HO CHI MINH UNIVERSITY OF SCIENCE

Faculty of Information Technology

Lab 3

Subject: Introduction to Big Data

Topic: Spark

Instructors*:*

- Lecturer Lê Ngọc Thành
- Lecturer Nguyễn Ngọc Thảo
- Lecturer Phạm Trọng Nghĩa

Students in charge: *Team Airavata*

- Huỳnh Cao Nhật Hiếu - 19127399
- Ngô Đăng Khoa - 19127444
- Nguyễn Gia Hân - 19127134
- Trần Thị Huế Minh - 19127476

Table of Content

I. Group information

- Group: *Airavata*
- Members:

Full Name	***Student ID***
Nguyễn Gia Hân	19127134
Huỳnh Cao Nhật Hiếu	19127399
Ngô Đăng Khoa	19127444
Trần Thị Huế Minh	19127476

II. Group's result work

1. Requirement 1

Ở yêu cầu đầu tiên, chúng ta cần phải sử dụng các thư viện machine trong Spark để có thể dự đoán doanh số bán hàng của các cửa hàng trong tương lai.

[1]
```
import findspark
findspark.init()
from pyspark.ml import Pipeline
from pyspark.ml.regression import GBTRegressor
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.sql import functions as f
from pyspark.ml.feature import StringIndexer, VectorIndexer, VectorAssembler
```

Đây là tất cả những thư viện của Spark để sử dụng cho phần này. Do chúng em cài Spark trên máy local nên phải cần sử dụng thêm thư viện findspark để có thể tạo được Spark Session.

Create Spark session

```
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local")\
        .appName("StructureAPI")\
        .config("spark.some.config.option", "some-value")\
        .getOrCreate()
```

Ngoài ra chúng em còn có sử dụng thêm các hàm từ gói sql của pyspark để có thể thao tác dễ dàng hơn. Các thư viện về học máy sẽ được giải thích rõ hơn ở các phần tiếp theo.

a) Khám phá dữ liệu

```
data = spark.read.load("sales_train.csv", format="csv", header=True, delimiter=",")
data.show(5)
```

✓ 10.4s

```
+----------+--------------+-------+-------+----------+------------+
|      date|date_block_num|shop_id|item_id|item_price|item_cnt_day|
+----------+--------------+-------+-------+----------+------------+
|02.01.2013|             0|     59|  22154|     999.0|         1.0|
|03.01.2013|             0|     25|   2552|     899.0|         1.0|
|05.01.2013|             0|     25|   2552|     899.0|        -1.0|
|06.01.2013|             0|     25|   2554|   1709.05|         1.0|
|15.01.2013|             0|     25|   2555|    1099.0|         1.0|
+----------+--------------+-------+-------+----------+------------+
only showing top 5 rows
```

```
data.count()
```

✓ 1.8s

```
2935849
```

Đầu tiên chúng ta sẽ đọc file dữ liệu **sales_train.csv** mà thầy đã cung cấp. Dữ liệu này có 5 cột thuộc tính và 2,935,849 dòng. Mỗi dòng tương ứng với dữ liệu mua bán từng món hàng được cập nhật hằng ngày ở các cửa hàng. Mô tả thêm về các cột trong dữ liệu:

- **date:** Ngày bán hàng theo format dd/mm/yyyy
- **date_num_block:** Số tháng liên tiếp từ tháng 1 năm 2013 đến tháng 10 năm 2015. Nếu là tháng 1/2013 thì sẽ mang giá trị là 0… cứ tiếp tục đến tháng cuối cùng là 10/2015 sẽ mang giá trị 33.
- **shop_id:** Mã của từng cửa hàng
- **item_id:** Mã của từng sản phẩm
- **item_price:** Giá thành mỗi sản phẩm
- **item_cnt_day:** Số lượng hàng hóa bán được và chúng ta sẽ đi dự đoán giá trị này

```
data.printSchema()
```
✓ 0.9s

```
root
 |-- date: string (nullable = true)
 |-- date_block_num: string (nullable = true)
 |-- shop_id: string (nullable = true)
 |-- item_id: string (nullable = true)
 |-- item_price: string (nullable = true)
 |-- item_cnt_day: string (nullable = true)
```

Schema của dữ liệu. Kiểu dữ liệu trong bảng tất cả đều là kiểu string không hợp lý đối với các cột numeric như **item_price, item_cnt_day,...** nên tiếp theo ta sẽ chuyển chúng về các kiểu dữ liệu số.

```
data = data.withColumn('date',f.to_date(data.date,'dd.MM.yyyy'))
data = data.withColumn("date_block_num",data.date_block_num.cast('int'))
data = data.withColumn("item_price",data.item_price.cast('double'))
data = data.withColumn("item_cnt_day",data.item_cnt_day.cast('int'))
```
✓ 0.1s

```
data.printSchema()
```

```
root
 |-- date: date (nullable = true)
 |-- date_block_num: integer (nullable = true)
 |-- shop_id: string (nullable = true)
 |-- item_id: string (nullable = true)
 |-- item_price: double (nullable = true)
 |-- item_cnt_day: integer (nullable = true)
```

Quá trình cast lại kiểu dữ liệu và kết quả sau khi đã chuyển đổi.

```
data.select([f.count(f.when(f.col(c).contains('None') | \
                            f.col(c).contains('NULL') | \
                            (f.col(c) == '' ) | \
                            f.col(c).isNull() | \
                            f.isnan(c), c
                           )).alias(c)
                for c in data.columns]).show()
```
✓ 12.7s

```
+----+--------------+-------+-------+----------+------------+
|date|date_block_num|shop_id|item_id|item_price|item_cnt_day|
+----+--------------+-------+-------+----------+------------+
|   0|             0|      0|      0|         0|           0|
+----+--------------+-------+-------+----------+------------+
```

Tiếp theo ta sẽ kiểm tra xem các cột dữ liệu có bị thiếu hay mất mát hay không.

```
data.summary().show()
```
✓ 14.9s

```
+-------+-----------------+------------------+------------------+------------------+------------------+
|summary|   date_block_num|           shop_id|           item_id|        item_price|      item_cnt_day|
+-------+-----------------+------------------+------------------+------------------+------------------+
|  count|          2935849|           2935849|           2935849|           2935849|           2935849|
|   mean|14.56991146343017|33.001728290521754|10197.227056977385| 890.8532326979881| 1.242640885140891|
| stddev|9.422987708755725| 16.22697304833424|6324.2973538914575|1729.7996307126411|2.6188344308954035|
|    min|                0|                 0|                 0|              -1.0|               -22|
|    25%|                7|              22.0|            4476.0|             249.0|                 1|
|    50%|               14|              31.0|            9346.0|             399.0|                 1|
|    75%|               23|              47.0|           15684.0|             999.0|                 1|
|    max|               33|                 9|              9999|          307980.0|              2169|
+-------+-----------------+------------------+------------------+------------------+------------------+
```

Sử dụng hàm **summary()** để mô tả các thông tin cơ bản của bảng dữ liệu. Ta biết các thông tin về số liệu giá trị, trung bình, max - min và các tứ phân vị của dữ liệu. Qua bảng trên ta cũng thấy được một số điểm giá trị âm nằm ở cột **item_price** và **item_cnt_day.**

- Sau khi mô tả dữ liệu, ta thấy được cột **item_cnt_day** và cột **item_price** có giá trị bất thường nhỏ hơn 0.

```
len(data.where(data.item_cnt_day < 0).collect())
```
✓ 5.3s

```
7356
```

- Số lượng dòng bất thường trên cột **item_cnt_day** là 7356

```
len(data.where(data.item_price < 0).collect())
```
✓ 4.8s

```
1
```

- Số lượng dòng bất thường trên cột **item_price** là 1

Khám phá các cột mang giá trị bất thường ở hai cột trên, ta biết được tổng số lượng dòng mang giá trị âm là 7,357 dòng.

```
data = data.where(data.item_price > 0).where(data.item_cnt_day > 0)
data.summary().show()
```
✓ 10.9s

```
+-------+------------------+------------------+------------------+-----------------+------------------+
|summary|    date_block_num|           shop_id|           item_id|       item_price|      item_cnt_day|
+-------+------------------+------------------+------------------+-----------------+------------------+
|  count|           2928492|           2928492|           2928492|          2928492|           2928492|
|   mean|14.569763209187528|33.002952372757036|10200.281966623095|889.4667512710128|1.2483373695403641|
| stddev| 9.422951383876265|16.225428560235574| 6324.395925384864|1727.498582243899| 2.619586031516712|
|    min|                 0|                 0|                 0|             0.07|                 1|
|    max|                33|                 9|              9999|         307980.0|              2169|
+-------+------------------+------------------+------------------+-----------------+------------------+
```

Do số lượng dòng mang giá trị bất thường có số lượng không đáng kể so với số lượng tổng của dữ liệu nên ta có thể loại bỏ các dòng đó để tiết kiệm chi phí.

```
data = data.withColumn('month',(f.floor(data['date_block_num']/12)+1).cast('int'))
data.show(5)
```

✓ 0.3s

```
+----------+--------------+-------+-------+----------+------------+-----+
|      date|date_block_num|shop_id|item_id|item_price|item_cnt_day|month|
+----------+--------------+-------+-------+----------+------------+-----+
|2013-01-02|             0|     59|  22154|     999.0|           1|    1|
|2013-01-03|             0|     25|   2552|     899.0|           1|    1|
|2013-01-06|             0|     25|   2554|   1709.05|           1|    1|
|2013-01-15|             0|     25|   2555|    1099.0|           1|    1|
|2013-01-10|             0|     25|   2564|     349.0|           1|    1|
+----------+--------------+-------+-------+----------+------------+-----+
only showing top 5 rows
```

Có vẻ như tháng mua hàng ở mọi năm cũng có thể quyết định đến doanh số mua hàng nên ta sẽ thêm cột **month** được tính toán từ cột **date_block_num** để có thể có thêm thuộc tính phục vụ cho việc xây dựng mô hình.

b) Huấn luyện mô hình hồi quy với MLib

```
trainingData = data.where(data.date_block_num <28)
testData = data.where(data.date_block_num>=28)
```

✓ 0.7s

Đầu tiên chúng ta sẽ tách tập huấn luyện theo từng **date_num_block.** Ta sẽ sử dụng 28 tháng đầu tiên trong quá trình thu thập cho tập huấn luyện và số tháng còn lại sẽ được sử dụng cho tập test.

Tiếp theo các đặc trưng huấn luyện sẽ được kết hợp lại thành cột features.

```
assembler = VectorAssembler(
    inputCols=['month','item_price','date_block_num'],
    outputCol='features',
    handleInvalid='keep'
)
```

9] ✓ 0.9s

```
assembler.transform(trainingData).show(5)
```

✓ 0.8s

```
+----------+--------------+-------+-------+----------+------------+-----+-----------------+
|      date|date_block_num|shop_id|item_id|item_price|item_cnt_day|month|         features|
+----------+--------------+-------+-------+----------+------------+-----+-----------------+
|2013-01-02|             0|     59|  22154|     999.0|           1|    1|  [1.0,999.0,0.0]|
|2013-01-03|             0|     25|   2552|     899.0|           1|    1|  [1.0,899.0,0.0]|
|2013-01-06|             0|     25|   2554|   1709.05|           1|    1|[1.0,1709.05,0.0]|
|2013-01-15|             0|     25|   2555|    1099.0|           1|    1| [1.0,1099.0,0.0]|
|2013-01-10|             0|     25|   2564|     349.0|           1|    1|  [1.0,349.0,0.0]|
+----------+--------------+-------+-------+----------+------------+-----+-----------------+
only showing top 5 rows
```

Sử dụng class **VectorAssembler()** để gom những cột numeric lại thành một vector các đặc trưng để huấn luyện.

```
gbt = GBTRegressor(featuresCol="features", labelCol='item_cnt_day', maxIter=10)
# Chain indexer and GBT in a Pipeline
pipeline = Pipeline(stages=[assembler, gbt])
```

✓ 0.1s

```
model = pipeline.fit(trainingData)
```

✓ 1m 54.2s

Chúng ta sẽ sử dụng mô hình hồi quy **Gradient-boosted tree regression** để huấn luyện cho dữ liệu. Gom các bước vecto hóa các đặc trưng dữ liệu và huấn luyện vào trong một pipeline.

```
evaluator = RegressionEvaluator(
    labelCol="item_cnt_day",
    predictionCol="prediction",
    metricName="rmse"
)
train_pred = model.transform(trainingData)
test_pred = model.transform(testData)

test_pred.select("prediction", "item_cnt_day").show(5)

print("RMSE train data = %g" % evaluator.evaluate(train_pred))
print("RMSE test data = %g" % evaluator.evaluate(test_pred))
```

✓ 25.5s

```
+------------------+------------+
|        prediction|item_cnt_day|
+------------------+------------+
| 1.181191524140303|           1|
|1.0293996116334725|           1|
| 1.290853487503798|           1|
| 1.181191524140303|           1|
| 1.181191524140303|           1|
+------------------+------------+
only showing top 5 rows

RMSE train data = 2.0582
RMSE test data = 5.06909
```

Về phần đánh giá mô hình chúng ta sẽ sử dụng class **RegressionEvaluator** từ thư viện evaluation của **pyspark.ml** với phương pháp đánh giá sẽ là Root Mean Square Error (RMSE). Qua kết quả đánh giá, ta thấy được RMSE trên tập test vẫn còn khá lớn, kết quả trên tập train tuy ít hơn nhưng với độ lỗi này vẫn không cho được kết quả tốt được.

c) Kết luận

Bài toán này là quá khó so với việc sử dụng các mô hình huấn luyện cơ bản có trên Spark. Để cải thiện được bài toán này em nghĩ chúng ta có thể thiết kế ra nhiều thuộc tính để mô hình có thể học hoặc sử dụng các mô hình như ARIMA để phân tích tốt hơn.

2. Requirement 2

```
import findspark
findspark.init()
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression, LinearSVC
import pyspark
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.sql import SparkSession
from pyspark.sql import functions as f
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
```

Ở phần này ta sẽ sử dụng các mô hình phân lớp (Classification) nên ngoài sử dụng các thư viện của Spark, chúng em còn sử dụng thêm các phương pháp đánh giá từ thư viện sklearn như **confusion_matrix, precision_score, recall_score.**

a) Dataset 1:

```
data1 = spark.read.load("waterQuality1.csv", format="csv", header=True, delimiter=",")
data1.show(5)
```

```
+---------+-------+-------+------+-------+----------+--------+------+--------+--------+-------+-----+--------+--------+-------+-----------+------+--------+------+-------+-------+
|aluminium|ammonia|arsenic|barium|cadmium|chloramine|chromium|copper|flouride|bacteria|viruses| lead|nitrates|nitrites|mercury|perchlorate|radium|selenium|silver|uranium|is_safe|
+---------+-------+-------+------+-------+----------+--------+------+--------+--------+-------+-----+--------+--------+-------+-----------+------+--------+------+-------+-------+
|     1.65|   9.08|   0.04|  2.85|  0.007|      0.35|    0.83|  0.17|    0.05|     0.2|      0|0.054|   16.08|    1.13|  0.007|      37.75|  6.78|    0.08|  0.34|   0.02|      1|
|     2.32|  21.16|   0.01|  3.31|  0.002|      5.28|    0.68|  0.66|     0.9|    0.65|   0.65|  0.1|    2.01|    1.93|  0.003|      32.26|  3.21|    0.08|  0.27|   0.05|      1|
|     1.01|  14.02|   0.04|  0.58|  0.008|      4.24|    0.53|  0.02|    0.99|    0.05|  0.003|0.078|   14.16|    1.11|  0.006|      50.28|  7.07|    0.07|  0.44|   0.01|      0|
|     1.36|  11.33|   0.04|  2.96|  0.001|      7.23|    0.03|  1.66|    1.08|    0.71|   0.71|0.016|    1.41|    1.29|  0.004|       9.12|  1.72|    0.02|  0.45|   0.05|      1|
|     0.92|  24.33|   0.03|   0.2|  0.006|      2.67|    0.69|  0.57|    0.61|    0.13|  0.001|0.117|    6.74|    1.11|  0.003|       16.9|  2.41|    0.02|  0.06|   0.02|      1|
+---------+-------+-------+------+-------+----------+--------+------+--------+--------+-------+-----+--------+--------+-------+-----------+------+--------+------+-------+-------+
only showing top 5 rows
```

```
data1.count()
```

```
7999
```

- **aluminium** - dangerous if greater than 2.8
- **ammonia** - dangerous if greater than 32.5
- **arsenic** - dangerous if greater than 0.01
- **barium** - dangerous if greater than 2
- **cadmium**- dangerous if greater than 0.005
- **chloramine** - dangerous if greater than 4
- **chromium** - dangerous if greater than 0.1
- **copper** - dangerous if greater than 1.3
- **flouride** - dangerous if greater than 1.5
- **bacteria** - dangerous if greater than 0
- **viruses** - dangerous if greater than 0
- **lead** - dangerous if greater than 0.015
- **nitrates** - dangerous if greater than 10
- **nitrites** - dangerous if greater than 1
- **mercury** - dangerous if greater than 0.002
- **perchlorate** - dangerous if greater than 56
- **radium** - dangerous if greater than 5
- **selenium** - dangerous if greater than 0.5
- **silver**- dangerous if greater than 0.1
- **uranium** - dangerous if greater than 0.3
- **is_safe** - class attribute {0 - not safe, 1 - safe}

```
data1.printSchema()
```

```
root
 |-- aluminium: string (nullable = true)
 |-- ammonia: string (nullable = true)
 |-- arsenic: string (nullable = true)
 |-- barium: string (nullable = true)
 |-- cadmium: string (nullable = true)
 |-- chloramine: string (nullable = true)
 |-- chromium: string (nullable = true)
 |-- copper: string (nullable = true)
 |-- flouride: string (nullable = true)
 |-- bacteria: string (nullable = true)
 |-- viruses: string (nullable = true)
 |-- lead: string (nullable = true)
```

Dữ liệu đầu tiên chúng em sử dụng là về đánh giá chất lượng nước ở file **waterQuality1.csv** được lấy từ Kaggle. Dữ liệu này có 7999 dòng và 21 cột. Với mỗi dòng là mẫu phân tích các chất có trong nước và kết luận rằng nước đó có an toàn hay không. Với mỗi cột sẽ là các chất thu thập được ở trong nước, trong đó cột cuối cùng là **is_safe** sẽ là cột label cho bài toán của chúng ta.

```
data1.select([f.count(f.when(f.col(c).contains('None') | \
                             f.col(c).contains('NULL') | \
                             (f.col(c) == '' ) | \
                             (f.col(c) == '#NUM!' ) | \
                             f.col(c).isNull() | \
                             f.isnan(c), c
                            )).alias(c)
                      for c in data1.columns]).show()
```

```
+---------+-------+-------+------+-------+---------+--------+------+--------+--------+-------+----+--------+--------+-------+-----------+------+--------+------+-------+-------+
|aluminium|ammonia|arsenic|barium|cadmium|chloramine|chromium|copper|flouride|bacteria|viruses|lead|nitrates|nitrites|mercury|perchlorate|radium|selenium|silver|uranium|is_safe|
+---------+-------+-------+------+-------+---------+--------+------+--------+--------+-------+----+--------+--------+-------+-----------+------+--------+------+-------+-------+
|        0|      3|      0|     0|      0|         0|       0|     0|       0|       0|      0|   0|       0|       0|      0|          0|     0|       0|     0|      0|      3|
+---------+-------+-------+------+-------+---------+--------+------+--------+--------+-------+----+--------+--------+-------+-----------+------+--------+------+-------+-------+
```

Kiểm tra các giá trị thiếu ở các cột và dòng, ta thấy được cột **ammonia** và **is_safe** có 3 giá trị bị thiếu.

```
data1 = data1.where(data1.ammonia != '#NUM!').where(data1.is_safe!='#NUM!')
```

Do số lượng đó là không đáng kể nên chúng ta sẽ bỏ đi các dòng đó.

```
for item in data1.columns:
    if item != 'is_safe':
        data1 = data1.withColumn(f'{item}', data1[item].cast('double'))
    else:
        data1 = data1.withColumn(f'{item}', data1[item].cast('int'))
```

Qua schema ở trên, ta thấy được tất cả các kiểu dữ liệu của cột đều là string nên chúng ta sẽ cast lại thành kiểu dữ liệu số phục vụ cho việc tính toán và huấn luyện dữ liệu.

```
for item in data1.columns:
    data1.select(item).describe().show()
```

```
Output exceeds the size limit. Open the full output data in a text editor
+-------+------------------+
|summary|         aluminium|
+-------+------------------+
|  count|              7996|
|   mean|0.6663956978489863|
| stddev|1.2653230979043648|
|    min|               0.0|
|    max|              5.05|
+-------+------------------+
```

```
+-------+------------------+
|summary|           ammonia|
+-------+------------------+
|  count|              7996|
|   mean|14.278211605802943|
| stddev|  8.87893018212118|
|    min|             -0.08|
|    max|             29.84|
+-------+------------------+

+-------+-------------------+
|summary|            arsenic|
+-------+-------------------+
|  count|               7996|
|   mean|0.16147698849422124|
...
|    min|                  0|
|    max|                  1|
+-------+-------------------+
```

Mô tả các thông tin cơ bản ở mỗi cột.

```
data1.groupBy('is_safe').count().show()
```

```
+-------+-----+
|is_safe|count|
+-------+-----+
|      1|  912|
|      0| 7084|
+-------+-----+
```

Số lượng mỗi lớp ở trong cột nhãn **is_safe.**

```
assem = VectorAssembler(inputCols=data1.columns[:-1],outputCol='features',handleInvalid='keep')
```

Tiếp theo ở phần xử lý dữ liệu cho mô hình phân lớp, ta sẽ gom các đặc trưng của dữ liệu thành 1 vector.

train-test split

```
(trainingData1, testData1) = data1.randomSplit([0.8, 0.2], 134)
```

Cuối cùng ta sẽ chia tập train và test theo tỉ lệ 80-20.

b) Dataset2

```
data2 = spark.read.load("gender_classification.csv", format="csv", header=True, delimiter=",")
data2.show(5)
```

```
+---------+-----------------+------------------+---------+---------+---------+------------------------+------+
|long_hair|forehead_width_cm|forehead_height_cm|nose_wide|nose_long|lips_thin|distance_nose_to_lip_long|gender|
+---------+-----------------+------------------+---------+---------+---------+------------------------+------+
|        1|             11.8|               6.1|        1|        0|        1|                       1|  Male|
|        0|               14|               5.4|        0|        0|        1|                       0|Female|
|        0|             11.8|               6.3|        1|        1|        1|                       1|  Male|
|        0|             14.4|               6.1|        0|        1|        1|                       1|  Male|
|        1|             13.5|               5.9|        0|        0|        0|                       0|Female|
+---------+-----------------+------------------+---------+---------+---------+------------------------+------+
only showing top 5 rows
```

```
data2.count()
```

```
5001
```

```
data2.printSchema()
```

```
root
 |-- long_hair: string (nullable = true)
 |-- forehead_width_cm: string (nullable = true)
 |-- forehead_height_cm: string (nullable = true)
 |-- nose_wide: string (nullable = true)
 |-- nose_long: string (nullable = true)
 |-- lips_thin: string (nullable = true)
 |-- distance_nose_to_lip_long: string (nullable = true)
 |-- gender: string (nullable = true)
```

Với tập dữ liệu số hai sẽ là dữ liệu về phân biệt nam và nữ qua số liệu được đo đạc từ cơ thể. Dữ liệu này được đọc ở file **gender_classification.csv,** được lấy từ Kaggle. Dữ liệu này có 5001 dòng và 8 cột. Với mỗi dòng sẽ là số liệu cơ thể đo đạc từ mỗi người. Mỗi dòng sẽ là các đặc trưng đo được từ cơ thể từ tóc dài hay ngắn, độ rộng của trán,... và cột phân lớp label sẽ là cột gender xem coi liệu đó là nam hay nữ.

```
data2.select([f.count(f.when(f.col(c).contains('None') | \
                             f.col(c).contains('NULL') | \
                             (f.col(c) == '' ) | \
                             (f.col(c) == '#NUM!' ) | \
                             f.col(c).isNull() | \
                             f.isnan(c), c
                            )).alias(c)
                      for c in data2.columns]).show()
```

```
+---------+----------------+-----------------+---------+---------+---------+------------------------+------+
|long_hair|forehead_width_cm|forehead_height_cm|nose_wide|nose_long|lips_thin|distance_nose_to_lip_long|gender|
+---------+----------------+-----------------+---------+---------+---------+------------------------+------+
|        0|               0|                0|        0|        0|        0|                       0|     0|
+---------+----------------+-----------------+---------+---------+---------+------------------------+------+
```

Không có giá trị nào bị thiếu trong bảng dữ liệu này.

Đổi kiểu dữ liệu của các cột

```
for item in data2.columns:
    if item != 'gender':
        data2 = data2.withColumn(f'{item}', data2[item].cast('double'))
```

8]

Qua schema ở trên, ta thấy được tất cả các kiểu dữ liệu của cột đều là string nên chúng ta sẽ cast lại thành kiểu dữ liệu số phục vụ cho việc tính toán và huấn luyện dữ liệu.

```
for item in data2.columns:
    data2.select(item).describe().show()
```

1]

Output exceeds the size limit. Open the full output data in a text editor

```
+-------+------------------+
|summary|         long_hair|
+-------+------------------+
|  count|              5001|
|   mean| 0.869626074785043|
| stddev|0.3367480365970515|
|    min|               0.0|
|    max|               1.0|
+-------+------------------+


+-------+------------------+
|summary| forehead_width_cm|
+-------+------------------+
|  count|              5001|
|   mean|13.181483703259353|
| stddev| 1.107128302494507|
|    min|              11.4|
|    max|              15.5|
+-------+------------------+
```

Mô tả các cột trong bảng dữ liệu.

```
data2.groupBy('gender').count().show()
```

```
+------+-----+
|gender|count|
+------+-----+
|Female| 2501|
|  Male| 2500|
+------+-----+
```

Số lượng của mỗi nhãn ở trong bộ phân lớp

```
assem = VectorAssembler(inputCols=data2.columns[:-1],outputCol='features',handleInvalid='keep')
data2 = assem.transform(data2)
```

Tiếp theo ở phần xử lý dữ liệu cho mô hình phân lớp, ta sẽ gom các đặc trưng của dữ liệu thành 1 vector.

```
labelIndexer = StringIndexer(inputCol="gender", outputCol="label").fit(data2)
data2 = labelIndexer.transform(data2)
```

Do cột label của chúng ta là dạng chuỗi ‘Male’ và ‘Female’ nên ta sẽ số hóa chúng.

```
(trainingData2, testData2) = data2.randomSplit([0.8, 0.2], 124)
```

Cuối cùng ta sẽ chia tập train và test theo tỉ lệ 80-20.

c) Logistic Regression

Mô hình phân lớp đầu tiên, nhóm em sử dụng Logistic Regression để phân lớp cho 2 tập dữ liệu trên.

Huấn luyện trên tập dataset1.

```
log_reg1 = LogisticRegression(featuresCol='features', labelCol='is_safe')
```

```
log_pipeline1 = Pipeline(stages=[assem, log_reg1])
log_model1 = log_pipeline1.fit(trainingData1)
```

```
predictions = log_model1.transform(testData1)
predictions.select("prediction", "is_safe", "features").show(5)
```

```
+----------+-------+--------------------+
|prediction|is_safe|            features|
+----------+-------+--------------------+
|       0.0|      0|[0.0,0.75,0.07,0....|
|       0.0|      0|[0.0,1.29,0.07,0....|
|       0.0|      0|[0.0,1.42,0.02,0....|
|       0.0|      0|[0.0,2.09,0.09,2....|
|       0.0|      0|[0.0,2.22,0.03,0....|
+----------+-------+--------------------+
only showing top 5 rows
```

Kết quả huấn luyện trên tập test của dataset1

```
evaluator = MulticlassClassificationEvaluator(
    labelCol="is_safe", predictionCol="prediction", metricName="accuracy")
```

```
accuracy = evaluator.evaluate(predictions)
print("Logistic Regression - Test Accuracy = %g" % (accuracy))
print("Logistic Regression - Test Error = %g" % (1.0 - accuracy))
```

```
Logistic Regression - Test Accuracy = 0.902251
Logistic Regression - Test Error = 0.0977492
```

Độ chính xác trên tập test của dataset 1 khá là cao khoảng 90%.

```
y_true = predictions.select('is_safe').rdd.flatMap(lambda x: x).collect()
y_pred = predictions.select('prediction').rdd.flatMap(lambda x: x).collect()
confusionmatrix = confusion_matrix(y_true, y_pred)
precision = precision_score(y_true, y_pred, average='micro')
recall = recall_score(y_true, y_pred, average='micro')
```

366]

```
print("The Confusion Matrix for Logistic Regression is :\n" + str(confusionmatrix))

print("The precision score for Logistic Regression is: " + str(precision))

print("The recall score for Decision Tree Model is: " + str(recall))
```

```
The Confusion Matrix for Logistic Regression is :
[[1351   26]
 [ 126   52]]
The precision score for Logistic Regression is: 0.9022508038585209
The recall score for Decision Tree Model is: 0.9022508038585209
```

Confusion matrix, recall và precision của tập test dataset1

Huấn luyện trên tập dataset2:

```
log_reg2 = LogisticRegression(featuresCol='features', labelCol='label')
```

```
log_pipeline2 = Pipeline(stages=[log_reg2])
log_model2 = log_pipeline2.fit(trainingData2)
```

prediction

```
predictions = log_model2.transform(testData2)
predictions.select("prediction", "label", "features").show(5)
```

```
+----------+-----+--------------------+
|prediction|label|            features|
+----------+-----+--------------------+
|       0.0|  0.0|(7,[1,2],[11.4,5.4])|
|       0.0|  0.0|(7,[1,2,4],[11.4,...|
|       1.0|  1.0|[0.0,11.5,5.3,1.0...|
|       0.0|  0.0|(7,[1,2,6],[11.5,...|
|       0.0|  0.0|(7,[1,2],[11.5,6.2])|
+----------+-----+--------------------+
only showing top 5 rows
```

```
evaluator = MulticlassClassificationEvaluator(
    labelCol="label", predictionCol="prediction", metricName="accuracy")
```

```
accuracy = evaluator.evaluate(predictions)
print("Logistic Regression - Test Accuracy = %g" % (accuracy))
print("Logistic Regression - Test Error = %g" % (1.0 - accuracy))
```

```
Logistic Regression - Test Accuracy = 0.971292
Logistic Regression - Test Error = 0.0287081
```

```
The Confusion Matrix for Logistic Regression is :
[[521  14]
 [ 16 494]]
The precision score for Logistic Regression is: 0.9712918660287081
The recall score for Decision Tree Model is: 0.9712918660287081
```

Số liệu đánh giá của Logistic Regression trên tập dữ liệu dataset2

d) Linear Support Machine

Tiếp theo chúng ta sẽ tiến hành phân lớp và đánh giá với mô hình Linear Support Machine

Huấn luyện trên tập dataset1

```
svc = LinearSVC(maxIter=10, regParam=0.1, labelCol='is_safe', featuresCol='features')
```

```
svc_pipeline1 = Pipeline(stages=[assem, svc])
svc_model1 = svc_pipeline1.fit(trainingData1)
```

```
predictions = svc_model1.transform(testData1)
predictions.select("prediction", "is_safe", "features").show(5)
```

```
+----------+-------+--------------------+
|prediction|is_safe|            features|
+----------+-------+--------------------+
|       0.0|      0|[0.0,0.75,0.07,0....|
|       0.0|      0|[0.0,1.29,0.07,0....|
|       0.0|      0|[0.0,1.42,0.02,0....|
|       0.0|      0|[0.0,2.09,0.09,2....|
|       0.0|      0|[0.0,2.22,0.03,0....|
+----------+-------+--------------------+
only showing top 5 rows
```

Kết quả dự đoán

```
evaluator = MulticlassClassificationEvaluator(
    labelCol="is_safe", predictionCol="prediction", metricName="accuracy")

accuracy = evaluator.evaluate(predictions)
print("LinearSVC - Test Accuracy = %g" % (accuracy))
print("LinearSVC - Test Error = %g" % (1.0 - accuracy))
```

```
LinearSVC - Test Accuracy = 0.885531
LinearSVC - Test Error = 0.114469
```

```
print("The Confusion Matrix for LinearSVC is :\n" + str(confusionmatrix))

print("The precision score for LinearSVC is: " + str(precision))

print("The recall score for LinearSVC is: " + str(recall))
```

```
The Confusion Matrix for LinearSVC is :
[[1377    0]
 [ 178    0]]
The precision score for LinearSVC is: 0.8855305466237942
The recall score for LinearSVC is: 0.8855305466237942
```

Độ chính xác và các số liệu đánh giá.

III. Self-evaluation

Nhóm chúng em đã cài đặt thành công các thuật toán học máy trên Spark và hoàn thành các yêu cầu của bài tập. Qua tập này chúng em biết được các thao tác xử lý dữ liệu cơ bản và biết được thêm một số hàm trong machine learning mà Spark có thể hỗ trợ. Đặc biệt các hàm của Spark có tốc độ xử lý trên các tập dữ liệu lớn như ở yêu cầu 1 là khá tốt. Các vấn đề chúng em gặp phải khi sử dụng Spark là các thư viện tuy xử lý nhanh nhưng không đa dạng nhiều mô hình nên các bài toán như ở yêu cầu 1 khó mà hoạt động hiệu quả.

IV. References

https://spark.apache.org/docs/latest/ml-classification-regression.html#linear-support-vector-machine

https://www.kaggle.com/datasets/mssmartypants/water-quality

https://www.kaggle.com/datasets/elakiricoder/gender-classification-dataset

https://www.kaggle.com/competitions/competitive-data-science-predict-future-sales/data