TRƯỜNG ĐẠI HỌC CÔNG NGHỆ - ĐHQGHN

KĨ THUẬT VÀ CÔNG NGHỆ DỮ LIỆU LỚN

Classifying in KNIME to identify big spenders in Catch the Pink Flamingo

Giảng viên: TS. Trần Hồng Việt
ThS. Ngô Minh Hương

NHÓM 4

Nguyễn Thị Ngọc Lan - 23020390

Hoàng Ngọc Điệp - 23020357

Tạ Giang Thuỳ Loan - 23020397

NỘI DUNG

I. GIỚI THIỆU

I. GIỚI THIỆU
CATCH THE PINK FLAMINGO

"Catch the Pink Flamingo" là một trò chơi trên thiết bị di động nơi người chơi thực hiện các nhiệm vụ theo thời gian thực để bắt những con hồng hạc hồng trên bản đồ dạng lưới.

Trò chơi yêu cầu người chơi phải phủ điểm lên toàn bộ bản đồ để qua màn, với mỗi level tăng dần về độ phức tạp. Người chơi có thể mua vật phẩm hỗ trợ như ống nhòm, băng đá hoặc thẻ chuyển điểm.

I. GIỚI THIỆU
CATCH THE PINK FLAMINGO

Một người được coi là **big spender** nếu người đó chi tiêu tổng số tiền lớn hơn hoặc bằng 1 ngưỡng nhất định (Rule Engine hoặc KMeans). Khi huấn luyện mô hình, dữ liệu được dùng chỉ là 3 session đầu tiên của mỗi người dùng.

Big spender ít nhưng tạo ra nhiều doanh thu cho chủ thầu vậy nên phải để mô hình cố gắng phân tích được chính xác nhiều big spender nhất có thể mặc dù dữ liệu có bị mất cân bằng để đưa ra được chiến lược tiếp thị phù hợp.

I. GIỚI THIỆU
CATCH THE PINK FLAMINGO

Bài toán xác định big spenders trong trò Catch the Pink Flamingo cần trải qua các công đoạn chính:

- Chuẩn bị dữ liệu và gán nhãn
- Phân tích dữ liệu và đánh giá
- Tiền xử lý dữ liệu và chọn đặc trưng
- Huấn luyện mô hình và đánh giá trên tập test

Nhãn: big spender, low spender - người dùng không chi tiêu nhiều

II. PHÂN TÍCH DỮ LIỆU

1.Gán nhãn & trực quan hoá

II. PHÂN TÍCH DỮ LIỆU

Gán nhãn bằng KMINE để phân tích và lựa chọn ta sẽ thu được 1 tập chứa các nhãn tương ứng với big spender và low spender.

Theo Rule Engine, ta có thể lấy 5% số người chi nhiều tiền nhất để định nghĩa big spender, với mức chi tiêu mốc là 70 đô.

Theo KMeans ta sẽ chọn số cluster là 2.

=> KMeans nhận biết được nhiều big spender hơn so với Rule Engine theo luật tự định nghĩa

```
[ ]  quantile = user_stats.approxQuantile("total_spent", [0.95], 0.01)[0]
```

```
[ ]  quantile
```

```
70.0
```

Hình 1: Tính ngưỡng chi tiêu để phân luật phân loại big spender và low spender theo Rule Engine

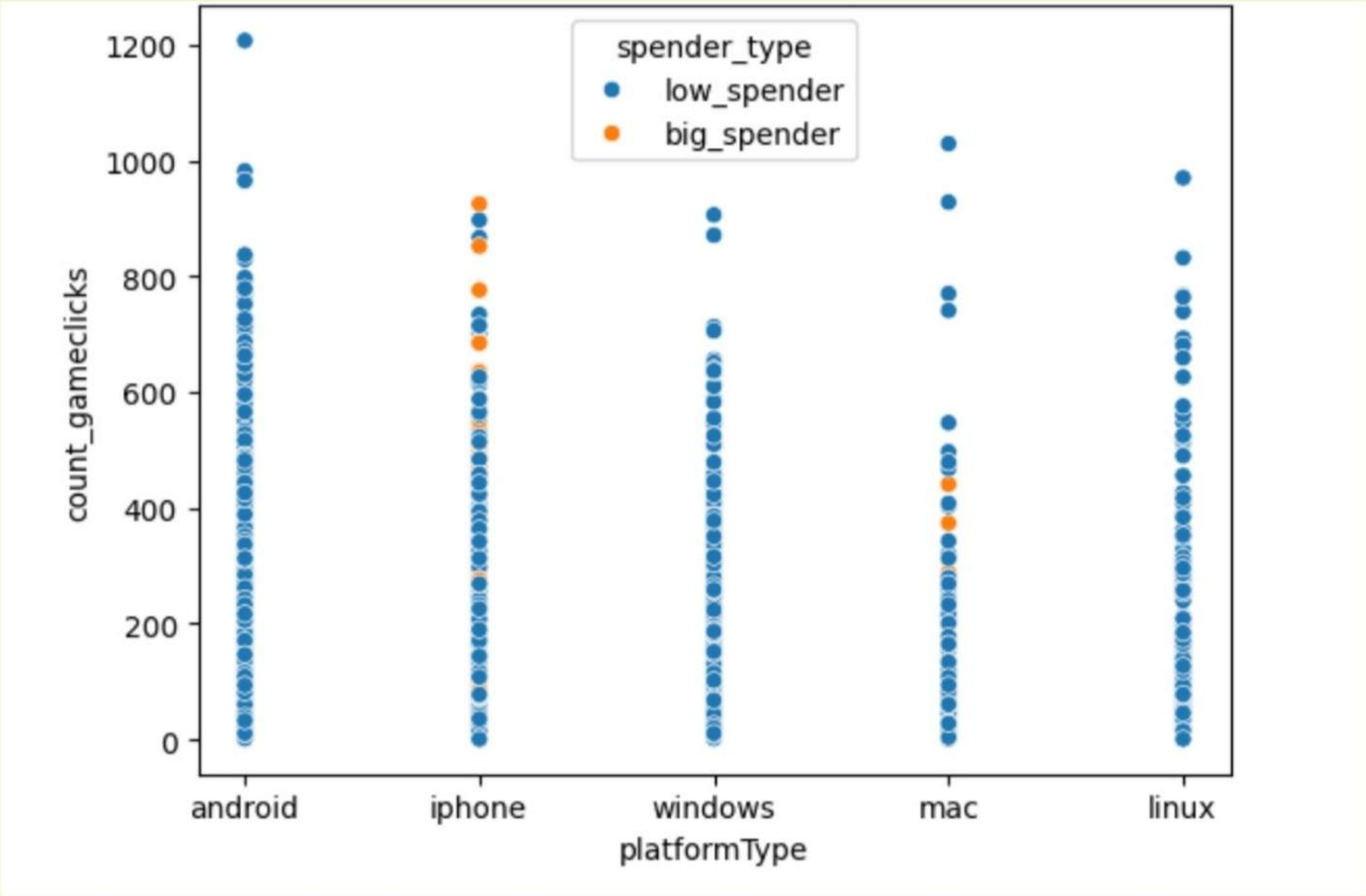
2. Phân tích dữ liệu

II. PHÂN TÍCH DỮ LIỆU

Tiền xử lý dữ liệu bao gồm:

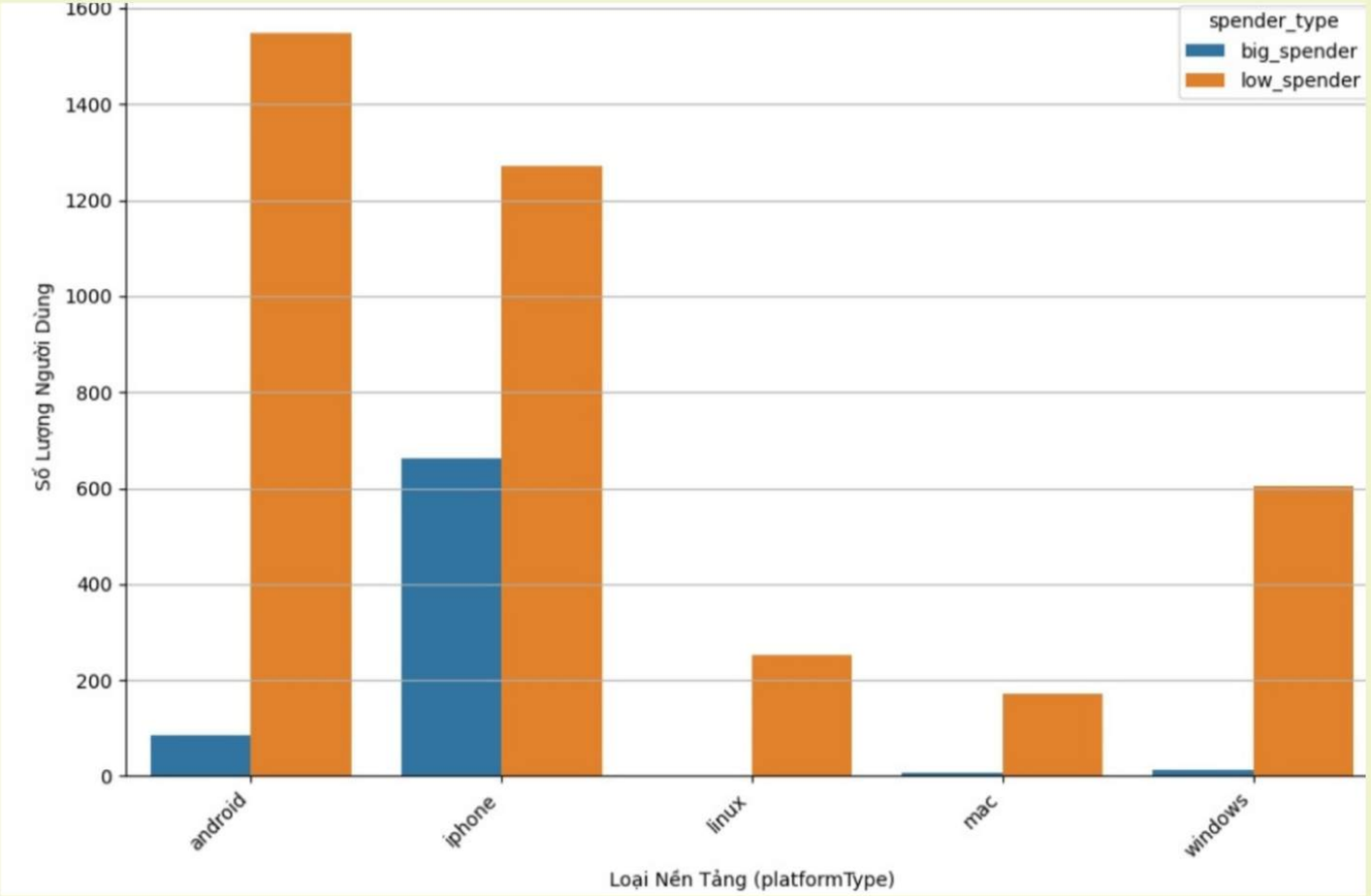
- Xử lý các giá trị NULL trong cột count_buyId và avg_price (bằng cách thay thế bằng 0 và chuyển đổi kiểu dữ liệu phù hợp)
- Tạo cột made_purchase để xác định các phiên có giao dịch
- Tạo cột mục tiêu spender_type (big spender /low spender) dựa trên total_spent của mỗi người dùng

II. PHÂN TÍCH DỮ LIỆU



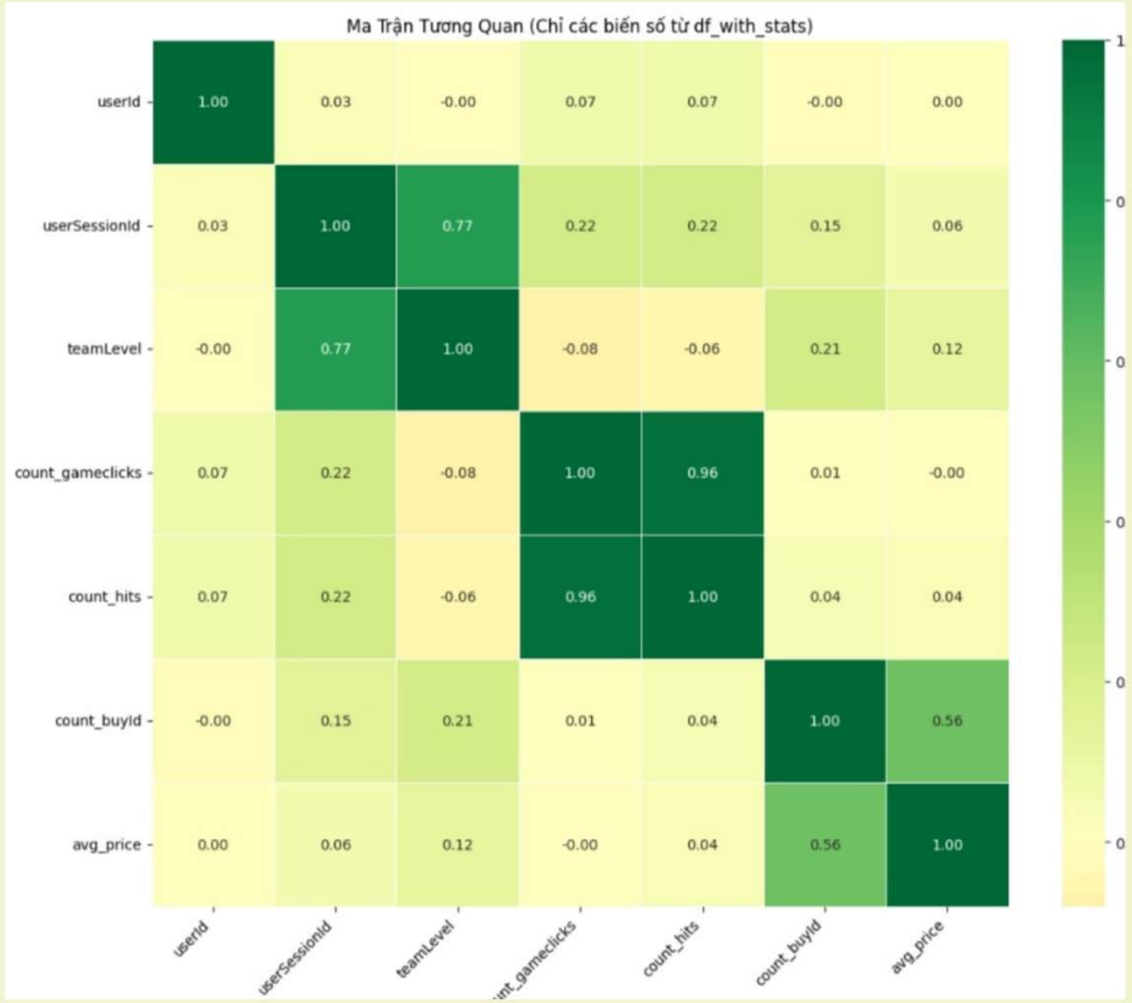
Hình 2 : Biểu đồ phân tán: Số Lượt Nhấp Chuột Trong Game theo Loại Nền Tảng và Phân Loại Người Chi Tiêu.

II. PHÂN TÍCH DỮ LIỆU



Hình 3 : Biểu đồ cột: Số Lượng Người Dùng theo Loại Nền Tảng và Phân Loại Người Chi Tiêu.

II. PHÂN TÍCH DỮ LIỆU



Hình 4 : Heatmap: Ma Trận Tương Quan giữa các Chỉ Số.

III. TIỀN XỬ LÝ DỮ LIỆU

III. TIỀN XỬ LÝ DỮ LIỆU

1.CHỌN ĐẶC TRƯNG

2. XỬ LÝ DỮ LIỆU

3. CHUẨN HOÁ

1. Chọn đặc trưng

III. TIỀN XỬ LÝ DỮ LIỆU

Bảng 1: Phân bố số lượng người chơi dựa trên spender_type và platformType

spender_type	platformType	count
big_spender	android	86
big_spender	iphone	663
big_spender	mac	7
big_spender	windows	14
low_spender	android	1549
low_spender	iphone	1271
low_spender	linux	252
low_spender	mac	172
low_spender	windows	605

Bảng 2 : Phân bố số lượng người chơi dựa trên spender_type và teamLevel

spender_type	teamLevel	count
big_spender	1	82
big_spender	2	94
big_spender	3	102
big_spender	4	117
big_spender	5	128
big_spender	6	129
big_spender	7	117
big_spender	8	1
low_spender	1	375
low_spender	2	432

2. Xử lý dữ liệu

III. TIỀN XỬ LÝ DỮ LIỆU

Bảng 3 : Các đặc trưng của dữ liệu trong 3 session đầu tiên

userId	platformType	userSessionId	teamLevel	count_gameclicks	count_hits	count_buyId	avg_price	made_purchase	total_purchases	total_spent	spender_type	session_rank
0	iphone	23473	1	237	28	0	0.0	0	0	0.0	low_spender	1
0	iphone	24943	2	353	35	0	0.0	0	0	0.0	low_spender	2
0	iphone	28377	3	381	32	0	0.0	0	0	0.0	low_spender	3
1	android	5835	1	73	7	0	0.0	0	8	19.0	low_spender	1
1	android	7847	2	95	14	0	0.0	0	8	19.0	low_spender	2
1	android	10041	3	93	9	2	3.0	1	8	19.0	low_spender	3
2	iphone	12107	3	7	0	0	0.0	0	0	0.0	low_spender	1
2	iphone	12411	4	39	3	0	0.0	0	0	0.0	low_spender	2
2	iphone	15899	5	35	6	0	0.0	0	0	0.0	low_spender	3
6	iphone	28628	7	90	5	0	0.0	0	0	0.0	low_spender	1
8	iphone	27410	4	25	1	0	0.0	0	3	33.0	low_spender	1

Bảng 4: Các đặc trưng của dữ liệu (phần số) sau khi đã gom nhóm

userId	platformType	avg_clicks	avg_hits	total_early_buys	early_avg_price	has_early_purchase	session_count
0	iphone	323.6666666666667	31.666666666666668	0	0.0	0	3
1	android	87.0	10.0	2	1.0	1	3
2	iphone	27.0	3.0	0	0.0	0	3
6	iphone	90.0	5.0	0	0.0	0	1
8	iphone	103.5	12.0	3	5.5	1	2
9	iphone	54.0	6.333333333333333	2	6.666666666666667	1	3
10	linux	719.0	81.66666666666667	4	0.6666666666666666	1	3
12	iphone	78.66666666666667	7.333333333333333	2	10.0	1	3

III. TIỀN XỬ LÝ DỮ LIỆU

```
[62] spender_indexer = StringIndexer(inputCol="spender_type", outputCol="spender_type_index")
     platform_indexer = StringIndexer(inputCol="platformType", outputCol="platform_index")
     final_features = spender_indexer.fit(final_features).transform(final_features)
     final_features = platform_indexer.fit(final_features).transform(final_features)
```

```
[63] final_features = final_features.drop('platformType','spender_type')
```

```
final_features.show()
```

```
+------+------------------+------------------+----------------+------------------+-------------+-------------+------------------+--------------+
|userId|        avg_clicks|          avg_hits|total_early_buys|   early_avg_price|session_count|max_teamLevel|spender_type_index|platform_index|
+------+------------------+------------------+----------------+------------------+-------------+-------------+------------------+--------------+
|     0| 323.6666666666667|31.666666666666668|               0|               0.0|            3|            3|               0.0|           0.0|
|     1|              87.0|              10.0|               2|               1.0|            3|            3|               0.0|           1.0|
|     2|              27.0|               3.0|               0|               0.0|            3|            5|               0.0|           0.0|
|     6|              90.0|               5.0|               0|               0.0|            1|            7|               0.0|           0.0|
|     8|             103.5|              12.0|               3|               5.5|            2|            5|               0.0|           0.0|
|     9|              54.0| 6.333333333333333|               2| 6.666666666666667|            3|            4|               1.0|           0.0|
|    10|             719.0| 81.66666666666667|               4|0.6666666666666666|            3|            3|               0.0|           3.0|
|    12| 78.66666666666667| 7.333333333333333|               2|              10.0|            3|            3|               1.0|           0.0|
|    13|             172.0|              16.5|               4|               4.0|            2|            7|               0.0|           1.0|
|    14|              93.0|               6.0|               0|               0.0|            1|            7|               0.0|           3.0|
```

Bảng 5: Các đặc trưng sau khi chọn và gom nhóm, chuyển đổi

```
[44] encoder = OneHotEncoder(
         inputCols=[ "platform_index"],
         outputCols=[ "platform_ohe"]
     )
     final_features = encoder.fit(final_features).transform(final_features)
```

3. Chuẩn hoá

III. TIỀN XỬ LÝ DỮ LIỆU

Hình 6: Chuyển đổi, chuẩn hoá các đặc trưng

```
pre_assembled = VectorAssembler(inputCols=numeric_cols, outputCol="numeric_vector").transform(final_features)

scaler = StandardScaler(inputCol="numeric_vector", outputCol="scaled_numeric", withMean=True, withStd=True)
scaler_model = scaler.fit(pre_assembled)
scaled_df = scaler_model.transform(pre_assembled)
final_assembler = VectorAssembler(
    inputCols=[
    "scaled_numeric",
    "max_teamLevel", "platform_ohe"],
    outputCol="features"
)
final_df = final_assembler.transform(scaled_df)
```

Hình 7 : Tỷ lệ big spender - low spender

```
+------------+-----+
|spender_type|count|
+------------+-----+
| big_spender|  770|
| low_spender| 3849|
+------------+-----+
```

IV. MÔ HÌNH

IV. MÔ HÌNH

```
rf = RandomForestClassifier(
    featuresCol="features",
    labelCol="spender_type_index",
    weightCol="classWeightCol",
    numTrees=90,
    maxDepth=15,
    maxBins=64
)
rf_model = rf.fit(train_df)
train_preds = rf_model.transform(train_df)

acc_eval = MulticlassClassificationEvaluator(
    labelCol="spender_type_index",
    predictionCol="prediction",
    metricName="accuracy"
)
train_acc = acc_eval.evaluate(train_preds)

auc_eval = BinaryClassificationEvaluator(
    labelCol="spender_type_index",
    rawPredictionCol="rawPrediction",
    metricName="areaUnderROC"
)
```

```
train_auc = auc_eval.evaluate(train_preds)

print(f"[Train] Random Forest - Accuracy = {train_acc:.4f}, AUC = {train_auc:.4f}")

test_preds = rf_model.transform(test_df)

acc_eval = MulticlassClassificationEvaluator(
    labelCol="spender_type_index",
    predictionCol="prediction",
    metricName="accuracy"
)
test_acc = acc_eval.evaluate(test_preds)

auc_eval = BinaryClassificationEvaluator(
    labelCol="spender_type_index",
    rawPredictionCol="rawPrediction",
    metricName="areaUnderROC"
)
test_auc = auc_eval.evaluate(test_preds)

print(f"[Test] Random Forest - Accuracy = {test_acc:.4f}, AUC = {test_auc:.4f}")
```

```
[Train] Random Forest - Accuracy = 0.9878, AUC = 0.9993
[Test] Random Forest - Accuracy = 0.9402, AUC = 0.9699
```

Kết quả và tham số tốt nhất huấn luyện bằng Random Forest

IV. MÔ HÌNH

```python
from pyspark.ml.classification import LogisticRegression

lr = LogisticRegression(
    featuresCol="features",
    labelCol="spender_type_index",
    weightCol="classWeightCol",
    maxIter=100
)

lr_model = lr.fit(train_df)
predictions = lr_model.transform(train_df)
evaluator = MulticlassClassificationEvaluator(
    labelCol="spender_type_index",
    predictionCol="prediction",
    metricName="accuracy"
)

accuracy = evaluator.evaluate(predictions)
print(f"Train Accuracy (LogReg) = {accuracy:.4f}")

auc_eval = BinaryClassificationEvaluator(
    labelCol="spender_type_index",
    rawPredictionCol="rawPrediction",
    metricName="areaUnderROC"
)
```

```python
auc = auc_eval.evaluate(predictions)
print(f"AUC (LogReg Weighted) = {auc:.4f}")
predictions = lr_model.transform(test_df)
evaluator = MulticlassClassificationEvaluator(
    labelCol="spender_type_index",
    predictionCol="prediction",
    metricName="accuracy"
)

accuracy = evaluator.evaluate(predictions)
print(f"Test Accuracy (LogReg) = {accuracy:.4f}")
auc = auc_eval.evaluate(predictions)
print(f"AUC (LogReg Weighted) = {auc:.4f}")
```

```
Train Accuracy (LogReg) = 0.9290
AUC (LogReg Weighted) = 0.9647
Test Accuracy (LogReg) = 0.9022
AUC (LogReg Weighted) = 0.9587
```

Kết quả và tham số tốt nhất huấn luyện bằng Logistic Regression

IV. MÔ HÌNH

```
gbt = GBTClassifier(
    featuresCol="features",
    labelCol="spender_type_index",
    weightCol="classWeightCol",
    maxIter=40,
    maxDepth=5,
    stepSize=0.7
)
model = gbt.fit(train_df)
predictions = model.transform(train_df)
accuracy = accuracy_eval.evaluate(predictions)
auc = auc_eval.evaluate(predictions)
print(f"GBT Train - Accuracy = {accuracy:.4f}, AUC = {auc:.4f}")
predictions = model.transform(test_df)
accuracy = accuracy_eval.evaluate(predictions)
auc = auc_eval.evaluate(predictions)
print(f"GBT Test - Accuracy = {accuracy:.4f}, AUC = {auc:.4f}")
```

```
GBT Train - Accuracy = 0.9956, AUC = 0.9999
GBT Test - Accuracy = 0.9348, AUC = 0.9526
```

Kết quả và tham số tốt nhất huấn luyện bằng Gradient Boosted Trees

V. KẾT LUẬN

V. KẾT LUẬN

Mô hình	Accuracy trên tập test	AUC trên tập test
Logistic Regression	0.9022	0.9587
Random Forest	**0.9402**	**0.9699**
GBT	0.9348	0.9526

Bảng 8 : Tổng hợp kết quả trên tập test

THANK YOU
FOR YOUR ATTENTION