

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN
NHẬP MÔN DỮ LIỆU LỚN



BÁO CÁO BÀI TẬP MÔN HỌC
NỘI DUNG: LAB 03 – MACHINE
LEARNING ON SPARK

Giảng viên hướng dẫn

: Huỳnh Lâm Hải Đăng

Lớp

: CQ2022/21

Sinh viên thực hiện

: Trương Tiến Anh- 22120017

Đoàn Minh Cường- 22120043

Đinh Viết Lợi- 22120188

Nguyễn Trần Lợi-22120190

Hồ Chí Minh, ngày 6 tháng 5 năm 2025

MỤC LỤC

MỤC LỤC.....	1
PHẦN 1: CLASSIFICATION WITH LOGISTIC REGRESSION	2
I. Structured API Implementation (High Level).....	2
1. Ý tưởng thực hiện	2
2. Mô tả chi tiết	2
II. MLlib RDD-Based Implementation.....	8
1. Ý tưởng thực hiện.....	8
2. Mô tả chi tiết	8
III. Low-Level Operations.....	13
1. Ý tưởng thực hiện.....	13
2. Mô tả chi tiết	13
IV. So sánh các phương pháp giải quyết	18
PHẦN 2: REGRESSION WITH DECISION TREES	19
I. Structured API Implementation (High Level).....	19
1. Ý tưởng thực hiện.....	19
2. Mô tả chi tiết	19
II. MLlib RDD-Based Implementation.....	24
1. Ý tưởng thực hiện.....	24
III. Low-Level Operations.....	26
1. Ý tưởng thực hiện.....	26
2. Mô tả chi tiết	27
IV. So sánh các phương pháp giải quyết	32
PHẦN 3: BÁO CÁO NHÓM.....	32

PHẦN 1: CLASSIFICATION WITH LOGISTIC REGRESSION

I. Structured API Implementation (High Level)

1. Ý tưởng thực hiện

- Structured API cung cấp lượng lớn các API giúp xử lý dữ liệu có scheme rõ ràng và hầu như không yêu cầu viết rất ít các hàm hỗ trợ hay xử lý thủ công.
- Tập dữ liệu được đưa ra cung cấp các giao dịch bằng thẻ tín dụng tại Châu Âu. Do yêu cầu về bảo mật riêng tư nên tên thuộc tính và giá trị tất cả các thuộc tính ngoại trừ `Thời gian giao dịch` và `Số tiền giao dịch` đều đã được chuẩn hoá và ẩn đi.
- Một vấn đề lớn của tập dữ liệu này là hiện tượng mất cân bằng dữ liệu cực nặng khi mà chỉ có 492 trên 284,807 giao dịch là giao dịch lừa đảo.
- Bài làm sẽ sử dụng Structured API với mục đích sau cùng là xây dựng một mô hình nhằm xác định xem giao dịch nào là lừa đảo. Quy trình sẽ bao gồm: khám phá dữ liệu, tiền xử lý dữ liệu cơ bản, phân chia tập huấn luyện, huấn luyện mô hình, cải thiện mô hình bằng phương pháp Undersampling.
- Việc ghi nhận các giao dịch lừa đảo là rất quan trọng, hơn rất nhiều so với các giao dịch không phải lừa đảo nên mô hình cũng sẽ được xây dựng theo hướng tối ưu nhất các dự đoán giao dịch là lừa đảo.

2. Mô tả chi tiết

- Khởi tạo Session và đọc dữ liệu đầu vào

```
data = spark.read.csv('hdfs://localhost:9000/creditcard.csv', header=True, inferSchema=True)
✓ 10.7s
```

Khám phá dữ liệu

```
data.show(5)
✓ 0.4s
```

Time	V1	V2	V3	V4	V5	V6	V7
0.0	-1.3598071336738	-0.0727811733098497	2.53634673796914	1.37815522427443	-0.338320769942518	0.462387777762292	0.239598554061257
0.0	1.19185711131486	0.26615071205963	0.16648011335321	0.448154078460911	0.0600176492822243	-0.0823608088155687	-0.078802983323113
1.0	-1.35835406159823	-1.34016307473609	1.77320934263119	0.379779593034328	-0.503198133318193	1.80049938079263	0.791460956450422
1.0	-0.966271711572087	-0.185226008082898	1.79299333957872	-0.863291275036453	-0.0103088796030823	1.24720316752486	0.23760893977178
2.0	-1.15823309349523	0.877736754848451	1.548717846511	0.403033933955121	-0.407193377311653	0.0959214624684256	0.592940745385545

Hình 1: Tập dữ liệu

- Kiểm tra missing values

3 Nhập môn dữ liệu lớn

```
data.select([sum(col(c).isNull().cast("int")).alias(c) for c in data.columns]).show()
```

✓ 1.9s

Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Nhận xét sơ bộ:

- Bộ dữ liệu bị lệch khi trong 284807 mẫu chỉ có 492 mẫu được ghi nhận là fraud.
- Bộ dữ liệu không tồn tại missing value.

Hình 2: Kiểm tra missing values

- Chuẩn hoá dữ liệu: Thực hiện chuẩn hoá Z-score đối với 2 dữ liệu chưa được chuẩn hoá là `time` và `Amount`, ta gom 2 thuộc tính lại thành 1 vector để chuẩn hoá sau đó tách ra lại.

Time_scaled	Amount_scaled	V1	V2	V3
-1.9965795183032222	0.24496383331670177	-1.3598071336738	-0.0727811733098497	2.53634673796914
-1.9965795183032222	-0.3424739398649182	1.19185711131486	0.26615071205963	0.16648011335321
-1.9965584604164612	1.16068388755694	-1.35835406159823	-1.34016307473609	1.77320934263119
-1.9965584604164612	0.14053400526590265	-0.966271711572087	-0.185226008082898	1.79299333957872
-1.9965374025297005	-0.07340321138793582	-1.15823309349523	0.877736754848451	1.548717846511

only showing top 5 rows

Hình 3: Chuẩn hoá dữ liệu

- Xoá dữ liệu trùng lặp: dữ liệu bị trùng sẽ ảnh hưởng đến chất lượng của mô hình

```
print("Number of rows in data before drop duplicates: ", final_data.count())
final_data=final_data.drop_duplicates()
print("Number of rows in data after drop duplicates: ", final_data.count())
```

✓ 5.1s

Number of rows in data before drop duplicates: 284807
Number of rows in data after drop duplicates: 283726

Hình 4: Xoá dữ liệu trùng lặp

- Phân chia tập dữ liệu: tập dữ liệu không những chỉ phải được phân chia với tỉ lệ train-test phù hợp mà còn phải đảm bảo số lượng nhãn được phân chia phù hợp với vấn đề mất cân bằng dữ liệu trầm trọng.

4 Nhập môn dữ liệu lớn

```
+-----+-----+
|Class| count|
+-----+-----+
|    1|   381|
|    0|226962|
+-----+-----+

+-----+-----+
|Class|count|
+-----+-----+
|    1|   92|
|    0|56291|
+-----+-----+
```

Hình 5: Kết quả phân chia train-test

- Huấn luyện mô hình cơ bản: sau khi tiền xử lý dữ liệu cơ bản, ta có thể tiến hành học từ tập huấn luyện nhanh chóng khi structured API hỗ trợ các API cho mô hình logistic regression sau khi assemble các thuộc tính thành vector.

```
training_cols= [c for c in train_data.columns if c not in ["Class"]]
assembler= VectorAssembler(inputCols=training_cols, outputCol="features")
assembled_train_data= assembler.transform(train_data).select("features", "Class")
✓ 0.1s

logistic_regression= LogisticRegression(featuresCol='features', labelCol='Class')
lg_model=logistic_regression.fit(assembled_train_data)
✓ 25.8s
```

Hình 6: Huấn luyện mô hình

- Đánh giá mô hình trên tập huấn luyện: Đánh giá mô hình bằng các chỉ số Accuracy, AUC, Precision, Recall và F1-score. Mô hình hiện tại cho thấy hiệu suất đối với lớp 0 cực cao trong khi với lớp 1 thì chỉ đạt mức trung bình, điều này là do việc mất cân bằng dữ liệu.

```
=== Train Evaluation ===
Coefficients: [-0.17187092350616137,0.2321453524126453,0.09587500646599136]
Intercept: -8.635162891978922
Accuracy: 0.9991862516110018
AUC (ROC): 0.9766783834522602
Precision (class 0, 1): [0.9993306353239593, 0.8740458015267175]
Recall (class 0, 1): [0.9998546012107754, 0.6010498687664042]
F1-score (class 0, 1): [0.9995925496043326, 0.7122861586314152]
```

Hình 7: Hiệu suất mô hình trên tập huấn luyện

- Dự đoán và kiểm tra trên tập test: Áp dụng mô hình vừa học được ở trên cho dữ liệu

5 Nhập môn dữ liệu lớn

của tập test để bổ sung thêm cột dự đoán, tính các chỉ số tp,tn,fp,fn để tính các thông số hiệu suất. Tuy nhiên kết quả chưa thể chấp nhận được đối với nhãn 1 nên ta sẽ tiếp tục cải thiện hiệu suất mô hình.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56291
1	0.86	0.66	0.75	92
accuracy			1.00	56383

Hình 8: Chỉ số trên tập test

- Undersampling: áp dụng phương pháp undersampling để giảm kích thước của nhãn 0 xuống để giúp mô hình học tốt hơn, có thể phân biệt rõ ràng nhãn 0 và nhãn 1. Lưu ý chỉ oversampling sau khi split tập train-test. Ta lấy tỉ lệ 1:80 thay vì 1:577 như tập dữ liệu gốc.

```
class_1 = train_data.filter(col('Class') == 1)
class_0 = train_data.filter(col('Class') == 0)

n_class_1 = class_1.count()
class_0_undersampled = class_0.orderBy(rand()).limit(n_class_1*80)
balanced_df = class_1.union(class_0_undersampled)

✓ 3.5s

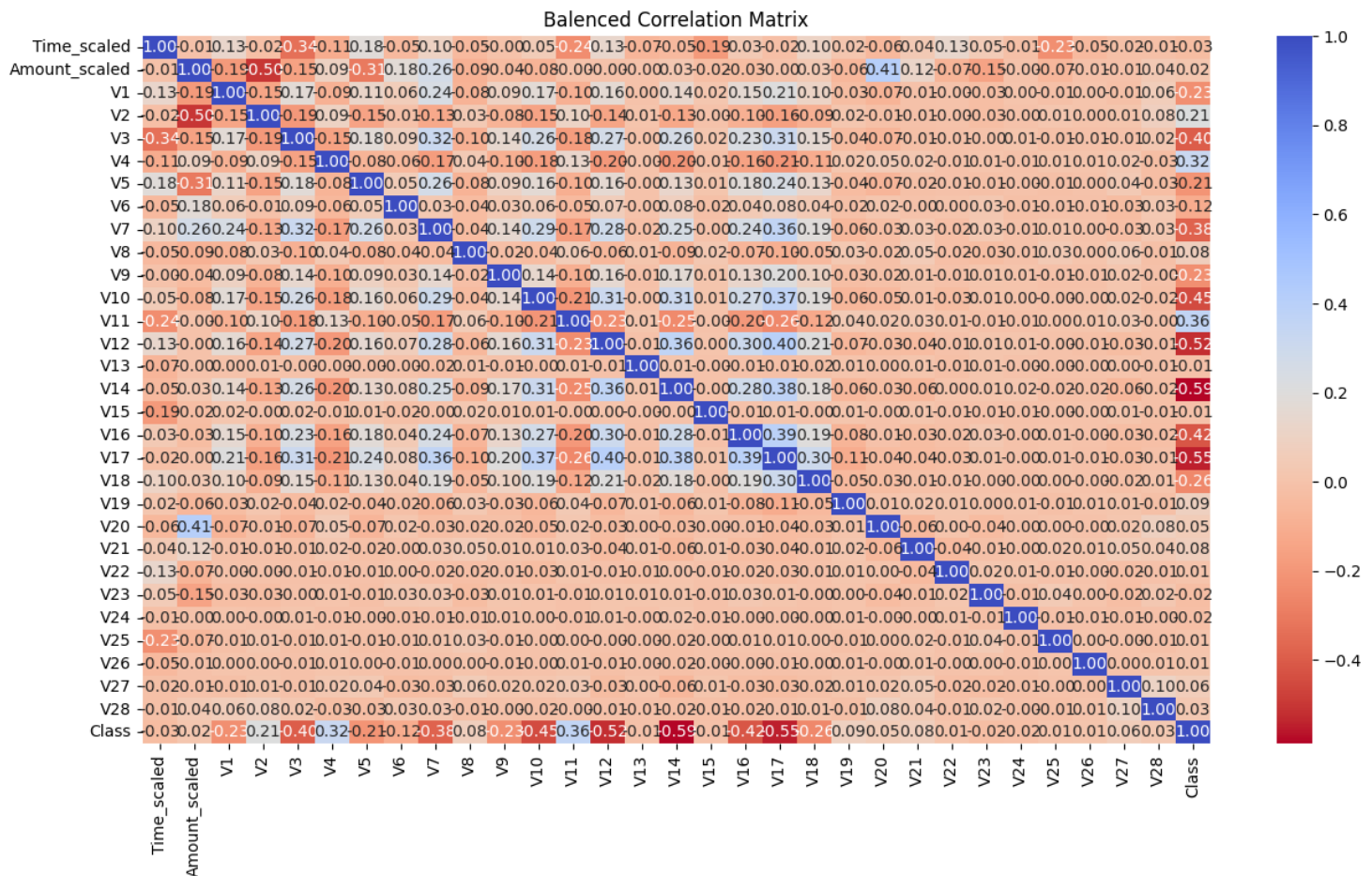
balanced_df.where(balanced_df['Class'] == 1).count(),balanced_df.where(balanced_df['Class'] == 0).count()
✓ 8.7s

(366, 30480)
```

Hình 9: Undersampling

- Vẽ biểu đồ tương quan để thấy kết quả của undersampling ảnh hưởng thế nào đến mô hình

6 Nhập môn dữ liệu lớn



Hình 10: Ma trận tương quan sau Undersampling

- Huấn luyện lại mô hình trên tập Undersampling: các thông số đánh giá đối với nhãn 1 có sự cải thiện đáng kể.

```
=== Train Evaluation ===
Coefficients: [-0.20527754686639102, 0.2711062539973966, 0.1347018711901001, 0.046427097347037205]
Intercept: -7.288117072783255
Accuracy: 0.997438889969526
AUC (ROC): 0.9839882463462558
Precision (class 0, 1): [0.9977405939945643, 0.9674267100977199]
Recall (class 0, 1): [0.9996719160104987, 0.8114754098360656]
F1-score (class 0, 1): [0.9987053212933676, 0.8826151560178305]
```

Hình 11: Hiệu suất mô hình trên tập Undersampling

- Dự đoán và kiểm tra tập test sau khi huấn luyện lại mô hình: Chỉ số Recall của mô hình đã có sự cải thiện đáng kể, giúp việc dự đoán một giao dịch là lừa đảo trở nên chính xác hơn.

7 Nhập môn dữ liệu lớn

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56291
1	0.77	0.82	0.79	92
accuracy			1.00	56383

Hình 12: Chỉ số trên tập test

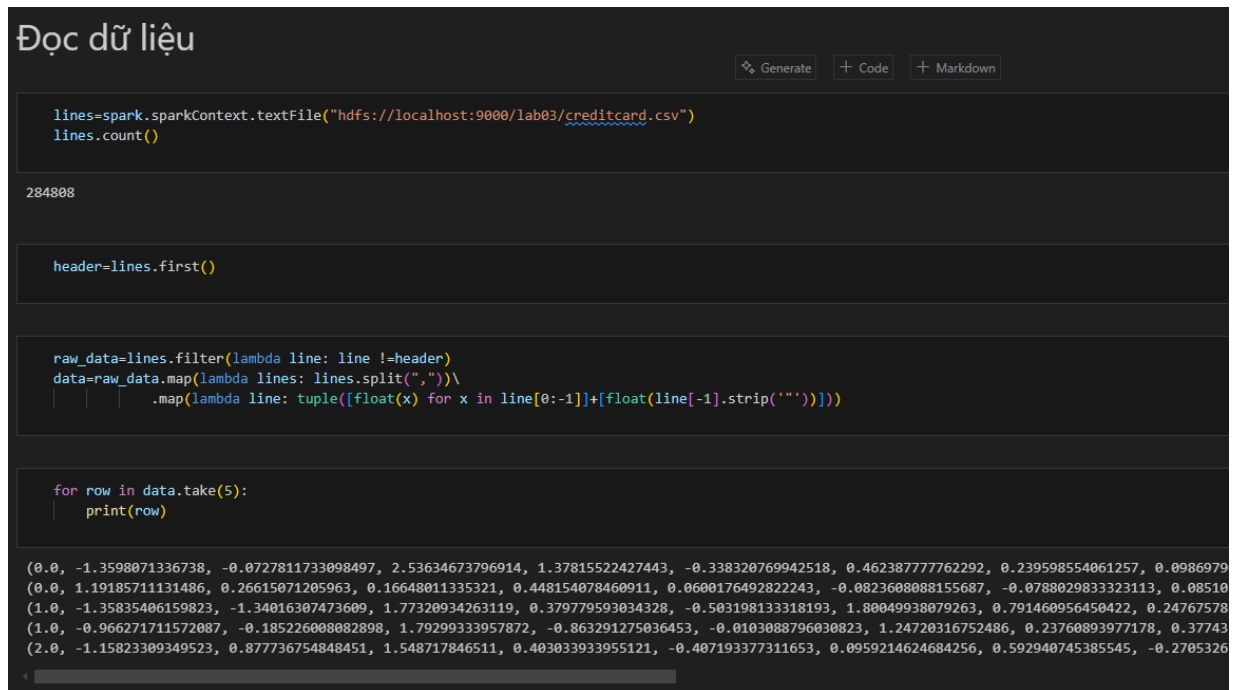
II. MLlib RDD-Based Implementation

1. Ý tưởng thực hiện

- Áp dụng các chức năng tích hợp sẵn của MLlib trong Spark trên dữ liệu RDD (spark.mllib) để thực hiện các tác vụ học máy, qua đó hiểu sâu hơn về cách Spark xử lý RDD.
- Đối với bài toán này, mục đích của việc xây dựng mô hình học máy nhằm phát hiện ra các giao dịch gian lận bằng cách sử dụng thư viện Mllib. Bài làm gồm có quy trình như sau: Tiền xử lý dữ liệu cơ bản, phân chia tập dữ liệu và chuyển thành RDD LabeledPoint, huấn luyện mô hình và chọn tham số tốt nhất, đánh giá mô hình sau khi huấn luyện và sau khi dự đoán.
- Như đã đề cập ở phần Structured API, vấn đề của dữ liệu này là bị mất cân bằng dữ liệu rất nặng.
- Việc ghi nhận các giao dịch lừa đảo là rất quan trọng. Dự báo nhằm còn hơn bỏ sót do đó việc chọn tham số tốt nhất của mô hình sẽ dựa vào chỉ số đánh giá trên lớp “gian lận”.

2. Mô tả chi tiết

- Khởi tạo Spark Session, đọc dữ liệu đầu vào và chuyển thành các dòng dữ liệu phù hợp trong việc xây dựng mô hình.



```
lines=spark.sparkContext.textFile("hdfs://localhost:9000/lab03/creditcard.csv")
lines.count()

284808

header=lines.first()

raw_data=lines.filter(lambda line: line != header)
data=raw_data.map(lambda lines: lines.split(",")\
                  .map(lambda line: tuple([float(x) for x in line[0:-1]]+[float(line[-1].strip("'"))]))

for row in data.take(5):
    print(row)

(0.0, -1.3598071336738, -0.0727811733098497, 2.53634673796914, 1.37815522427443, -0.338320769942518, 0.462387777762292, 0.239598554061257, 0.0986979
(0.0, 1.19185711131486, 0.26615071205963, 0.16648011335321, 0.448154078460911, 0.0600176492822243, -0.0823608088155687, -0.0788029833323113, 0.08510
(1.0, -1.35835406159823, -1.34016307473609, 1.77320934263119, 0.379779593034328, -0.503198133318193, 1.80049938079263, 0.791460956450422, 0.24767578
(1.0, -0.966271711572087, -0.185226008082898, 1.79299333957872, -0.863291275036453, -0.0103088796030823, 1.24720316752486, 0.23760893977178, 0.37743
(2.0, -1.15823309349523, 0.877736754848451, 1.548717846511, 0.403033933955121, -0.407193377311653, 0.0959214624684256, 0.592940745385545, -0.2705326
```

Hình 13. Tập dữ liệu.

- Tiền xử lý dữ liệu gồm các bước sau: Loại bỏ dữ liệu lặp và loại bỏ dữ liệu trống. Sau khi thực hai bước trên. Số dòng dữ liệu giảm xuống còn 283726 dòng.

9 Nhập môn dữ liệu lớn

Tiền xử lý dữ liệu

Loại bỏ dữ liệu lặp

```
data.count()
```

```
[8]
```

```
... 284807
```

```
data=data.distinct()  
data.count()
```

```
[9]
```

```
... 283726
```

Loại bỏ dữ liệu thiếu

```
data = data.filter(lambda line: all(x is not None and x != '' for x in line))
```

```
[10]
```

```
data.count()
```

```
[11]
```

```
... 283726
```

Hình 14. Tiền xử lý dữ liệu.

- Chuẩn hóa dữ liệu theo phương pháp Min-Max: Đưa tất cả về cùng một thang đo, giúp mô hình hội tụ nhanh hơn và tránh việc đặc trưng có giá trị lớn lấn át đặc trưng có giá trị nhỏ.

Chuẩn hóa dữ liệu Min-Max

```
features = data.map(lambda line: line[0:-1])  
  
min_features=features.reduce(lambda x, y: [min(x[i], y[i]) for i in range(len(x))])  
max_features=features.reduce(lambda x, y: [max(x[i], y[i]) for i in range(len(x))])  
  
normalized_data = data.map(lambda line: tuple([(line[i] - min_features[i]) / (max_features[i] - min_features[i]) for i in range(len(line)-1)]+[line[-1]]))  
  
for row in normalized_data.take(5):  
    print(row)
```

```
(0.0, 0.978541954971695, 0.770666508227654, 0.8402984903939014, 0.2717964907547009, 0.7661203363388934, 0.2621916978704357, 0.26487543874149616, 0.7862983529047245, 0.787304967822585e-06, 0.9352170233299468, 0.7531176669488862, 0.8681408192619086, 0.26876550734448534, 0.7623287857209992, 0.28112212055047436, 0.27017718255653134, 0.787304967822585e-06, 0.9418780172089026, 0.7653039594895851, 0.8684836477480651, 0.21366122165460716, 0.7656469003978437, 0.27555923742073796, 0.26680305504220257, 0.1157460993564517e-05, 0.9386168309047992, 0.7765197872285701, 0.8642507014068559, 0.26979635271182784, 0.762975086664976, 0.2639841616818171, 0.26896777555589724, 0.1157460993564517e-05, 0.951057144520383, 0.7773933049100515, 0.857187426639569, 0.24447172439094772, 0.7685503696493547, 0.26272087672843375, 0.2682565838511204, 0.787304967822585e-06)
```

Hình 15. Chuẩn hóa dữ liệu Min-Max.

- Chia dữ liệu train-validation-test theo tỉ lệ như sau: 0.7, 0.15, 0.15. Tỉ lệ này cân bằng giữa với việc huấn luyện, tối ưu và đánh giá. Và việc phân chia còn đảm bảo các tập dữ liệu đều có đủ cả hai lớp.

10 Nhập môn dữ liệu lớn

Tách dữ liệu

```
train_0, val_0, test_0 = normalized_data.filter(lambda x: x[-1] == 0.0).randomSplit([0.7, 0.15, 0.15], seed=42)
train_1, val_1, test_1 = normalized_data.filter(lambda x: x[-1] == 1.0).randomSplit([0.7, 0.15, 0.15], seed=42)
train = train_0.union(train_1)
val = val_0.union(val_1)
test = test_0.union(test_1)
```

Hình 16. Chia tập dữ liệu.

- Chuyển thành dạng RDD LabeledPoint: Tạo một tập dữ liệu huấn luyện thống nhất (đầu vào và đầu ra đi kèm nhau) phù hợp cho mô hình huấn luyện.

Chuyển thành RDD LabeledPoint

```
train_data = train.map(lambda line: LabeledPoint(line[-1], Vectors.dense(line[0:-1])))
val_data = val.map(lambda line: LabeledPoint(line[-1], Vectors.dense(line[0:-1])))
test_data = test.map(lambda line: LabeledPoint(line[-1], Vectors.dense(line[0:-1])))
```

Hình 17. Chuyển thành RDD LabeledPoint.

- Thực hiện huấn luyện mô hình: Mô hình sử dụng là mô hình LogisticRegressionWithLBFGS với các tham số sử dụng là iterations (số vòng lặp tối đa mà thuật toán L-BFGS được phép chạy để hội tụ), regParam (Tham số điều chuẩn, thường là L2), intercept (có bias hay không) và numClasses (số lượng lớp phân loại). Về thang đo, nhóm sử dụng thư viện có sẵn MulticlassMetrics để đánh giá. Sau đó thực hiện Grid Search để tìm ra tham số tốt nhất với tiêu chí đánh giá là Recall của lớp 1 càng lớn càng tốt.

11 Nhập môn dữ liệu lớn

```
iterations=[20,50,100]
regParams=[0, 0.01, 0.1]

best_recall=0.0
best_model=None
best_params=None
best_metrics=None

from pyspark.mllib.evaluation import MulticlassMetrics
for i in iterations:
    for j in regParams:
        print(f"Training with Iterations: {i}, RegParams {j}")
        model = LogisticRegressionWithLBFGS.train(train_data, iterations=i, regParam=j, intercept=True, numClasses=2)

        predictions = val_data.map(lambda p: (float(model.predict(p.features)), p.label))

        metrics = MulticlassMetrics(predictions)

        recall = metrics.recall(1.0)

        if recall > best_recall:
            best_recall = recall
            best_model = model
            best_metrics = metrics
            best_params = (i, j)
        print(f"Iterations: {i}, RegParams {j}, Recall: {recall:.4f}")

print(f"Best Recall: {best_recall:.4f} with Iterations: {best_params[0]}, RegParams {best_params[1]}")
```

Hình 18. Thực hiện huấn luyện mô hình.

```
Training with Iterations: 20, RegParams 0.01
Iterations: 20, RegParams 0.01, Recall: 0.4925
Training with Iterations: 20, RegParams 0.1
Iterations: 20, RegParams 0.1, Recall: 0.2836
Training with Iterations: 50, RegParams 0
Iterations: 50, RegParams 0, Recall: 0.6716
Training with Iterations: 50, RegParams 0.01
Iterations: 50, RegParams 0.01, Recall: 0.4925
Training with Iterations: 50, RegParams 0.1
Iterations: 50, RegParams 0.1, Recall: 0.2836
Training with Iterations: 100, RegParams 0
Iterations: 100, RegParams 0, Recall: 0.6716
Training with Iterations: 100, RegParams 0.01
Iterations: 100, RegParams 0.01, Recall: 0.4925
Training with Iterations: 100, RegParams 0.1
Iterations: 100, RegParams 0.1, Recall: 0.2836
Best Recall: 0.6716 with Iterations: 50, RegParams 0
```

- Sau khi huấn luyện, mô hình tốt nhất có iteration = 50 và regParam = 0

Hình 19. Chọn được mô hình tốt nhất.

- Đánh giá mô hình sau khi huấn luyện: Các chỉ số dùng để đánh giá là Accuracy, Precision, Recall và F1-Score. Mặc dù kết quả của lớp 0 rất tốt nhưng chỉ số của lớp 1 đặc biệt là Recall lại không cao.

```
Test Accuracy: 0.9994
Test Precision [0, 1]: [0.9994826207610178, 0.9375]
Test Recall [0, 1]: [0.9999294167470532, 0.6716417910447762]
Test F1 Score [0, 1]: [0.9997059688326964, 0.7826086956521741]
```

Hình 20. Đánh giá sau khi huấn luyện.

- Thực hiện dự đoán.

12 Nhập môn dữ liệu lớn

Thực hiện dự đoán

```
predictions= test_data.map(lambda p: (float(best_model.predict(p.features)),p.label))
```

Hình 21. Dự đoán.

- Đánh giá mô hình sau khi dự đoán: Các chỉ số đánh giá như sau bước huấn luyện. Kết quả của Recall của lớp 1 lại có chút cải thiện so với sau khi huấn luyện. Tuy nhiên, muốn kết quả tốt hơn, chúng ta cần thực hiện như bổ sung thêm dữ liệu hoặc oversampling lớp 1, undersampling lớp 0.

```
Test Accuracy: 0.9991  
Test Precision [0, 1]: [0.9993167789662158, 0.813953488372093]  
Test Recall [0, 1]: [0.9998114319387154, 0.546875]  
Test F1 Score [0, 1]: [0.9995640442553994, 0.6542056074766355]
```

Hình 22. Đánh giá sau khi dự đoán

- So sánh với Structured API: Hiệu suất tương đương so với Structured API nếu không thực hiện các phương pháp cải thiện hiệu suất.

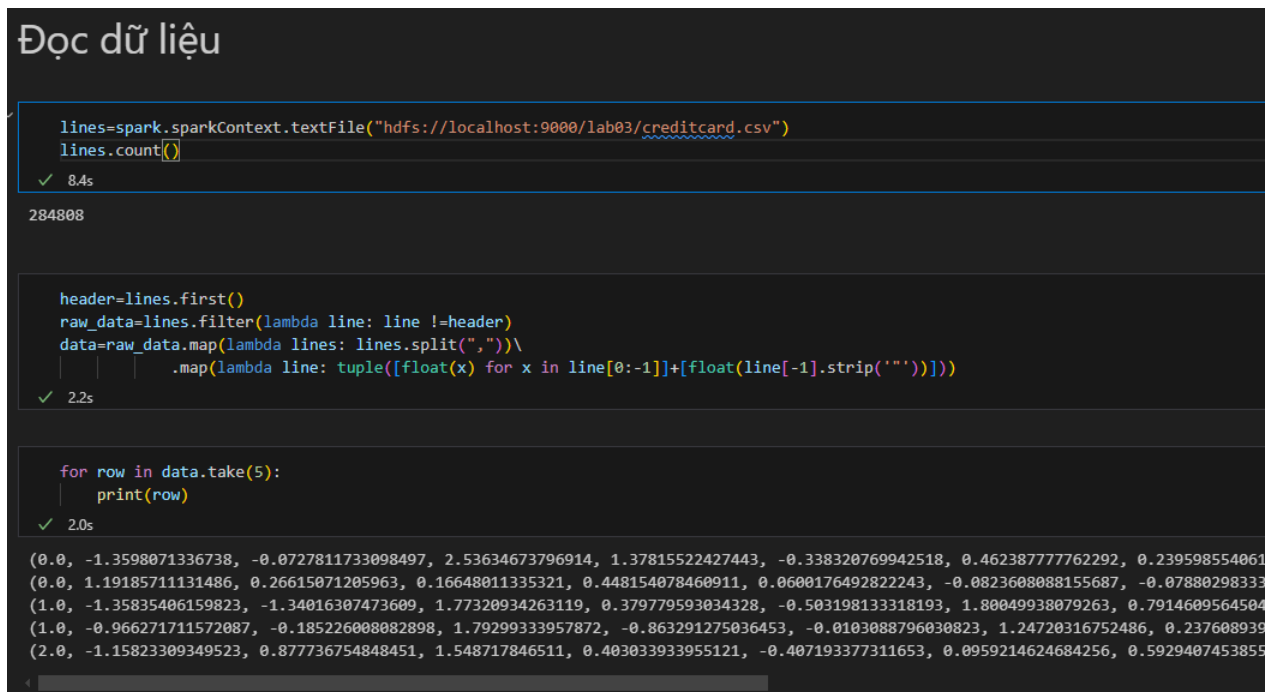
III. Low-Level Operations

1. Ý tưởng thực hiện

- Xây dựng thủ công mô hình Logistics Regression bằng các phép toán RDD cơ bản (mà không dựa vào MLlib), nhằm nâng cao tư duy song song và hiểu biết về tính toán phân tán.
- Đối với bài toán này, mục đích của việc xây dựng mô hình học máy nhằm phát hiện các giao dịch gian lận bằng cách tự xây dựng thủ công mô hình theo RDD mà không sử dụng các thư viện nào khác.
- Quy trình làm bao gồm: tiền xử lý dữ liệu cơ bản, chuẩn hóa dữ liệu, chuyển dữ liệu thành đầu vào thích hợp, phân chia dữ liệu, xây dựng mô hình, huấn luyện mô hình và đánh giá mô hình sau khi huấn luyện và sau khi dự đoán.
- Như đã đề cập ở trên, dữ liệu bị mất cân bằng dữ liệu giữa lớp 0 và lớp 1 do đó việc xây dựng mô hình rất khó đạt được hiệu suất tốt.

2. Mô tả chi tiết

- Khởi tạo Spark Session, đọc dữ liệu đầu vào và chuyển thành các dòng dữ liệu phù hợp trong việc xây dựng mô hình.



```
Đọc dữ liệu

lines=spark.sparkContext.textFile("hdfs://localhost:9000/lab03/creditcard.csv")
lines.count()
✓ 8.4s

284808

header=lines.first()
raw_data=lines.filter(lambda line: line !=header)
data=raw_data.map(lambda line: line.split(",")\
    .map(lambda line: tuple([float(x) for x in line[0:-1]]+[float(line[-1].strip("'"))]))
✓ 2.2s

for row in data.take(5):
    print(row)
✓ 2.0s

(0.0, -1.3598071336738, -0.0727811733098497, 2.53634673796914, 1.37815522427443, -0.338320769942518, 0.462387777762292, 0.239598554061
(0.0, 1.19185711131486, 0.26615071205963, 0.16648011335321, 0.448154078460911, 0.0600176492822243, -0.0823608088155687, -0.07880298333
(1.0, -1.35835406159823, -1.34016307473609, 1.77320934263119, 0.379779593034328, -0.503198133318193, 1.80049938079263, 0.7914609564504
(1.0, -0.966271711572087, -0.185226008082898, 1.79299333957872, -0.863291275036453, -0.0103088796030823, 1.24720316752486, 0.237608939
(2.0, -1.15823309349523, 0.877736754848451, 1.548717846511, 0.403033933955121, -0.407193377311653, 0.0959214624684256, 0.5929407453855
```

Hình 23. Tập dữ liệu.

- Tiền xử lý dữ liệu gồm các bước sau: Loại bỏ dữ liệu lặp và loại bỏ dữ liệu trống. Sau khi thực hai bước trên. Số dòng dữ liệu giảm xuống còn 283726 dòng.

14 Nhập môn dữ liệu lớn

Tiền xử lý dữ liệu

Bỏ dữ liệu lặp

```
data=data.distinct()
data.count()
```

✓ 32.9s

283726

Bỏ dữ liệu trống

```
data = data.filter(lambda line: all(x is not None and x != '' for x in line))
data.count()
```

✓ 8.5s

283726

Hình 24. Tiền xử lý dữ liệu.

- Chuẩn hóa dữ liệu bằng Min-Max: Đưa tất cả về cùng một thang đo, giúp mô hình hội tụ nhanh hơn và tránh việc đặc trưng có giá trị lớn lấn át đặc trưng có giá trị nhỏ.

Chuẩn hóa dữ liệu Min-Max

```
features = data.map(lambda line: line[0:-1])

min_features=features.reduce(lambda x, y: [min(x[i], y[i]) for i in range(len(x))])
max_features=features.reduce(lambda x, y: [max(x[i], y[i]) for i in range(len(x))])

normalized_data = data.map(lambda line: tuple([(line[i] - min_features[i]) / (max_features[i] - min_features[i]) for i in range(len(line)-1)]+[line[-1]]))

for row in normalized_data.take(5):
    print(row)
```

✓ 23.1s

(5.787304967822585e-06, 0.9352170233299468, 0.7531176669488862, 0.8681408192619086, 0.26876550734448534, 0.7623287857209992, 0.28112212055047436, 0.27017718255653134, 0.7
(2.314921987129034e-05, 0.9791841390781553, 0.7687461628975437, 0.8381998380477365, 0.3052410082915901, 0.7670080361177429, 0.26576158850544196, 0.26532408598518387, 0.7
(4.051113477475809e-05, 0.9473484372473652, 0.7822199780865544, 0.8560311087101947, 0.23011142915846172, 0.7721044933924512, 0.26732409614145036, 0.27218252491620354, 0.7
(7.523496458169359e-05, 0.9508713012726631, 0.7769548189300637, 0.8534354682730945, 0.21969318874293933, 0.7718806192614154, 0.2617341715477467, 0.26966654584570826, 0.7
(9.259687948516135e-05, 0.9700990100496973, 0.7528891638518003, 0.8552485617299047, 0.2889077774147477, 0.7576971199759014, 0.2761816416491853, 0.2600030730157834, 0.7901

Hình 25. Chuẩn hóa dữ liệu Min-Max.

- Chuyển dữ liệu thành đầu vào phù hợp: chuyển danh dữ liệu gồm vector đặc trưng và nhãn. Ngoài ra bổ sung thêm bias vào vector đặc trưng.

Chuyển thành đầu vào hợp lý ([features],label) và thêm bias

```
rdd_data= normalized_data.map(lambda line: ([1.0]+[x for x in line[:-1]], line[-1]))

for row in rdd_data.take(5):
    print(row)
```

✓ 1.3s

([1.0, 5.787304967822585e-06, 0.9352170233299468, 0.7531176669488862, 0.8681408192619086, 0.26876550734448534, 0.7623287857209992, 0.28112212055047436, 0.27017718255653134, 0.7
([1.0, 2.314921987129034e-05, 0.9791841390781553, 0.7687461628975437, 0.8381998380477365, 0.3052410082915901, 0.7670080361177429, 0.26576158850544196, 0.26532408598518387, 0.7
([1.0, 4.051113477475809e-05, 0.9473484372473652, 0.7822199780865544, 0.8560311087101947, 0.23011142915846172, 0.7721044933924512, 0.26732409614145036, 0.27218252491620354, 0.7
([1.0, 7.523496458169359e-05, 0.9508713012726631, 0.7769548189300637, 0.8534354682730945, 0.21969318874293933, 0.7718806192614154, 0.2617341715477467, 0.26966654584570826, 0.7
([1.0, 9.259687948516135e-05, 0.9700990100496973, 0.7528891638518003, 0.8552485617299047, 0.2889077774147477, 0.7576971199759014, 0.2761816416491853, 0.2600030730157834, 0.7901

Hình 26. Chuyển thành đầu vào hợp lý và thêm bias.

15 Nhập môn dữ liệu lớn

- Chia dữ liệu thành train-test với tỉ lệ 0.8 và 0.2. Đây là một cách chia phổ biến với 80% dữ liệu huấn luyện cung cấp đủ mẫu để mô hình học được quy luật và mối quan hệ giữa đặc trưng và nhãn. Ngoài ra chia dữ liệu sao cho dữ liệu train và test đều có chứa 2 nhãn là 0 và 1.

Tách dữ liệu

```
train_0,test_0= rdd_data.filter(lambda line: line[-1] == 0.0).randomSplit([0.8, 0.2], seed=42)
train_1,test_1= rdd_data.filter(lambda line: line[-1] == 1.0).randomSplit([0.8, 0.2], seed=42)

train_data=train_0.union(train_1)
test_data=test_0.union(test_1)
```

✓ 0.0s

Hình 27. Chia dữ liệu

- Xây dựng các hàm có trong mô hình Logistics Regression. Đầu tiên là hàm sigmoid.

Hàm sigmoid

```
def sigmoid(x):
    return 1 / (1 + math.exp(-x))
```

✓ 0.0s

Hình 28. Hàm sigmoid

- Hàm dot với chức năng nhân trọng số với một dòng dữ liệu.

Hàm nhân trọng số với một dòng dữ liệu

```
def dot(weights, features):
    return sum(w*x for w, x in zip(weights, features))
```

✓ 0.0s

Hình 29. Hàm nhân trọng số với một dòng dữ liệu

- Hàm tính gradient với một dòng dữ liệu.

Hàm tính gradient với một dòng dữ liệu

```
def gradient(weights, features, label):
    prediction = sigmoid(dot(weights, features))
    error = prediction - label
    return [error * x for x in features]
```

✓ 0.0s

Hình 30. Hàm tính gradient với một dòng dữ liệu.

16 Nhập môn dữ liệu lớn

- Hàm train mô hình với tham số *num_iterations*, *learning_rate*: Khởi tạo bộ trọng số với giá trị 0. Thực hiện gradient descent cho tới khi đủ *num_iterations* lần với tốc độ học *learning_rate*. Kết quả trả về bộ trọng số *weights*.

Hàm train mô hình

```
def train_logistic_regression(data, num_iterations, learning_rate):
    weights = [0.0] * (len(data.first()[0])+1)
    count = data.count()
    for _ in range(num_iterations):
        print("Iteration: ", _ + 1)
        gradients = data.map(lambda x: gradient(weights, x[0], x[1])).reduce(lambda x, y: [a+b for a, b in zip(x, y)])

        gradients = [g / count for g in gradients]

        weights = [w - learning_rate * g for w, g in zip(weights, gradients)]

    return weights
```

✓ 0.0s

Hình 31. Hàm thực hiện huấn luyện.

- Hàm dự đoán với tham số là weights (bộ trọng số) và features (1 dòng dữ liệu).

Hàm dự đoán

```
def predict(weights, features):
    return 1 if sigmoid(dot(weights, features)) >= 0.5 else 0
```

✓ 0.0s

Hình 32. Hàm thực hiện dự đoán.

- Hàm evaluate_metric có chức năng tính các giá trị accuracy, precision, recall, f1_score của một nhãn cụ thể.

```
def evaluate_metric(data, weights, label):
    predictions = data.map(lambda x: (predict(weights, x[0]), x[1]))
    tp = predictions.filter(lambda x: x[0] == label and x[1] == label).count()
    tn = predictions.filter(lambda x: x[0] != label and x[1] != label).count()
    fp = predictions.filter(lambda x: x[0] == label and x[1] != label).count()
    fn = predictions.filter(lambda x: x[0] != label and x[1] == label).count()

    accuracy = (tp + tn) / (tp + tn + fp + fn) if (tp + tn + fp + fn) > 0 else 0
    precision = tp / (tp + fp) if (tp + fp) > 0 else 0
    recall = tp / (tp + fn) if (tp + fn) > 0 else 0

    f1_score = (2 * precision * recall) / (precision + recall) if (precision + recall) > 0 else 0

    return accuracy, precision, recall, f1_score
```

Hình 33. Hàm thực hiện tính các metric.

- Hàm evaluate_model có chức năng đánh giá mô hình. Thực hiện bằng cách chạy hàm evaluate_metric cho hai nhãn của mô hình và trả về kết quả cho từng nhãn.

17 Nhập môn dữ liệu lớn

```
def evaluate_model(data, weights):  
  
    accuracy_0, precision_0, recall_0, f1_score_0 = evaluate_metric(data, weights, 0.0)  
    accuracy_1, precision_1, recall_1, f1_score_1 = evaluate_metric(data, weights, 1.0)  
  
    return (accuracy_0, precision_0, recall_0, f1_score_0), (accuracy_1, precision_1, recall_1, f1_score_1)
```

Hình 34. Hàm thực hiện đánh giá mô hình.

- Thực hiện huấn luyện mô hình với tham số *num_iterations* bằng 100 và *learning_rate* bằng 0.5. Sau khi thử với nhiều tham số khác nhau song vẫn không đạt được hiệu suất tốt. Đây là khó khăn trong việc huấn luyện. Không xác định được tham số phù hợp cho mô hình để có hiệu suất tốt.

Thực hiện huấn luyện mô hình

```
weights= train_logistic_regression(train_data, num_iterations=100, learning_rate=0.5)
```

✓ 45m 54.9s

Hình 35. Thực hiện huấn luyện mô hình.

- Đánh giá mô hình sau khi huấn luyện và đánh giá sau khi dự đoán. Như đã thấy, kết quả hoàn toàn nghiêng về bên lớp 0 và không hoàn toàn có lớp 1 nào được dự đoán.

```
Accuracy: 0.9982935829041636  
Precision [0,1]: [0.9982935829041636, 0]  
Recall [0,1]: [1.0, 0.0]  
F1 Score [0,1]: [0.9991460628656195, 0]
```

Hình 36. Đánh giá sau khi huấn luyện.

```
Accuracy: 0.9984915437718505  
Precision [0,1]: [0.9984915437718505, 0]  
Recall [0,1]: [1.0, 0.0]  
F1 Score [0,1]: [0.9992452025965031, 0]
```

Hình 37. Đánh giá sau khi dự đoán.

- So sánh kết quả với hai phương pháp Structured API và RDD-Based: Kết quả từ mô hình Logistic Regression triển khai bằng RDD cấp thấp cho thấy hiệu suất thấp hơn so với hai phương pháp trên, đặc biệt là với lớp gian lận (lớp 1). Hầu hết các điểm dữ liệu đều bị dự đoán về lớp 0, cho thấy mô hình không học được đủ thông tin phân biệt.
- Các thử thách gặp phải trong quá trình làm: Khó xác định được learning rate và num iterations phù hợp. Mô hình không học được lớp thiểu số do chênh lệch nhãn, dẫn đến mất cân bằng nghiêm trọng trong kết quả dự đoán.

IV. So sánh các phương pháp giải quyết

	Ưu điểm	Nhược điểm
Structured API-High Level	Hiệu suất cao nhờ Catalyst Optimizer & Tungsten Engine. Dễ viết, dễ đọc khi cú pháp tương tự SQL. Hỗ trợ nhiều API có sẵn, tránh xử lý thủ công. Dễ dàng build các pipeline trong machine learning	Chỉ hỗ trợ xử lý dữ liệu có cấu trúc, có scheme rõ ràng.
MLLib RDD-Based	Kiểm soát tốt hơn trên RDD. Mô hình tối ưu có sẵn. Có đánh giá mô hình chi tiết.	Ít linh hoạt hơn Structured API.
Low-Level Operations	Giúp hiểu sâu hơn về phép toán RDD. Kiểm soát toàn bộ quá trình tính toán. Phù hợp để học và nghiên cứu.	Khó triển khai, dễ sai sót. Hiệu suất kém nếu không tinh chỉnh kỹ.

PHẦN 2: REGRESSION WITH DECISION TREES

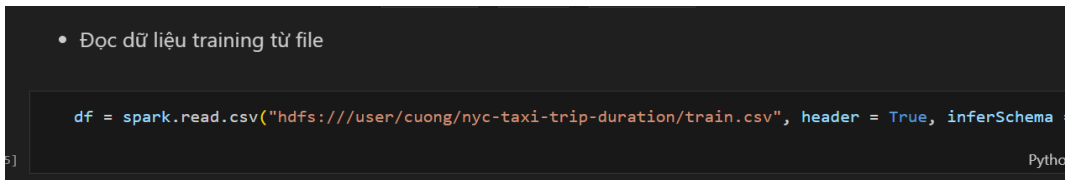
I. Structured API Implementation (High Level)

1. Ý tưởng thực hiện

- Spark Structured API (spark.ml) cung cấp các công cụ cấp cao để xây dựng pipeline học máy một cách dễ dàng và có khả năng mở rộng tốt trên dữ liệu lớn. Việc sử dụng Structured API giúp đơn giản hóa quy trình tiền xử lý và huấn luyện mô hình.
- Bài toán yêu cầu dự đoán một biến liên tục (continuous variable), do đó đây là một bài toán hồi quy (regression). Việc sử dụng Decision Tree Regressor sẽ cho phép mô hình học được các mối quan hệ phi tuyến (non-linear) giữa các đặc trưng và đầu ra.
- Cây quyết định (Decision Tree) là một mô hình dễ hiểu, có thể trực quan hóa, và có khả năng xử lý tốt các đặc trưng dạng số và không yêu cầu chuẩn hóa dữ liệu đầu vào.
- Quy trình sẽ bao gồm các bước: tiền xử lý dữ liệu (bao gồm xử lý dữ liệu thiếu, kết hợp đặc trưng bằng VectorAssembler), huấn luyện mô hình hồi quy cây quyết định, trích xuất cấu trúc cây, và cuối cùng là đánh giá mô hình bằng các chỉ số như RMSE (Root Mean Squared Error) và R^2 (R-squared) trên tập kiểm tra.
- Việc triển khai bài toán hồi quy trên dữ liệu lớn bằng Decision Tree giúp thể hiện khả năng xử lý và huấn luyện mô hình song song của Spark, đồng thời cung cấp hiểu biết trực quan về cách các đặc trưng ảnh hưởng đến kết quả dự đoán.

2. Mô tả chi tiết

- Khởi tạo Spark Session và đọc dữ liệu



```
df = spark.read.csv("hdfs:///user/cuong/nyc-taxi-trip-duration/train.csv", header = True, inferSchema = True)
```

Hình 38. Đọc file từ HDFS.

20 Nhập môn dữ liệu lớn

```
df.printSchema()

root
 |-- id: string (nullable = true)
 |-- vendor_id: integer (nullable = true)
 |-- pickup_datetime: timestamp (nullable = true)
 |-- dropoff_datetime: timestamp (nullable = true)
 |-- passenger_count: integer (nullable = true)
 |-- pickup_longitude: double (nullable = true)
 |-- pickup_latitude: double (nullable = true)
 |-- dropoff_longitude: double (nullable = true)
 |-- dropoff_latitude: double (nullable = true)
 |-- store_and_fwd_flag: string (nullable = true)
 |-- trip_duration: integer (nullable = true)
```

Hình 39. Schema của dữ liệu train

- Kiểm tra dữ liệu bị thiếu

```
• Kiểm tra giá trị null của dữ liệu

> null_counts = df.select([sum.when(col(c).isNull(), 1).otherwise(0).alias(c) for c in df.columns])
  null_counts.show()

[8]

... [Stage 6:=====] (1 + 7) < 8]
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id|vendor_id|pickup_datetime|dropoff_datetime|passenger_count|pickup_longitude|pickup_latitude|dropoff_longitude|dropoff_latitude|store_and_fwd_flag|trip_duration|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 0|      0|           0|           0|           0|           0|           0|           0|           0|           0|           0|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

Hình 40. Xử lý dữ liệu thiếu

21 Nhập môn dữ liệu lớn

- Xử lý dữ liệu bị trùng

```
• Xóa những dòng duplicate

df = df.dropDuplicates()
df.count()

1458644

df.count()

1458644
```

Hình 41. Xóa các dòng trùng lặp

- Tạo assembler các thuộc tính số để có thể đưa vào mô hình

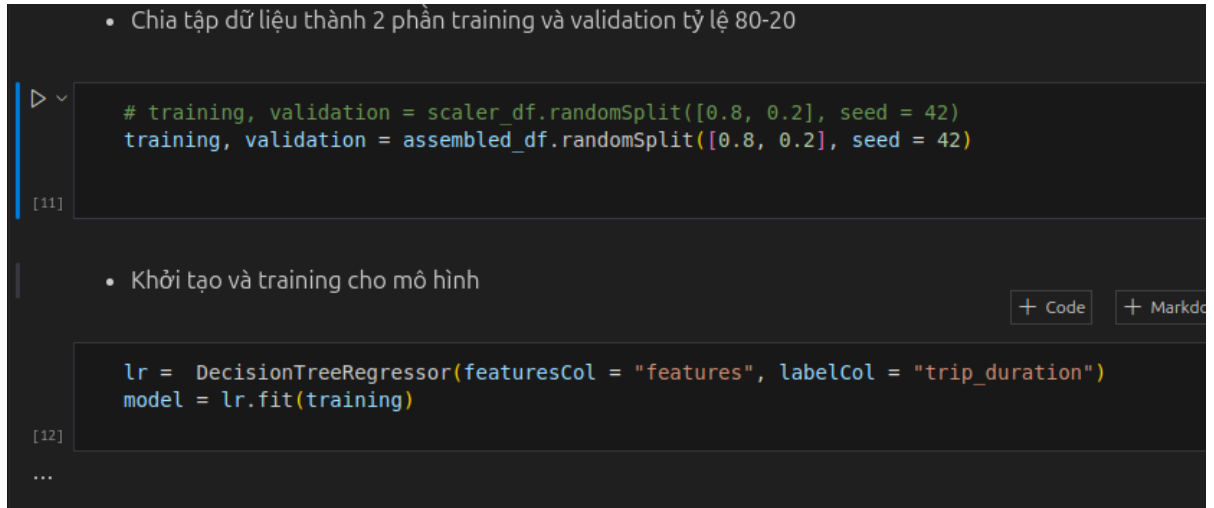
```
assembled_df.show()

[Stage 15:=====]
+-----+-----+
|          features|trip_duration|
+-----+-----+
| [-73.976951599121...|          721|
| [-73.992500305175...|         1216|
| [-73.952377319335...|          709|
| [-73.996566772460...|          952|
| [-73.956291198730...|          430|
| [-73.987808227539...|          264|
| [-74.007629394531...|         1267|
| [-73.987991333007...|          610|
| [-73.988655090332...|          586|
| [-73.989646911621...|          250|
| [-73.974662780761...|          978|
| [-73.985511779785...|          867|
| [-73.973884582519...|         1266|
| [-73.870849609375...|         2129|
| [-73.990516662597...|          425|
| [-73.982238769531...|          983|
| [-73.985633850097...|         1511|
| [-73.986000061035...|         1358|
| [-73.952537536621...|          462|
| [-73.984542846679...|          816|
+-----+-----+
```

Hình 42. Assembler dữ liệu đầu vào

22 Nhập môn dữ liệu lớn

- Chia tập dữ liệu và training mô hình

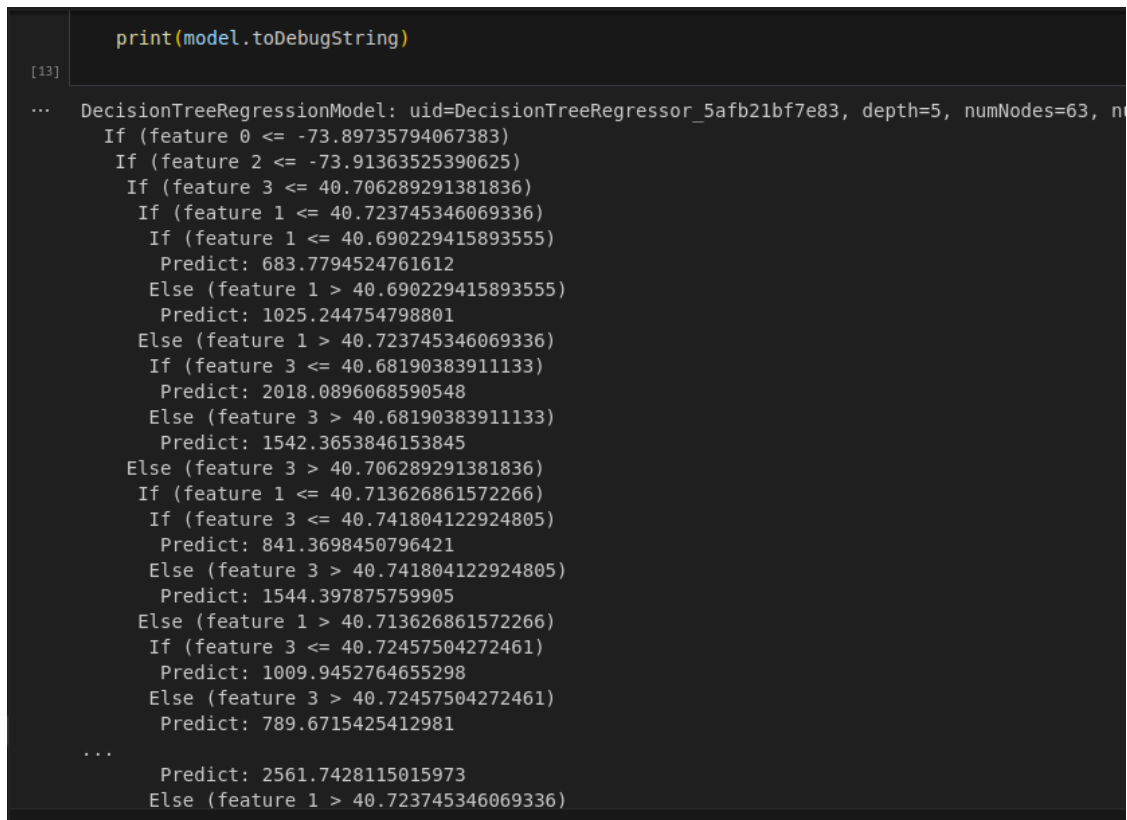


The screenshot shows a Jupyter Notebook interface with two code cells. The first cell, labeled [11], contains code to split data into training and validation sets using `randomSplit` with an 80-20 split ratio and seed 42. The second cell, labeled [12], contains code to create a `DecisionTreeRegressor` model and fit it to the training data. The notebook has a dark theme and includes buttons for '+ Code' and '+ Markdown' in the second cell.

```
[11] # training, validation = scaler_df.randomSplit([0.8, 0.2], seed = 42)
      training, validation = assembled_df.randomSplit([0.8, 0.2], seed = 42)

[12] lr = DecisionTreeRegressor(featuresCol = "features", labelCol = "trip_duration")
      model = lr.fit(training)
```

Hình 43. Chia dữ liệu và huấn luyện



The screenshot shows a Jupyter Notebook interface with a code cell labeled [13] that prints the debug string of a trained `DecisionTreeRegressor` model. The output is a complex decision tree structure with multiple if-else conditions and predicted values. The notebook has a dark theme.

```
[13] print(model.toDebugString)

... DecisionTreeRegressionModel: uid=DecisionTreeRegressor_5afb21bf7e83, depth=5, numNodes=63, n
    If (feature 0 <= -73.89735794067383)
    If (feature 2 <= -73.91363525390625)
    If (feature 3 <= 40.706289291381836)
    If (feature 1 <= 40.723745346069336)
    If (feature 1 <= 40.690229415893555)
    Predict: 683.7794524761612
    Else (feature 1 > 40.690229415893555)
    Predict: 1025.244754798801
    Else (feature 1 > 40.723745346069336)
    If (feature 3 <= 40.68190383911133)
    Predict: 2018.0896068590548
    Else (feature 3 > 40.68190383911133)
    Predict: 1542.3653846153845
    Else (feature 3 > 40.706289291381836)
    If (feature 1 <= 40.713626861572266)
    If (feature 3 <= 40.741804122924805)
    Predict: 841.3698450796421
    Else (feature 3 > 40.741804122924805)
    Predict: 1544.397875759905
    Else (feature 1 > 40.713626861572266)
    If (feature 3 <= 40.72457504272461)
    Predict: 1009.9452764655298
    Else (feature 3 > 40.72457504272461)
    Predict: 789.6715425412981
    ...
    Predict: 2561.7428115015973
    Else (feature 1 > 40.723745346069336)
```

Hình 44. Cây quyết định

23 Nhập môn dữ liệu lớn

- Đánh giá

```
• Đánh giá mô hình bằng các độ đo

evaluator = RegressionEvaluator(labelCol = 'trip_duration', predictionCol = 'prediction')
rmse_val = evaluator.setMetricName('rmse').evaluate(predictions_val)
r2_val = evaluator.setMetricName('r2').evaluate(predictions_val)
print(f"[VALIDATION] RMSE: {rmse_val:.2f}, R²: {r2_val:.2f}")

[15]

... [Stage 50:=====> (3 + 6) 9]
[VALIDATION] RMSE: 4913.78, R²: 0.01
```

Hình 45. Đánh giá mô hình

- Dự đoán kết quả và ghi ra file

```
... +-----+-----+
|      id|trip_duration|
+-----+-----+
|id3004672|      789|
|id3505355|      683|
|id1217141|      789|
|id2150126|      789|
|id1598245|      789|
|id0668992|      789|
|id1765014|     1009|
|id0898117|      841|
|id3905224|     2090|
|id1543102|      789|
|id3024712|      683|
|id3665810|      789|
|id1836461|      789|
|id3457080|      789|
|id3376065|      789|
|id3008739|      789|
|id0902216|      789|
|id3564824|      789|
|id0820280|      789|
|id0775088|      789|
+-----+-----+
only showing top 20 rows
```

Hình 46. Kết quả dự đoán

II. MLlib RDD-Based Implementation

1. Ý tưởng thực hiện

Mục tiêu bài toán:

- Dự đoán thời gian chuyến đi (trip_duration) dựa trên các thông tin như:
- Vị trí đón và trả khách (longitude, latitude)
- Số lượng hành khách (passenger_count)

Ý tưởng giải pháp: Áp dụng thuật toán Decision Tree Regression của MLlib (dựa trên RDD) để xây dựng mô hình dự đoán thời gian chuyến đi.

RDD-Based API:

- Cần kiểm soát chặt chẽ các thao tác xử lý dữ liệu.
- Tối ưu bộ nhớ và hiệu suất khi xử lý tệp dữ liệu lớn dạng text.
- Thích hợp để làm quen với nguyên lý hoạt động của hệ sinh thái Spark.

2. Mô tả chi tiết

- Khởi tạo Spark Session, đọc dữ liệu đầu vào và chuyển thành các dòng dữ liệu phù hợp trong việc xây dựng mô hình.

```
Spark initialization

spark = SparkSession.builder \
    .appName("DecisionTreeRegressor") \
    .getOrCreate()
sc = spark.sparkContext

your 131072x1 screen size is bogus. expect trouble
25/05/02 09:22:18 WARN Utils: Your hostname, LAPTOP-I95CP2I9 resolves to a loopback address: 127.0.1.1; using 10.255.255.254 instead
25/05/02 09:22:18 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/05/02 09:22:19 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
```

Hình 47. Khởi tạo Spark

- Dữ liệu từ các tệp sample_submission.csv, train.csv, test.csv được đọc bằng textFile, tạo các RDD, mỗi dòng là một chuỗi văn bản (string).

25 Nhập môn dữ liệu lớn

```
sample_data = sc.textFile("hdfs://localhost:9000/lab03/sample_submission.csv")
train_data = sc.textFile("hdfs://localhost:9000/lab03/train.csv")
test_data = sc.textFile("hdfs://localhost:9000/lab03/test.csv")

sample_data_header = sample_data.first()
train_data_header = train_data.first()
test_data_header = test_data.first()

sample_data = sample_data.filter(lambda line: line != sample_data_header)
train_data = train_data.filter(lambda line: line != train_data_header)
test_data = test_data.filter(lambda line: line != test_data_header)
```

Hình 48. Đọc dữ liệu từ HDFS

- Parsing the data: Chọn ra các cột dạng numeric là những đặc trưng cần thiết để phục vụ huấn luyện mô hình từ tập dữ liệu `train.csv`. Sau đó ép kiểu `float`. `train_data_parsed` thu được là một RDD của danh sách số thực, mỗi phần tử đại diện cho một chuyến đi với 5 đặc trưng + 1 nhãn.

```
train_data_parsed = train_data.map(lambda line: line.split(",")).map(lambda parts: [
    float(parts[5]), float(parts[6]), #pickup_longitude, pickup_latitude
    float(parts[7]), float(parts[8]), #dropoff_longitude, dropoff_latitude
    float(parts[4]), #passenger_count
    float(parts[10]) #trip_duration
])
```

Hình 49. Chọn lọc dữ liệu đầu vào

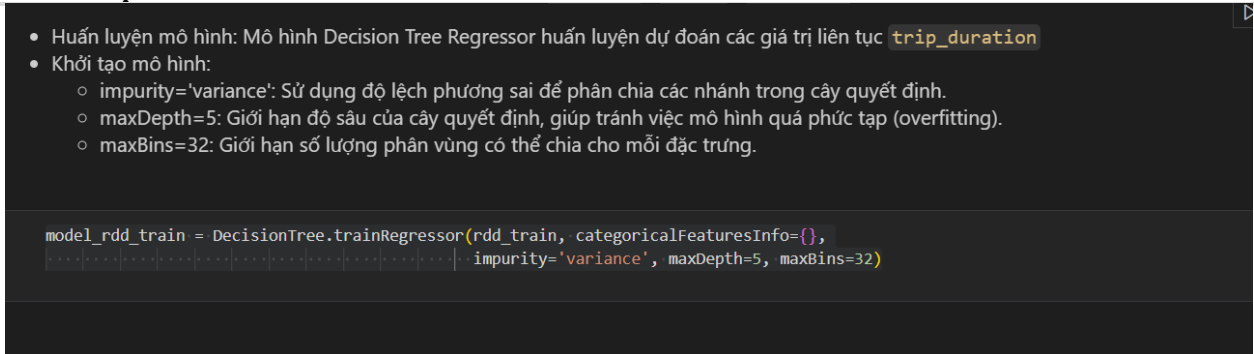
- Tiền xử lý dữ liệu: Chia dữ liệu thành tập huấn luyện và kiểm tra: Dữ liệu được chia thành hai phần: (training set) và (validation set). Việc chia này đảm bảo rằng mô hình không "học thuộc" (overfitting) dữ liệu huấn luyện và có thể tổng quát tốt trên dữ liệu chưa thấy. Chuyển đổi thành `LabeledPoint`: đóng gói đặc trưng và nhãn thành cấu trúc `LabeledPoint` để phù hợp với các mô hình học máy trong Spark.

```
train, validation = train_data_parsed.randomSplit([0.8, 0.2], seed=42)
rdd_train = train.map(lambda cols:
    LabeledPoint(cols[-1], Vectors.dense(cols[:-1]))
)
rdd_test = validation.map(lambda cols:
    LabeledPoint(cols[-1], Vectors.dense(cols[:-1]))
)
```

Hình 50. Chia dữ liệu và chuẩn hóa

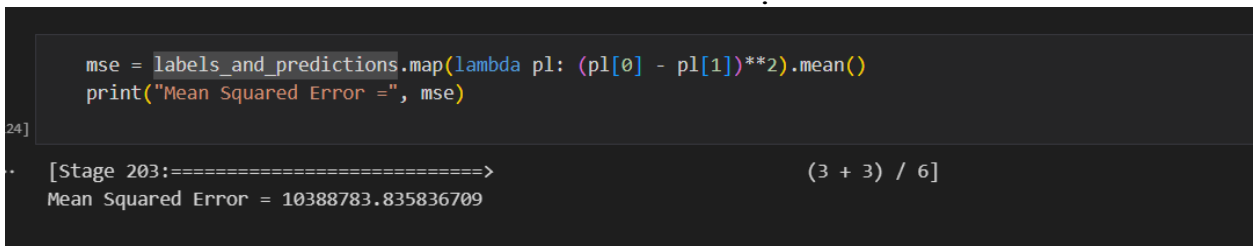
- Huấn luyện mô hình: Huấn luyện mô hình: Mô hình Decision Tree Regressor huấn luyện dự đoán các giá trị liên tục `trip_duration`

26 Nhập môn dữ liệu lớn



Hình 51. Huấn luyện mô hình

- Đánh giá mô hình: MSE đo lường trung bình bình phương khoảng cách giữa giá trị dự đoán và thực tế. Ở đây, giá trị này hơn 10 triệu là khá lớn. Điều này có nghĩa là mỗi lần dự đoán trung bình có thể lệch tới gần 54 phút so với thực tế.
- Cải thiện mô hình:
 - Thêm đặc trưng mới như thời gian trong ngày, ngày trong tuần.v.v.
 - Tăng độ sâu của cây (`maxDepth`) để mô hình học kỹ hơn.
 - Chuẩn hóa dữ liệu nếu có outlier.
 - Thử mô hình khác như Random Forest hoặc Gradient Boosted Trees.



Hình 52. Đánh giá mô hình

III. Low-Level Operations

1. Ý tưởng thực hiện

Mục tiêu bài toán

- Dự đoán `trip_duration` (thời gian chuyển đi) từ các đặc trưng:
- Vị trí đón và trả khách (kinh độ, vĩ độ)
- Số hành khách (`passenger_count`)

Ý tưởng giải pháp

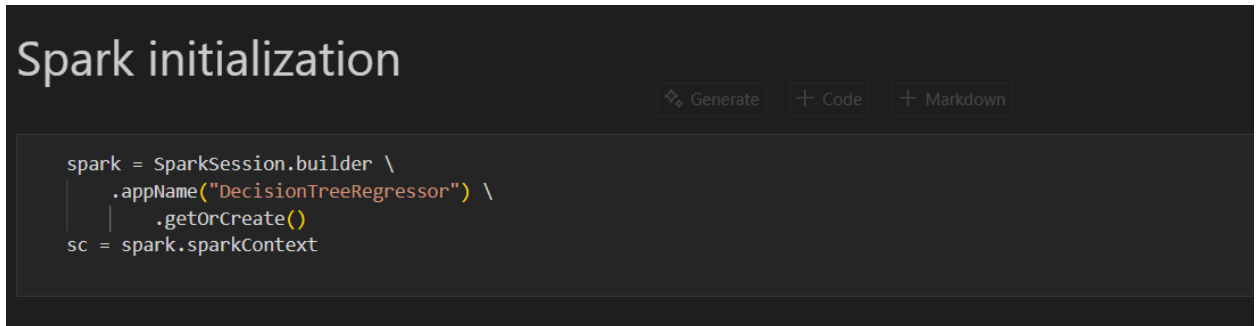
- Thay vì sử dụng thư viện có sẵn như MLlib, mô hình được triển khai thủ công từ đầu (low-level) bằng cách:
- Xây dựng cây hồi quy (decision tree regressor) từ số liệu huấn luyện

27 Nhập môn dữ liệu lớn

- Áp dụng thuật toán chia nhánh (recursive splitting) theo feature và threshold tối ưu
- Dự đoán dữ liệu alidation thông qua việc traverse cây quyết định

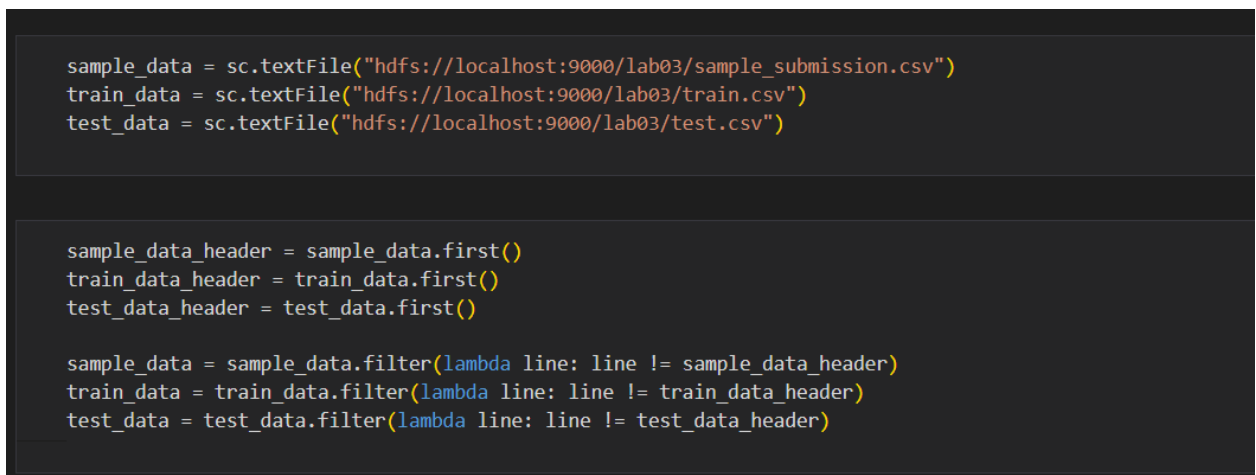
2. Mô tả chi tiết

- Khởi tạo Spark Session, đọc dữ liệu đầu vào và chuyển thành các dòng dữ liệu phù hợp trong việc xây dựng mô hình.



Hình 53. Khởi tạo Spark

- Dữ liệu từ các tệp sample_submission.csv, train.csv, test.csv được đọc bằng textFile, tạo các RDD, mỗi dòng là một chuỗi văn bản (string).



Hình 54. Đọc file từ HDFS

- Parsing the data: Chọn ra các cột dạng numeric là những đặc trưng cần thiết để phục vụ huấn luyện mô hình từ tập dữ liệu `train.csv`. Sau đó ép kiểu `float`. `train_data_parsed` thu được là một RDD của danh sách số thực, mỗi phần tử đại diện cho một chuyên đi với 5 đặc trưng + 1 nhãn.

28 Nhập môn dữ liệu lớn

```
train_data_parsed = train_data.map(lambda line: line.split(",")).map(lambda parts: [
    float(parts[5]), float(parts[6]), #pickup_longitude, pickup_latitude
    float(parts[7]), float(parts[8]), #dropoff_longitude, dropoff_latitude
    float(parts[4]), #passenger_count
    float(parts[10]) #trip_duration
])
```

Hình 55. Chọn lọc dữ liệu đầu vào

- Tiền xử lý dữ liệu: Chia dữ liệu thành tập huấn luyện và kiểm tra: Dữ liệu được chia thành hai phần: (training set) và (validation set). Việc chia này đảm bảo rằng mô hình không "học thuộc" (overfitting) dữ liệu huấn luyện và có thể tổng quát tốt trên dữ liệu chưa thấy.

Chia tập dữ liệu thành 2 phần 80/20:

- **train**: Tập dữ liệu dùng để huấn luyện mô hình.
- **validation**: Tập dữ liệu dùng để đánh giá mô hình đã được training trước đó.

```
train, validation = train_data_parsed.randomSplit([0.8, 0.2], seed=42)
```

Hình 56. Chia dữ liệu train, validation

- Split data: Chia dữ liệu thành 2 nhánh (trái, phải) dựa trên 'feature_index' và 'threshold'
 - Nhánh trái chứa các điểm có giá trị \leq threshold
 - Nhánh phải chứa các điểm có giá trị $>$ threshold

```
def split_data(data, feature_index, threshold):
    ... left = data.filter(lambda x: x[feature_index] <= threshold)
    ... right = data.filter(lambda x: x[feature_index] > threshold)
    ... return left, right
```

Hình 57. Chia nhánh cây

29 Nhập môn dữ liệu lớn

- Calculate mse: Mean Squared Error (MSE) của tập dữ liệu đầu vào và giá trị trung bình y . MSE được dùng để đo độ phân tán so với trung bình.

```
def calculate_mse(data):
    y_i = data.map(lambda x: x[-1])
    y_i_mean = y_i.mean()
    mse = y_i.map(lambda x: (x - y_i_mean) ** 2).mean()
    return mse, y_i_mean
```

Hình 58. Hàm tính MSE

- Find best split: Tìm ra `feature` và ngưỡng `threshold` tốt nhất để chia dữ liệu, sao cho MSE của hai tập con (sau khi chia) là nhỏ nhất.

```
def find_best_split(data, num_features, sample_thresholds=20):
    best_feature, best_threshold, best_mse = None, None, float("inf")
    data_count = data.count()

    for feature_idx in range(num_features):
        # Lấy mẫu threshold đại diện
        feature_values = data.map(lambda x: x[feature_idx]).distinct().takeSample(False, sample_thresholds)
        for threshold in feature_values:
            left, right = split_data(data, feature_idx, threshold)
            left_count = left.count()
            right_count = right.count()

            # Bỏ qua nếu chia không hợp lý
            if left_count == 0 or right_count == 0:
                continue

            left_mse, _ = calculate_mse(left)
            right_mse, _ = calculate_mse(right)
            mse = (left_mse * left_count + right_mse * right_count) / data_count

            if mse < best_mse:
                best_feature, best_threshold, best_mse = feature_idx, threshold, mse
    return best_feature, best_threshold
```

Hình 59. Hàm chia dữ liệu tối ưu nhất

- Build tree: Xây dựng cây theo phương pháp đệ quy:
 - Nếu đã đạt đến độ sâu tối đa hoặc dữ liệu quá ít -> trả về giá trị trung bình hàm lá (dự đoán)
 - Ngược lại:
 - Tìm cách chia tối ưu (find_best_split)
 - Đệ quy xây nhánh trái và phải

30 Nhập môn dữ liệu lớn

```
def build_tree(data, depth, max_depth):
    if depth == max_depth or data.count() <= 5:
        _, y_i_mean = calculate_mse(data)
        return y_i_mean

    best_feature, best_threshold = find_best_split(data, len(data.first()) - 1)

    if best_feature is None:
        _, y_i_mean = calculate_mse(data)
        return y_i_mean

    left, right = split_data(data, best_feature, best_threshold)
    left.cache()
    right.cache()

    return {
        "feature": best_feature,
        "threshold": best_threshold,
        "left": build_tree(left, depth + 1, max_depth),
        "right": build_tree(right, depth + 1, max_depth)
    }
```

Hình 60. Hàm xây dựng Decision tree

- Predict
 - Kiểm tra nếu tree là một dictionary -> đây là nút không phải lá.
 - So sánh row[feature] với threshold:
 - Nếu nhỏ hơn hoặc bằng -> duyệt tiếp xuống nhánh trái (left)
 - Ngược lại -> duyệt xuống nhánh phải (right)
 - Nếu không phải dict (tức là giá trị thực tại lá) -> trả về giá trị đó (là dự đoán).

```
def predict(tree, row):
    if isinstance(tree, dict):
        if row[tree["feature"]] <= tree["threshold"]:
            return predict(tree["left"], row)
        else:
            return predict(tree["right"], row)
    else:
        return tree
```

Hình 61. Hàm dự đoán

- Đánh giá mô hình: MSE đo lường trung bình bình phương khoảng cách giữa giá trị dự đoán và thực tế. Ở đây, giá trị này gần 10 triệu là khá lớn. Điều này có nghĩa là mỗi lần dự đoán trung bình có thể lệch tới gần 53 phút so với thực tế. Cho thấy mô hình train chưa được tối ưu. R^2 xấp xỉ 0 cho thấy, mô hình gần như không học được quy tắc nào

31 Nhập môn dữ liệu lớn

cả.

```
# In kết quả
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R^2 Score: {r2}")
```

```
Mean Squared Error (MSE): 9891066.18029336
Root Mean Squared Error (RMSE): 3145.006546939666
R^2 Score: 0.01781753491506699
```

Hình 62. Đánh giá mô hình

IV. So sánh các phương pháp giải quyết

	Ưu điểm	Nhược điểm
Structured API-High Level	- Cú pháp ngắn gọn, dễ viết và dễ hiểu - Tích hợp tốt với Spark SQL và DataFrame - Tối ưu hiệu suất nhờ Catalyst Optimizer và Tungsten Engine	- Khó can thiệp sâu vào quá trình xử lý - Ít linh hoạt nếu cần thực hiện các thao tác đặc thù ngoài mô hình hóa đơn giản
MLLib RDD-Based	- Dễ xây dựng mô hình học máy - Có sẵn các thuật toán như Decision Tree, Random Forest, v.v. - Có thể chạy phân tán trực tiếp trên RDD	- Mức độ trừu tượng vẫn còn cao - Không tùy biến được cấu trúc thuật toán - Đôi khi chậm nếu dữ liệu không được chuẩn hóa tốt
Low-Level Operations	- Linh hoạt tối đa: tự xây dựng mô hình, kiểm soát mọi bước - Hiểu rõ cách thuật toán hoạt động - Phù hợp với mục đích học thuật hoặc nghiên cứu	- Cần viết nhiều mã hơn - Dễ lỗi logic - Hiệu suất có thể kém hơn nếu không cache và chia dữ liệu hợp lý

PHẦN 3: BÁO CÁO NHÓM

- Phương thức trao đổi và làm việc:
 - Vì khối lượng công việc lớn đồng thời để đảm bảo chất lượng, nhóm phân chia thành 2 đội nhóm chịu trách nhiệm giải quyết hai vấn đề lớn.
 - Nhóm tổ chức phiên họp nhóm sau một khoảng thời gian làm việc cá nhân để theo dõi sát tình hình giải quyết bài tập.
 - Phiên họp nhằm kiểm tra kết quả, ý tưởng giải quyết cho từng vấn đề.
 - Giao công việc cho từng thành viên trên workspace của nhóm sau khi thống nhất kết quả.
- Bảng phân công công việc:

Thành viên	Công việc	Lưu ý
Trương Tiên Anh	Thực hiện các phương thức giải quyết bài tập đối với vấn đề Regression with Decision Trees. Khảo sát chất lượng các file, tổ chức thư mục và chú thích tại từng file.	Cả hai thành viên giải quyết vấn đề cùng nhau, chia sẻ các phương pháp cho từng cách tiếp cận và cân nhắc tính khả dụng ở các cách tiếp cận khác.
Đoàn Minh Cường		
Đinh Viết Lợi	Thực hiện các phương thức giải quyết bài tập đối với vấn đề Classification with Logistic Regression. Viết báo cáo tổng hợp bài tập nhóm.	
Nguyễn Trần Lợi		