý các cột trong dataset vừa được load từ file *mushroom.csv*.

```
in_cols = df.schema.names[1:]

# Convert the string into numerical code
# label encoding
for col in in_cols:
  genderEncoder = StringIndexer(inputCol= col, outputCol= col + "
encode").fit(df)
  df = genderEncoder.transform(df)
label_indexer = StringIndexer(inputCol=df.schema.names[0],
outputCol='label').fit(df)

df = label_indexer.transform(df)

df1 = df.select('label', 'cap-shape encode', 'cap-surface encode', 'cap-color
encode', 'bruises encode', 'odor encode', 'gill-attachment encode', 'gill-
spacing encode', 'gill-size encode', 'gill-color encode', 'stalk-shape
encode', 'stalk-root encode', 'stalk-surface-above-ring encode', 'stalk-
surface-below-ring encode', 'stalk-color-above-ring encode', 'stalk-color-
below-ring encode', 'veil-type encode', 'veil-color encode', 'ring-number
```

```
encode', 'ring-type encode', 'spore-print-color encode', 'population encode',
'habitat encode')

# Remove "veil-type encode" column
df1 = df1.drop("veil-type encode")

# VectorAsm
vec_assembler =
VectorAssembler(inputCols=df1.columns[1:],outputCol='features')

main_df = vec_assembler.transform(df1)
main_df = main_df.select(['features', 'label'])
```

Bước 4: Chia dữ liệu thành tập huấn luyện và tập kiểm thử theo tỉ lệ 80:20 qua hàm *randomSplit()*.

```
(df_train, df_val) = main_df.randomSplit([0.8, 0.2], seed=2022)
```

Bước 5: Xây dựng mô hình decision tree trên tập huấn luyện qua hàm *DecisionTreeClassifier()*.

```
dtc = DecisionTreeClassifier(featuresCol="features", labelCol="label")
dtc = dtc.fit(df_train)
pred = dtc.transform(df_val)
pred.show(3)
```

Bước 6: Xây dựng mô hình random forest trên tập huấn luyện.

```
rdc = RandomForestClassifier(featuresCol='features', labelCol='label')
rdc = rdc.fit(df_train)
pred1 = rdc.transform(df_val)
pred1.show(3)
```

Bước 7: Đánh giá hai mô hình trên tập kiểm thử.

- Đánh giá mô hình **Decision Tree**:

```
evaluator=MulticlassClassificationEvaluator(predictionCol="prediction")
acc = evaluator.evaluate(pred)
print("Deciонsion Tree Prediction Accuracy: ", acc)
```

- Đánh giá mô hình **Random Forest**:

```
evaluator1=MulticlassClassificationEvaluator(predictionCol="prediction")
acc1 = evaluator1.evaluate(pred1)
print("Random Forest Prediction Accuracy: ", acc1)
```

Bước 8: Sử dụng pipeline để thiết lập các bước trên thành một luồng xử lý duy nhất.

```
# Get all columns except "class" column
in_cols = df.schema.names[1:]
```

```
# Create a list of StringIndexer objects to convert strings to integer
string_indexers = [StringIndexer(inputCol=col, outputCol=col+'_index') for col
in in_cols]

# Create a list of OneHotEncoder objects to convert integer indices of cat
levels to one-hot encoded columns
onehot_encoders = [OneHotEncoder(dropLast=False, inputCol=col+'_index',
outputCol=col+'_onehot') for col in in_cols]

# Create a VectorAssembler object that assembles all the one-hot encoded
columns into one column
onehot_cols = [col+'_onehot' for col in in_cols]
feat_assembler = VectorAssembler(inputCols=onehot_cols, outputCol='features')

# Create a StringIndexer object that converts "class" column from {e, p} to
{0, 1}
label_indexer = StringIndexer(inputCol=df.schema.names[0], outputCol='label')

# Create a Pipeline object that combines all the transformations we defined
above
pipeline = Pipeline(stages=string_indexers+onehot_encoders+[feat_assembler,
label_indexer])


# Use the pipeline object to transform our dataframe
mushrooms_trans = pipeline.fit(df).transform(df).cache()

# Divide the data into training set and test set in the ratio 80:20
mushrooms_train, mushrooms_val = mushrooms_trans.randomSplit([0.8, 0.2],
seed=2022)

# Build a decision tree model and a random forest model on the training set
dtc_pipeline = DecisionTreeClassifier(featuresCol="features",
labelCol="label").fit(mushrooms_train)
rdc_pipeline = RandomForestClassifier(featuresCol='features',
labelCol='label').fit(mushrooms_train)

pred_pipeline = dtc_pipeline.transform(mushrooms_val)
pred1_pipeline = rdc_pipeline.transform(mushrooms_val)
```

```
# Evaluate two models on test set
# Decision Tree
evaluator_pipeline=MulticlassClassificationEvaluator(predictionCol="prediction
")
acc_dtc = evaluator_pipeline.evaluate(pred_pipeline)

# Random Forest
evaluator1_pipeline=MulticlassClassificationEvaluator(predictionCol="predictio
n")
acc_rdc = evaluator1_pipeline.evaluate(pred1_pipeline)
```

***** Link video demo:**

https://drive.google.com/file/d/1xpZvuKjDqy4IHAbVQHEWrPv1IAWwyZX7/view?usp=sharing

3. Quá trình thực hiện bài 3 và video demo:

Bước 1: Import các thư viện cần thiết (numpy, pandas).

```
import pandas as pd
import numpy as np
```

Bước 2: Cài đặt PySpark và khởi động.

```
!pip install pyspark
!pip install findspark

import findspark
findspark.init

from pyspark.sql import SparkSession # required to created a dataframe
spark=SparkSession.builder.appName("Basics").getOrCreate()
```

Bước 3: Đọc dữ liệu từ file **plants.data**.

```
!ls data

plants_data = spark.read.load("../../Bai3/data/plants.data", format="csv",
header=False, delimiter=",")
```

Bước 4: Tiền xử lý

- Nối các cột trong plants_data lại với nhau:

```
from pyspark.sql.functions import expr, split, explode

df_cau3 = plants_data.withColumn("value", expr(" _c0 || ',' || _c1 || ',' ||
_c2"))
```

- Tạo cột name và cột location.

```
df_cau3 = df_cau3.withColumn("row_ar", split("value", ",")) \
  .select( \
          expr("row_ar[0]").alias("name"),
          expr("trim('[]', string(slice(row_ar, 2, size(row_ar) -
1)))").alias("location")
  )
```

- Tách nội dung ở cột location thành các mảng.

```
df_cau3 = df_cau3.withColumn('list_sa', split("location", ',')) \
  .select( \
      df_cau3.name,
      expr('list_sa')
  )
```

- Sử dụng *explode* để khám phá từng giá trị trong mảng.

```
df_cau3 = df_cau3.select(df_cau3.name, explode(df_cau3.list_sa))
```

- Sử dụng *groupby* và chuyển về dạng pivot để được các cột với giá trị 1.

```
from pyspark.sql.functions import lit

df_cau = df_cau3.withColumn('flag', lit(1))
pivotDF = df_cau.groupBy('name').pivot('col').sum()
pivotDF.show()
```

- Điền các giá trị null bằng giá trị 0.

```
pivotDF = pivotDF.na.fill(value=0)
```

- Lưu dữ liệu vào **plants.csv** (trong thư mục **output**).

```
pivotDF.write.csv("plants_csv")
```

Bước 5: Thực hiện gom cụm bằng **VectorAssembler**.

- Tạo vector đặc trưng từ DataFrame. Để tạo vector đặc trưng (feature vector) cho từng dòng dữ liệu, dùng lớp VectorAssembler để ghép giá trị ở các cột trong DataFrame thành một cột mới chứa vector đặc trưng:

```
from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(inputCols = list(pivotDF.columns)[1:],
                            outputCol="features")
plants_features = assembler.transform(pivotDF)
plants_features.show(5)
```

- Lúc này dữ liệu đã sẵn sàng để đưa vào huấn luyện mô hình ML (ở đây là K-means). Gỉa sử chọn số cụm là 30, tiến hành huấn luyện mô hình K-means với PySpark bằng các lệnh sau:

```
from pyspark.ml.clustering import KMeans

kmeans = KMeans().setK(30).setSeed(0)
model = kmeans.fit(plants_features)
```

- Đánh giá kết quả gom cụm bằng ClusteringEvaluator.

```
predictions = model.transform(plants_features)
from pyspark.ml.evaluation import ClusteringEvaluator
# Evaluate clustering by computing Silhouette score
```

```
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(predictions)
print("Silhouette with squared euclidean distance = " + str(silhouette))
```

Bước 6: Thử nghiệm với một số giá trị k (50, 100, 150, 200).

```
def test_kmeans(data, k):
  kmeans = KMeans().setK(k).setSeed(0)
  model = kmeans.fit(data)

  predictions = model.transform(data)

  #test
  evaluator = ClusteringEvaluator()
  silhouette = evaluator.evaluate(predictions)
  print("Silhouette with squared euclidean distance = " + str(silhouette))

test_kmeans(plants_features, 5)
test_kmeans(plants_features, 100)
test_kmeans(plants_features, 150)
test_kmeans(plants_features, 200)
```

***** Link video demo:**

https://drive.google.com/file/d/1AKCEr3l7mQdLketKdemNWgDewjhzwTi5/view?usp=sharing