ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
KHOA TOÁN TIN
FACULTY OF MATHEMATICS AND INFORMATICS

Báo cáo Hệ Hỗ Trợ Quyết Định

Chủ đề: Car Brand Consultation

Giáo viên hướng dẫn: Trịnh Ngọc Thăng

Sinh viên : Nguyễn Bá Hùng 20216835

Ngày 7 tháng 1 năm 2025

Mục lục

I Bài toán: Phân lớp tư vấn hãng xe **4**
1 Mô tả bài toán nghiệp vụ 4
2 Mục tiêu của bài toán 4

II Xử lý dữ liệu **4**
1 Thu thập dữ liệu từ nhiều nguồn 4
1.1 Chi tiết các trường dữ liệu 5
2 Đánh nhãn dữ liệu 5
2.1 Quy trình đánh nhãn 6
3 Tiền xử lý dữ liệu 7
3.1 Chọn Các Đặc Trưng (Features) 7
3.2 Làm Sạch Dữ Liệu 7
3.3 Phân Tích Phân Bố Lớp 8
4 Chia Dữ Liệu Thành Tập Huấn Luyện Và Tập Kiểm Tra 8
4.1 Lưu Dữ Liệu Đã Xử Lý 8
5 Thống kê dữ liệu mẫu 8

III Đánh giá mô hình **10**
1 Đề xuất và lựa chọn các tiêu chí 10
1.1 Accuracy (Root Mean Squared Error) 10
1.2 Precision, Recall, F1-score 10

IV Cải tiến mô hình **12**
1 Mô hình: K-Nearest Neighbor 12
1.1 Giới thiệu về Mô Hình K-Nearest Neighbors (KNN) 12
1.2 Các Bước Thực Hiện 12
2 Mô hình: Random Forest 15
2.1 Giới thiệu 15
2.2 Quy trình thực hiện 15
3 Mô hình:AdaBoost 17
3.1 Giới thiệu 17
3.2 Quy trình thực hiện 17
4 Mô hình: AdaBoost cải tiến 20
5 Mô hình:GradientBoosting 22
5.1 Giới thiệu 22
5.2 Các bước thực hiện 22
6 Mô hình:Support Vector Machine (SVM) 23
6.1 Giới thiệu 23
6.2 Các bước thực hiện 23

7 Mô hình: XGBoost 25
7.1 Giới thiệu 25
7.2 Các bước thực hiện 26
8 Mô hình: LightGBM 27
8.1 Giới thiệu về mô hình LightGBM 27
8.2 Các bước thực hiện 27
9 Mô hình: Decision Tree 29
9.1 Giới thiệu về Mô hình Decision Tree 29
9.2 Các bước thực hiện 29
10 Mô hình:Bagging Classifier 31
10.1 Giới thiệu về Mô hình Bagging Classifier 31
10.2 Các bước thực hiện 32

V Đóng gói sản phẩm 34

I Bài toán: Phân lớp tư vấn hãng xe

1 Mô tả bài toán nghiệp vụ

Bài toán "Dự đoán hãng xe phù hợp"là một bài toán phân tích và dự đoán phổ biến trong lĩnh vực học máy, nhằm phân loại và tư vấn cho khách hàng hãng xe phù hợp với nhu cầu, sở thích và điều kiện tài chính của họ. Mô hình sẽ phân loại khách hàng vào các hãng xe dựa trên các đặc điểm, yêu cầu và sự lựa chọn của họ.

2 Mục tiêu của bài toán

- **Dự đoán hãng xe phù hợp**: Mục tiêu chính là xây dựng một mô hình học máy dự đoán hãng xe phù hợp cho khách hàng dựa trên thông tin về sở thích, tài chính, nhu cầu sử dụng và các yếu tố liên quan khác.
- **Cung cấp tư vấn cá nhân hóa**: Mô hình giúp đưa ra lời khuyên cụ thể về hãng xe phù hợp nhất cho từng khách hàng, giúp họ dễ dàng lựa chọn mẫu xe và thương hiệu phù hợp với nhu cầu.
- **Tăng cường trải nghiệm khách hàng**: Nhờ vào các yếu tố khách quan, mô hình giúp tạo ra các giải pháp tối ưu cho từng nhóm khách hàng, đồng thời giảm thiểu thời gian tìm kiếm hãng xe thích hợp.
- **Hỗ trợ chiến lược marketing của hãng xe**: Hệ thống dự báo và phân loại khách hàng giúp các hãng xe nhắm đúng đối tượng tiềm năng và đưa ra các chiến lược marketing hiệu quả để thu hút khách hàng mục tiêu.

II Xử lý dữ liệu

1 Thu thập dữ liệu từ nhiều nguồn

Liên kết bộ dữ liệu: `https://www.kaggle.com/datasets/thedevastator/uncovering-factors-that-affect-used-car-prices`

- **Dung lượng bộ dữ liệu:** 70.96 MB
- **Bộ dữ liệu** `autos.csv` chứa thông tin về xe ô tô đã qua sử dụng, bao gồm loại người bán, loại phương tiện, năm đăng ký và các đặc điểm khác, giúp phân tích giá xe ô tô đã qua sử dụng

1.1 Chi tiết các trường dữ liệu

- **dateCrawled:** Ngày xe được thu thập. (Ngày)
- **name:** Tên của xe. (Chuỗi)
- **seller:** Loại người bán (riêng tư hoặc đại lý). (Chuỗi)
- **offerType:** Loại hình giao dịch (ví dụ: bán, sửa chữa, v.v.). (Chuỗi)
- **price:** Giá của xe. (Số nguyên)
- **abtest:** Loại thử nghiệm (A hoặc B). (Chuỗi)
- **vehicleType:** Loại phương tiện (ví dụ: SUV, sedan, v.v.). (Chuỗi)
- **yearOfRegistration:** Năm đăng ký của xe. (Số nguyên)
- **gearbox:** Loại hộp số (số tay hoặc tự động). (Chuỗi)
- **powerPS:** Công suất của xe tính bằng PS. (Số nguyên)
- **model:** Mẫu xe. (Chuỗi)
- **kilometer:** Số kilomet xe đã đi. (Số nguyên)
- **monthOfRegistration:** Tháng đăng ký của xe. (Số nguyên)
- **fuelType:** Loại nhiên liệu (ví dụ: dầu diesel, xăng, v.v.). (Chuỗi)
- **brand:** Thương hiệu của xe. (Chuỗi)
- **notRepairedDamage:** Xe có thiệt hại chưa được sửa chữa hay không. (Chuỗi)
- **dateCreated:** Ngày tạo xe. (Ngày)
- **nrOfPictures:** Số lượng hình ảnh của xe. (Số nguyên)
- **postalCode:** Mã bưu chính của xe. (Số nguyên)
- **lastSeen:** Ngày xe được nhìn thấy lần cuối. (Ngày)

2 Đánh nhãn dữ liệu

Đánh nhãn dữ liệu là quá trình gán nhãn cho các điểm dữ liệu để mô hình học máy có thể học và dự đoán chính xác. Mục đích chính bao gồm:

- Cung cấp dữ liệu huấn luyện
- Đánh giá hiệu suất

- Chẩn đoán và sửa lỗi
- Cải thiện chất lượng dữ liệu
- Hỗ trợ các ứng dụng thực tế như nhận diện khuôn mặt, xử lý ngôn ngữ tự nhiên, và xe tự lái

Việc này giúp đảm bảo mô hình hoạt động chính xác và hiệu quả.

2.1 Quy trình đánh nhãn

Quá trình đánh nhãn dữ liệu được thực hiện với mục tiêu mã hóa các đặc trưng phân loại (categorical features) thành các giá trị số nguyên, hỗ trợ quá trình huấn luyện mô hình học máy.

Các đặc trưng phân loại trong dữ liệu bao gồm:

- `brand` (Thương hiệu)
- `model` (Mẫu xe)
- `vehicleType` (Loại phương tiện)
- `gearbox` (Loại hộp số)
- `fuelType` (Loại nhiên liệu)

Mã hóa này là cần thiết vì các mô hình học máy thường chỉ có thể xử lý dữ liệu số. Do đó, việc chuyển đổi các giá trị phân loại thành số nguyên sẽ làm cho dữ liệu dễ hiểu và phù hợp với các thuật toán học máy.

Phương pháp mã hóa Sử dụng `LabelEncoder` từ thư viện `sklearn.preprocessing` để chuyển đổi các giá trị phân loại thành số nguyên. Đối với mỗi cột phân loại, một đối tượng `LabelEncoder` được khởi tạo và sử dụng phương thức `.fit_transform()` để ánh xạ các giá trị phân loại thành dãy số nguyên.

Phương thức `.fit_transform()` đảm bảo các giá trị phân loại trong từng cột được mã hóa nhất quán, tạo ra bảng dữ liệu sẵn sàng cho quá trình học máy.

```
# Encode categorical columns
print("\n4. ENCODING CATEGORICAL FEATURES...")
label_encoders = {}
categorical_columns = ['brand', 'model', 'vehicleType', 'gearbox', 'fuelType'
    ]

for col in categorical_columns:
    print(f"Encoding column: {col}")
    label_encoders[col] = LabelEncoder()
```

```
9     df_processed[col] = label_encoders[col].fit_transform(df_processed[col])
```

Listing 1: Mã hóa các đặc trưng phân loại

3 Tiền xử lý dữ liệu

3.1 Chọn Các Đặc Trưng (Features)

Quá trình xử lý dữ liệu bắt đầu bằng việc lựa chọn các đặc trưng quan trọng cho mục đích phân tích. Các đặc trưng này bao gồm thông tin về thương hiệu, mẫu xe, năm đăng ký, loại phương tiện, hộp số, công suất, số km đã đi, loại nhiên liệu và giá trị xe. Những cột dữ liệu được chọn bao gồm:

```
features = ['brand', 'yearOfRegistration', 'model', 'vehicleType', 'gearbox',
            'powerPS', 'kilometer', 'fuelType', 'price']
df_processed = df[features].copy()
```

3.2 Làm Sạch Dữ Liệu

Tiến trình làm sạch dữ liệu gồm ba bước chính:

1. **Loại bỏ giá trị ngoại lệ trong cột giá** (`price`): Để đảm bảo dữ liệu hợp lý, ta giới hạn giá trị trong phạm vi từ 100 đến 150.000. Những giá trị ngoài phạm vi này sẽ bị loại bỏ:

   ```
   df_processed = df_processed[df_processed['price'].between(100, 150000)]
   ```

2. **Loại bỏ giá trị ngoại lệ trong cột số km** (`kilometer`): Giữ lại các mẫu xe có số km hợp lý, trong phạm vi từ 0 đến 300.000 km:

   ```
   df_processed = df_processed[df_processed['kilometer'].between(0, 300000)]
   ```

3. **Xử lý các giá trị thiếu**: Loại bỏ các dòng chứa giá trị thiếu (`null`) trong dữ liệu để bảo đảm chất lượng dữ liệu đầu vào:

   ```
   before_clean = len(df_processed)
   df_processed = df_processed.dropna()
   after_clean = len(df_processed)
   ```

3.3 Phân Tích Phân Bố Lớp

Sau khi làm sạch, bước tiếp theo là phân tích sự phân bố các nhãn trong cột `brand`. Điều này giúp hiểu rõ hơn về sự phân bố của các thương hiệu trong dữ liệu, với những thương hiệu phổ biến có số mẫu lớn hơn và những thương hiệu ít phổ biến có số mẫu ít hơn. Để minh họa sự phân bố, chúng ta sử dụng biểu đồ cột:

```
class_dist = df_processed['brand'].value_counts()
class_dist.plot(kind='bar')
```

4 Chia Dữ Liệu Thành Tập Huấn Luyện Và Tập Kiểm Tra

Dữ liệu được chia thành hai phần sau khi tiền xử lý:

- **Tập huấn luyện (train)**: Chiếm 80% tổng số dữ liệu.
- **Tập kiểm tra (test)**: Chiếm 20% tổng số dữ liệu.

Việc phân chia này được thực hiện thông qua hàm `train_test_split` từ thư viện `sklearn.model_selection`:

```
y = df_processed['brand']
X = df_processed.drop('brand', axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

4.1 Lưu Dữ Liệu Đã Xử Lý

Cuối cùng, các tập huấn luyện và kiểm tra sẽ được lưu thành các file CSV để phục vụ cho quá trình huấn luyện mô hình sau này:

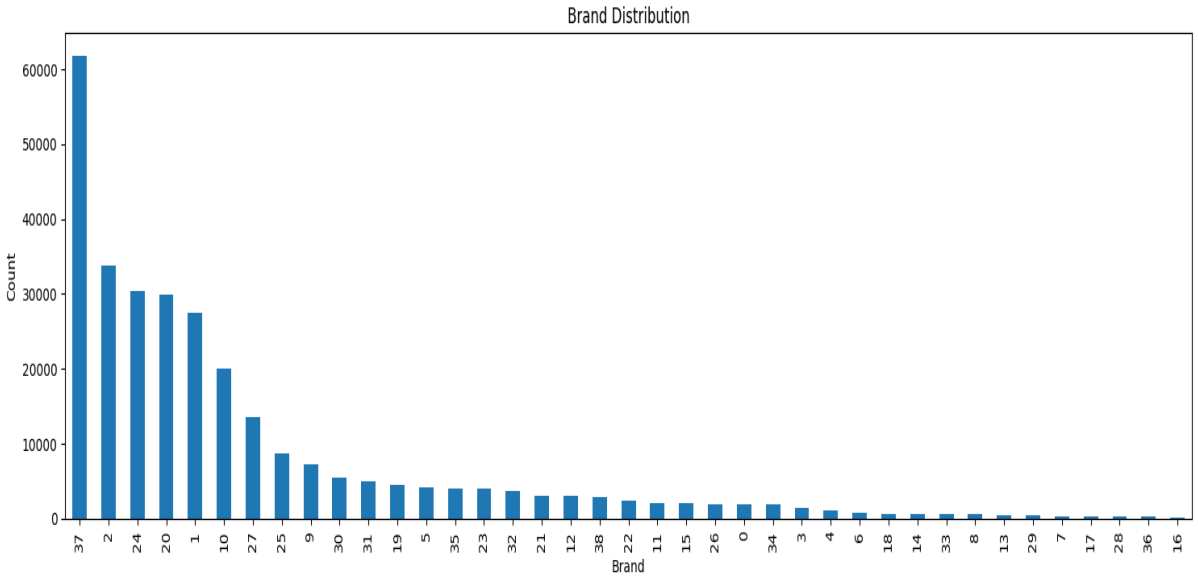
```
train_data = pd.concat([X_train, y_train], axis=1)
test_data = pd.concat([X_test, y_test], axis=1)
train_data.to_csv('car_train_data.csv', index=False)
test_data.to_csv('car_test_data.csv', index=False)
```

5 Thống kê dữ liệu mẫu

-Số lượng giá trị null trong mỗi cột:
- Biểu đồ phân bố của trường Branch:

Bảng 1: Số lượng giá trị null trong từng cột

Cột	Số lượng giá trị null
brand	0
yearOfRegistration	0
model	20,484
vehicleType	37,869
gearbox	20,209
powerPS	0
kilometer	0
fuelType	33,386
price	0



III Đánh giá mô hình

1 Đề xuất và lựa chọn các tiêu chí

1.1 Accuracy (Root Mean Squared Error)

- Trong học máy (machine learning), "accuracy"(độ chính xác) là một chỉ số đo lường hiệu suất của một mô hình, đặc biệt là trong các bài toán phân loại (classification). Accuracy được tính bằng tỷ lệ giữa số lượng dự đoán đúng trên tổng số dự đoán.

$$\text{Accuracy} = \frac{\text{Số lượng dự đoán đúng}}{\text{Tổng số dự đoán}} \tag{1}$$

- Accuracy là một chỉ số đơn giản và dễ hiểu, nhưng nó có thể không phải là chỉ số tốt nhất trong một số trường hợp, đặc biệt là khi dữ liệu không cân bằng (imbalanced dataset), tức là khi số lượng mẫu trong các lớp không bằng nhau. Trong những trường hợp như vậy, các chỉ số khác như Precision, Recall, F1-score, hoặc AUC-ROC có thể cung cấp cái nhìn toàn diện hơn về hiệu suất của mô hình.

1.2 Precision, Recall, F1-score

- Precision đo lường tỷ lệ các dự đoán dương (positive) mà thực sự là dương (positive). Nó trả lời câu hỏi: Trong số các mẫu mà mô hình dự đoán là dương, có bao nhiêu mẫu thực sự là dương?

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2}$$

Trong đó:

- TP (True Positive): Số lượng mẫu thực sự là dương và được dự đoán là dương.
- FP (False Positive): Số lượng mẫu thực sự là âm nhưng được dự đoán là dương.

- Recall đo lường tỷ lệ các mẫu dương thực sự (positive) được mô hình nhận diện chính xác. Nó trả lời câu hỏi: Trong số các mẫu thực sự là dương, có bao nhiêu mẫu được mô hình dự đoán đúng?

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3}$$

Trong đó:

- TP (True Positive): Số lượng mẫu thực sự là dương và được dự đoán là dương.
- FN (False Negative): Số lượng mẫu thực sự là dương nhưng được dự đoán là âm.

- F1-score là trung bình điều hòa (harmonic mean) của precision và recall, và nó cung cấp một chỉ số cân bằng giữa hai chỉ số này. F1-score đặc biệt hữu ích khi bạn cần cân nhắc đồng đều giữa precision và recall.

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

- F1-score dao động từ 0 đến 1, với 1 là giá trị tốt nhất (nghĩa là mô hình có cả precision và recall cao).

- Tóm lại, việc sử dụng precision, recall và F1-score sẽ giúp ta hiểu rõ hơn về hiệu suất của mô hình, đặc biệt là trong những tình huống mà accuracy không đủ để đánh giá.

IV Cải tiến mô hình

1 Mô hình: K-Nearest Neighbor

1.1 Giới thiệu về Mô Hình K-Nearest Neighbors (KNN)

Mô hình K-Nearest Neighbors (KNN) là một phương pháp phân loại đơn giản và hiệu quả trong học máy. Mô hình này hoạt động dựa trên giả thuyết rằng các đối tượng có cùng thuộc tính sẽ có xu hướng thuộc về các lớp giống nhau. Trong KNN, kết quả phân loại của một đối tượng mới được quyết định bởi các lớp của **k láng giềng gần nhất** trong không gian đặc trưng của dữ liệu.

1.2 Các Bước Thực Hiện

Trong quá trình huấn luyện mô hình KNN cho bài toán phân loại nhãn hiệu ô tô, ta thực hiện các bước chính như sau:

Bước 1: Tải và chuẩn bị dữ liệu

```
train_data = pd.read_csv('car_train_data.csv')
test_data = pd.read_csv('car_test_data.csv')
```

Dữ liệu huấn luyện và kiểm tra được tải lên từ các tệp `.csv`. Dữ liệu đã được mã hóa trước để có thể sử dụng trực tiếp mà không cần mã hóa lại.

Bước 2: Xử lý các đặc trưng và mục tiêu

```
X_train = train_data.drop('brand', axis=1)
y_train = train_data['brand']
X_test = test_data.drop('brand', axis=1)
y_test = test_data['brand']
```

- `X_train` và `X_test` là các đặc trưng (features) của dữ liệu huấn luyện và kiểm tra, còn `y_train` và `y_test` là nhãn mục tiêu (target). - Dữ liệu nhãn hiệu ô tô được phân loại theo cột `brand`, do đó, ta bỏ cột này để sử dụng làm mục tiêu phân loại.

Bước 3: Chuẩn hóa các đặc trưng số

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Để cải thiện hiệu quả mô hình KNN, ta chuẩn hóa các đặc trưng số thông qua phương pháp `StandardScaler` giúp các giá trị đặc trưng có cùng độ đo (mean=0, variance=1).

Bước 4: Huấn luyện mô hình KNN

```
knn = KNeighborsClassifier(n_neighbors=5, weights='distance', metric='
    euclidean', n_jobs=-1)
knn.fit(X_train_scaled, y_train)
```

- `KNeighborsClassifier` được khởi tạo với các tham số: - `n_neighbors=5`: Sử dụng 5 láng giềng gần nhất để phân loại. - `weights='distance'`: Các láng giềng gần hơn sẽ có trọng số lớn hơn trong quá trình phân loại. - `metric='euclidean'`: Dùng khoảng cách Euclid để tính toán sự tương tự giữa các đối tượng. - `n_jobs=-1`: Sử dụng tất cả các luồng CPU để tính toán, giúp tăng tốc độ xử lý. - Mô hình được huấn luyện trên dữ liệu `X_train_scaled` và mục tiêu `y_train`.

Bước 5: Đánh giá mô hình

```
y_pred = knn.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
```

Mô hình được đánh giá bằng cách dự đoán các nhãn `y_pred` cho dữ liệu kiểm tra `X_test_scaled`. Độ chính xác (`accuracy`) được tính toán từ tỷ lệ giữa số lượng dự đoán đúng và tổng số mẫu kiểm tra.

Bước 6: Lưu mô hình và scaler

```
with open('knn_model.pkl', 'wb') as f:
    pickle.dump((knn, scaler), f)
```

Mô hình huấn luyện và `scaler` được lưu trữ dưới dạng tệp `.pkl` để có thể tái sử dụng trong các ứng dụng hoặc quy trình phân tích khác.

Bước 7: Hiển thị kết quả trực quan

```
plt.figure(figsize=(15, 10))

# Confusion Matrix
plt.subplot(2, 2, 1)
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')

# Class Distribution
plt.subplot(2, 2, 2)
pd.Series(y_train).value_counts().head(10).plot(kind='bar')
plt.title('Top 10 Brands (Training)')
```

```
plt.xticks(rotation=45)

# Prediction Distribution
plt.subplot(2, 2, 3)
pd.Series(y_pred).value_counts().head(10).plot(kind='bar')
plt.title('Top 10 Predicted Brands')
plt.xticks(rotation=45)

plt.tight_layout()
plt.savefig('knn_results.png')
plt.show()
```

Kết quả của mô hình được trực quan hóa:

- **Ma trận nhầm lẫn (Confusion Matrix)**: Giúp đánh giá chính xác mức độ sai số của mô hình.
- **Phân phối lớp**: Thể hiện số lượng mẫu thuộc mỗi lớp trong dữ liệu huấn luyện và kết quả dự đoán.

Các biểu đồ được lưu dưới định dạng ảnh (`.png`) và hiển thị ra màn hình.

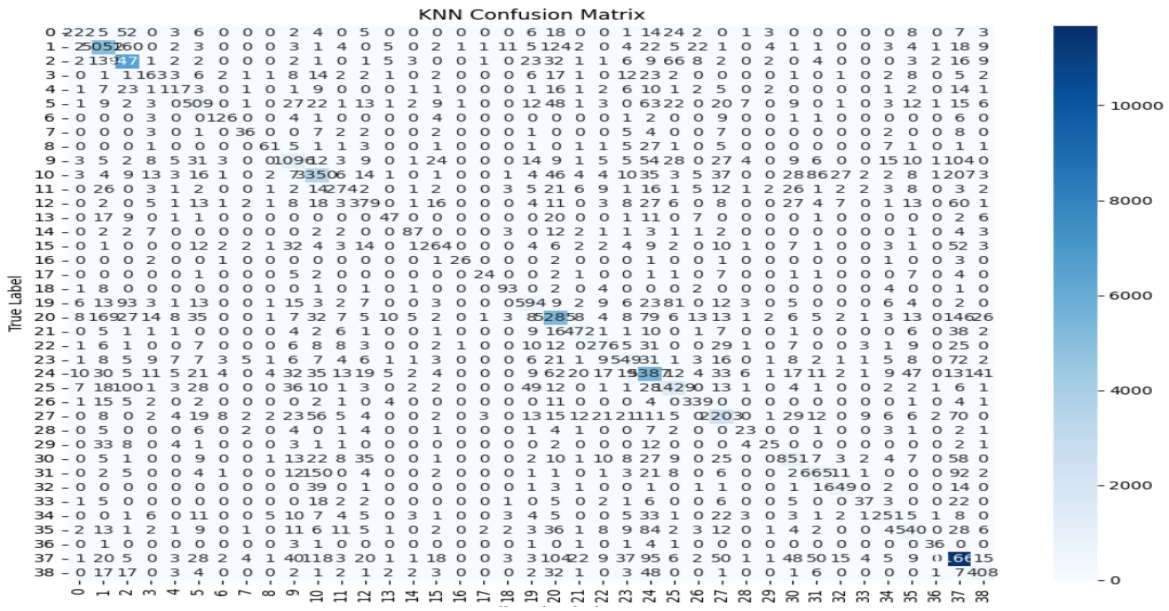
Kết quả

Hiệu suất mô hình đạt được thông qua các số liệu chính như sau:

- **Độ chính xác:** `0.85`.
- **Báo cáo phân loại:** Độ chính xác, độ thu hồi và F1-score được hiển thị trong báo cáo phân loại.

accuracy			0.85	58752
macro avg	0.75	0.67	0.70	58752
weighted avg	0.85	0.85	0.85	58752

- Ma trận nhầm lẫn:



2 Mô hình: Random Forest

2.1 Giới thiệu

Mô hình Random Forest được sử dụng để dự đoán thương hiệu xe hơi dựa trên các đặc trưng như loại xe, hộp số, nhiên liệu, và các thông số khác. Bài báo cáo trình bày quy trình xây dựng và huấn luyện mô hình từ xử lý dữ liệu đến đánh giá hiệu suất.

2.2 Quy trình thực hiện

Huấn luyện mô hình Random Forest

Mô hình Random Forest được thiết lập với các siêu tham số như sau:

- **n_estimators**: 200 (số lượng cây).
- **max_depth**: 20 (chiều sâu tối đa của mỗi cây).
- **min_samples_split**: 5 (số lượng mẫu tối thiểu để chia nút).
- **min_samples_leaf**: 2 (số lượng mẫu tối thiểu trong một lá).

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(
    n_estimators=200,
    max_depth=20,
    min_samples_split=5,
    min_samples_leaf=2,
```

```
    random_state=42,
    n_jobs=-1
)
rf.fit(X_train, y_train)
```

Đánh giá mô hình

Hiệu suất mô hình được đánh giá bằng độ chính xác, ma trận nhầm lẫn, và báo cáo phân loại.

```
from sklearn.metrics import accuracy_score, classification_report,
    confusion_matrix

y_pred = rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print(classification_report(y_test, y_pred))
```

Trực quan hóa kết quả

Kết quả bao gồm:

- **Feature Importance:** Độ quan trọng của từng đặc trưng.
- **Confusion Matrix:** Ma trận nhầm lẫn.
- **Phân bố các lớp trong dữ liệu huấn luyện và dự đoán.**

```
import matplotlib.pyplot as plt
import seaborn as sns


plt.subplot(2, 2, 1)
feature_imp = pd.DataFrame({
    'feature': X_train.columns,
    'importance': rf.feature_importances_
}).sort_values('importance', ascending=False)
sns.barplot(x='importance', y='feature', data=feature_imp)
plt.title('Feature Importance')


plt.subplot(2, 2, 2)
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
```

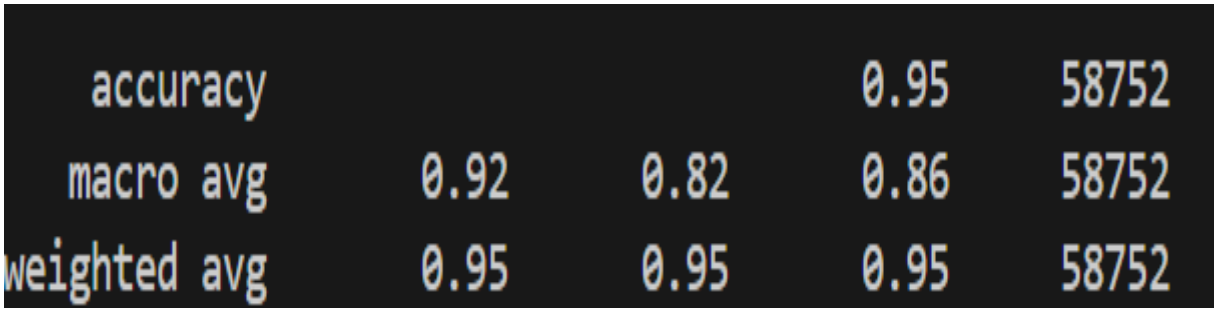
```
plt.title('Confusion Matrix')

plt.tight_layout()
plt.savefig('random_forest_results.png')
plt.show()
```

Kết quả

Hiệu suất mô hình đạt được thông qua các số liệu chính như sau:

- **Độ chính xác:** 0.95.
- **Báo cáo phân loại:** Độ chính xác, độ thu hồi và F1-score được hiển thị trong báo cáo phân loại.



accuracy			0.95	58752
macro avg	0.92	0.82	0.86	58752
weighted avg	0.95	0.95	0.95	58752

3 Mô hình:AdaBoost

3.1 Giới thiệu

Mô hình AdaBoost (Adaptive Boosting) là một phương pháp học máy thuộc nhóm ensemble learning, kết hợp nhiều mô hình yếu (weak learners), thường là các cây quyết định nông, để tạo ra một mô hình mạnh. AdaBoost hoạt động bằng cách phân bổ trọng số cao hơn cho các mẫu dữ liệu khó phân loại, từ đó giúp cải thiện khả năng phân loại tổng thể. Trong bài báo cáo này, mô hình AdaBoost được sử dụng để dự đoán thương hiệu xe hơi dựa trên các đặc trưng khác nhau.

3.2 Quy trình thực hiện

Huấn luyện mô hình AdaBoost

Mô hình AdaBoost được huấn luyện với các siêu tham số như sau:

- **Base Estimator:** *Decision Tree* với độ sâu tối đa là 3.
- **Số lượng mô hình:** 100.

- **Tốc độ học (Learning Rate):** 0.1.

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

base_dt = DecisionTreeClassifier(
    max_depth=3,
    min_samples_split=5,
    min_samples_leaf=2
)

ada = AdaBoostClassifier(
    estimator=base_dt,
    n_estimators=100,
    learning_rate=0.1,
    random_state=42
)
ada.fit(X_train, y_train)
```

Listing 2: Huấn luyện mô hình AdaBoost

Đánh giá mô hình

Hiệu suất mô hình được đánh giá bằng độ chính xác, ma trận nhầm lẫn, và báo cáo phân loại.

```
from sklearn.metrics import accuracy_score, classification_report,
    confusion_matrix

y_pred = ada.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print(classification_report(y_test, y_pred))
```

Listing 3: Đánh giá mô hình

Trực quan hóa kết quả

Kết quả trực quan bao gồm:

- **Feature Importance:** Tầm quan trọng của từng đặc trưng.
- **Confusion Matrix:** Ma trận nhầm lẫn.
- **Phân bố lớp:** Phân bố các nhãn mục tiêu trong tập huấn luyện và dự đoán.

```python
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(15, 10))

# Feature Importance
plt.subplot(2, 2, 1)
feature_imp = pd.DataFrame({
    'feature': X_train.columns,
    'importance': ada.feature_importances_
}).sort_values('importance', ascending=False)
sns.barplot(x='importance', y='feature', data=feature_imp)
plt.title('Feature Importance')

# Confusion Matrix
plt.subplot(2, 2, 2)
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')

plt.tight_layout()
plt.savefig('adaboost_results.png')
plt.show()
```

Listing 4: Trực quan hóa kết quả

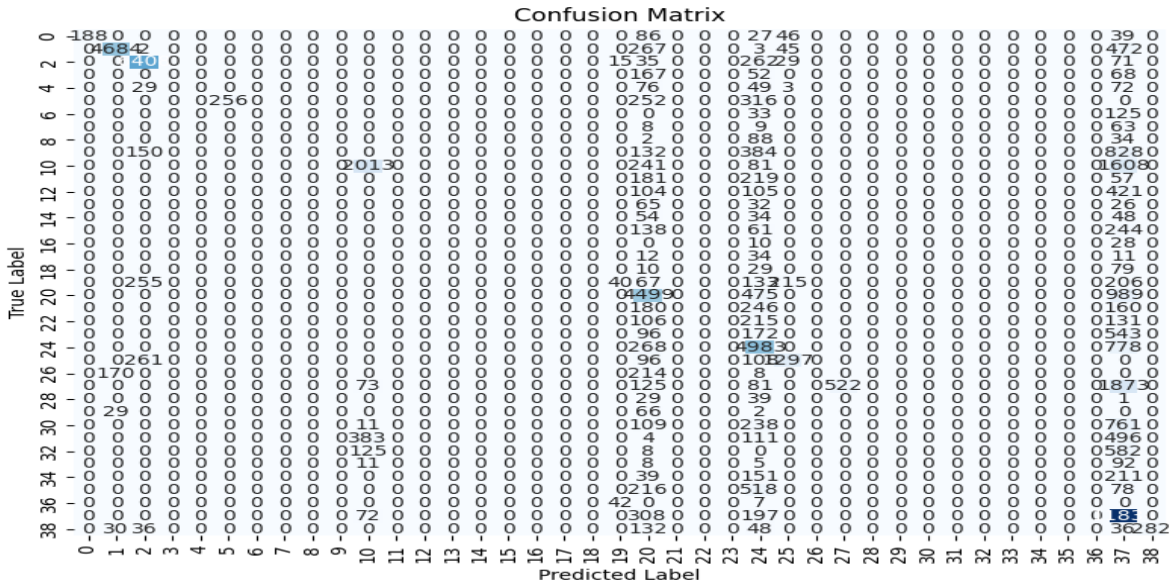
Kết quả

Hiệu suất mô hình đạt được thông qua các số liệu chính như sau:

- **Độ chính xác:** 0.63.
- **Báo cáo phân loại:** Độ chính xác, độ thu hồi và F1-score được hiển thị trong báo cáo phân loại.

accuracy			0.63	58752
macro avg	0.24	0.18	0.19	58752
weighted avg	0.57	0.63	0.56	58752

- Ma trận nhầm lẫn:



4 Mô hình: AdaBoost cải tiến

Huấn luyện mô hình AdaBoost cải tiến

Mô hình AdaBoost được cải tiến với các thay đổi về siêu tham số:

- **Base Estimator (Decision Tree):**
  - Tăng `max_depth` lên 5.
  - `min_samples_split` tăng lên 10.
  - `min_samples_leaf` giảm xuống 4 để tránh quá khớp.
- **AdaBoost Parameters:**
  - `n_estimators` tăng từ 100 lên 200.
  - `learning_rate` giảm xuống 0.05 để cải thiện quá trình hội tụ.

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

base_dt = DecisionTreeClassifier(
    max_depth=5,
    min_samples_split=10,
    min_samples_leaf=4
)

```

```
ada = AdaBoostClassifier(
    estimator=base_dt,
    n_estimators=200,
    learning_rate=0.05,
    random_state=42
)
ada.fit(X_train, y_train)
```

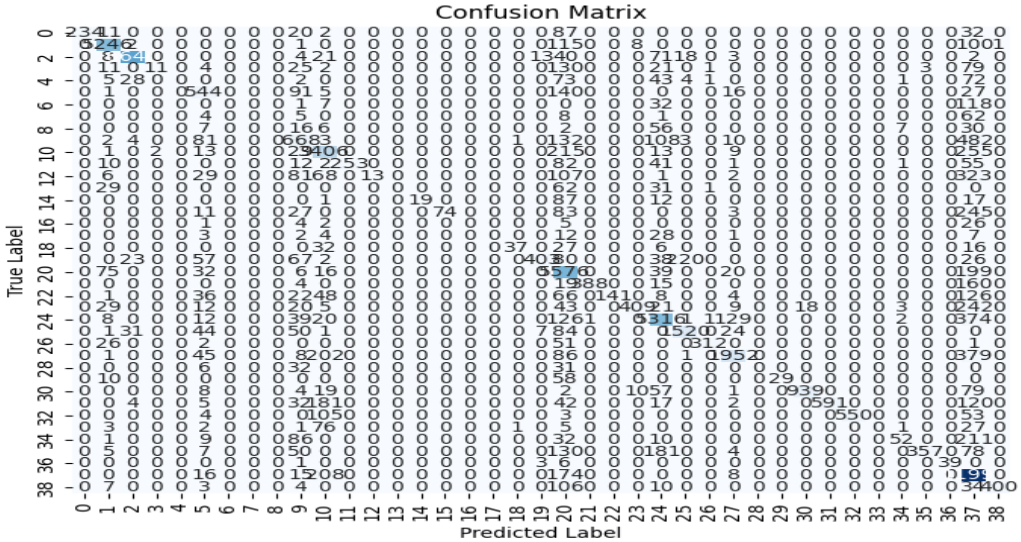
Kết quả

Hiệu suất mô hình đạt được thông qua các số liệu chính như sau:

- **Độ chính xác:** 0.63.
- **Báo cáo phân loại:** Độ chính xác, độ thu hồi và F1-score được hiển thị trong báo cáo phân loại.

accuracy			0.82	58752
macro avg	0.70	0.45	0.50	58752
weighted avg	0.83	0.82	0.80	58752

- Ma trận nhầm lẫn:



5 Mô hình:GradientBoosting

5.1 Giới thiệu

Gradient Boosting là một kỹ thuật học máy mạnh mẽ, thường được sử dụng cho các bài toán phân loại và hồi quy. Mô hình này hoạt động bằng cách xây dựng một chuỗi các cây quyết định, trong đó mỗi cây mới được xây dựng để cải thiện các lỗi của cây trước đó. Dưới đây là các bước thực hiện mô hình.

5.2 Các bước thực hiện

Khởi Tạo Mô Hình Gradient Boosting

Chúng ta sẽ khởi tạo mô hình Gradient Boosting với số lượng cây quyết định là 100.

```
# 4. Initialize the Gradient Boosting model
model = GradientBoostingClassifier(n_estimators=100, random_state=42)
```

Huấn Luyện Mô Hình

Mô hình sẽ được huấn luyện trên dữ liệu huấn luyện.

```
# 5. Train the model
print("Training the model...")
model.fit(X_train, y_train)
```

Đánh Giá Mô Hình

Sau khi huấn luyện, chúng ta sẽ đánh giá mô hình bằng cách dự đoán trên dữ liệu kiểm tra và tính toán độ chính xác.

```
# 6. Evaluate the model
print("Evaluating the model...")
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
```

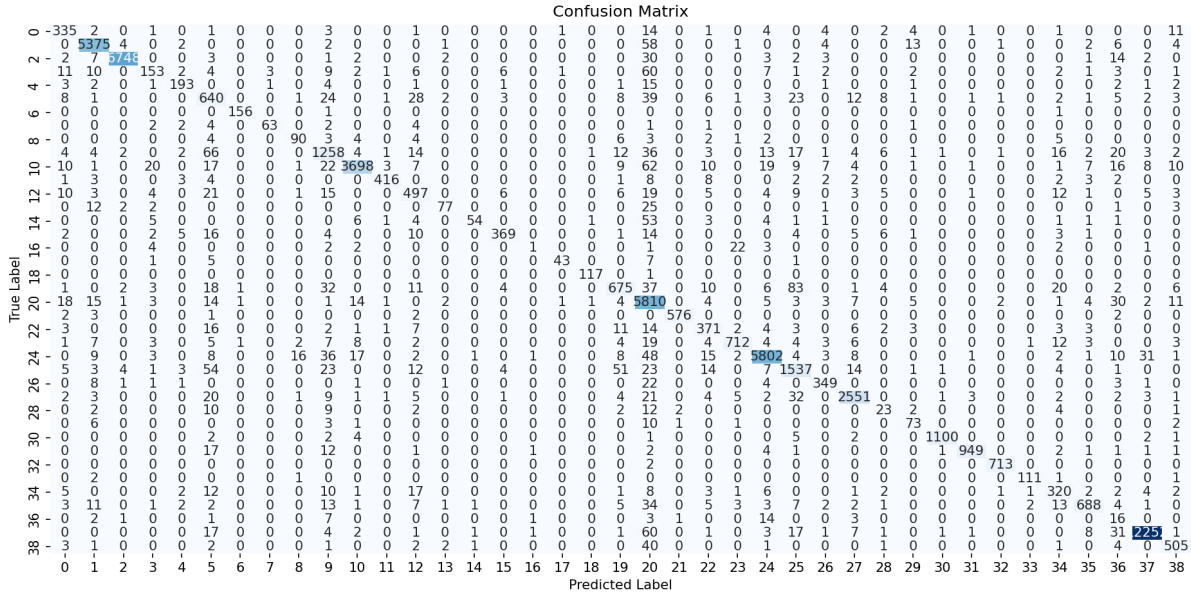
Kết quả

Hiệu suất mô hình đạt được thông qua các số liệu chính như sau:

- **Độ chính xác:** `0.94`.
- **Báo cáo phân loại:** Độ chính xác, độ thu hồi và F1-score được hiển thị trong báo cáo phân loại.

accuracy			0.87	58752
macro avg	0.85	0.57	0.64	58752
weighted avg	0.88	0.87	0.86	58752

- Ma trận nhầm lẫn:



6 Mô hình:Support Vector Machine (SVM)

6.1 Giới thiệu

Support Vector Machine (SVM) là một trong những thuật toán học máy phổ biến cho các bài toán phân loại. Mô hình này tìm kiếm một siêu phẳng (hyperplane) tối ưu để phân chia các lớp trong không gian đặc trưng. Dưới đây là các bước thực hiện mô hình SVM.

6.2 Các bước thực hiện

Định Nghĩa Mô Hình SVM

Chúng ta sẽ định nghĩa mô hình SVM và chuẩn bị cho việc tinh chỉnh các tham số.

```
# 2. Define the SVM model
print("\nTuning hyperparameters with GridSearchCV...")
svc = SVC()
```

Tinh Chỉnh Tham Số với GridSearchCV

Sử dụng GridSearchCV để tìm kiếm các tham số tốt nhất cho mô hình SVM.

```
# Define parameter grid
param_grid = {
    'C': [0.1, 1, 10, 100, 1000],
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'gamma': ['scale', 'auto', 0.001, 0.01, 0.1, 1],
    'degree': [2, 3, 4]  # Only relevant for 'poly' kernel
}

# Grid Search
grid_search = GridSearchCV(svc, param_grid, cv=5, scoring='accuracy', n_jobs
    =-1, verbose=1)
grid_search.fit(X_train_scaled, y_train)
```

Đánh Giá Mô Hình Đã Tinh Chỉnh

Sau khi tìm được mô hình tốt nhất, chúng ta sẽ đánh giá nó trên tập kiểm tra.

```
# 3. Evaluate the tuned model
y_pred = best_model.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)

print(f"\nAccuracy on test data: {accuracy:.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=label_encoder.
    classes_))
```

Ma Trận Nhầm Lẫn

Chúng ta sẽ tạo ma trận nhầm lẫn để trực quan hóa kết quả phân loại.

```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.
    classes_, yticklabels=label_encoder.classes_)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.savefig("svm_confusion_matrix.png")
plt.show()
```

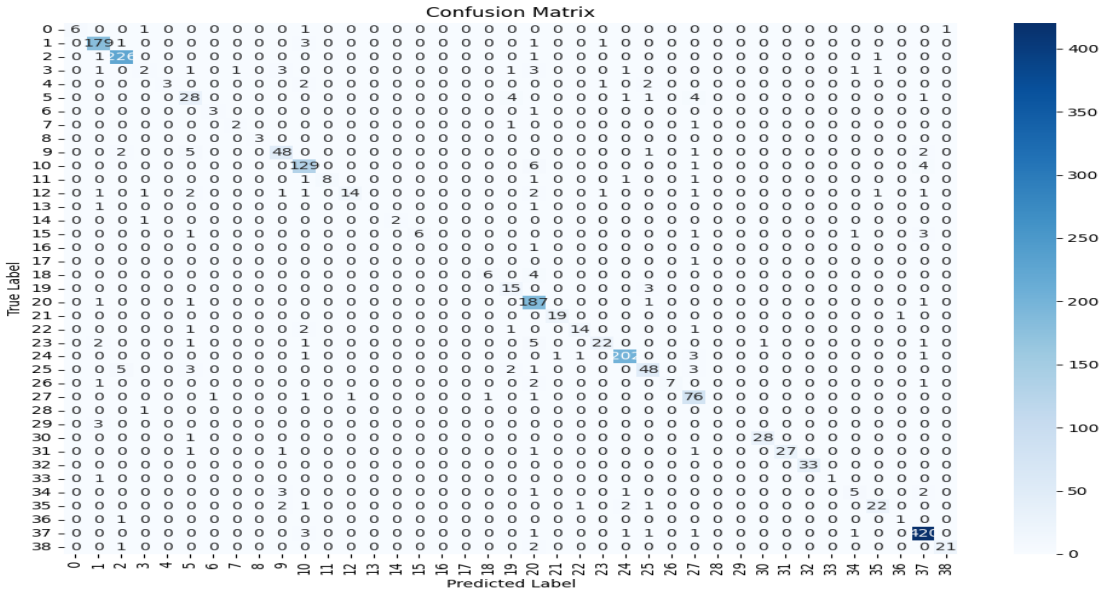
Kết quả

Hiệu suất mô hình đạt được thông qua các số liệu chính như sau:

- **Độ chính xác:** `0.91`.
- **Báo cáo phân loại:** Độ chính xác, độ thu hồi và F1-score được hiển thị trong báo cáo phân loại.

accuracy			0.91	2000
macro avg	0.75	0.64	0.68	2000
weighted avg	0.90	0.91	0.90	2000

- Ma trận nhầm lẫn:



7 Mô hình: XGBoost

7.1 Giới thiệu

XGBoost (Extreme Gradient Boosting) là một thuật toán học máy mạnh mẽ, thường được sử dụng cho các bài toán phân loại và hồi quy. Mô hình này sử dụng phương pháp boosting để cải thiện độ chính xác của dự đoán. Dưới đây là các bước thực hiện mô hình XGBoost.

7.2 Các bước thực hiện

Huấn Luyện Mô Hình XGBoost

Chúng ta sẽ khởi tạo và huấn luyện mô hình XGBoost với các tham số đã định nghĩa.

```
# Train XGBoost
print("\nTraining XGBoost model...")
start_time = time.time()

xgb = XGBClassifier(
    n_estimators=200,
    max_depth=7,
    learning_rate=0.1,
    objective='multi:softmax',
    num_class=n_classes,
    random_state=42,
    n_jobs=-1
)

xgb.fit(X_train, y_train)
train_time = time.time() - start_time
```

Đánh Giá Mô Hình

Sau khi huấn luyện, chúng ta sẽ đánh giá mô hình bằng cách dự đoán trên dữ liệu kiểm tra và tính toán độ chính xác.

```
# Evaluate
y_pred = xgb.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"\nResults:")
print(f"Training time: {train_time:.2f} seconds")
print(f"Accuracy: {accuracy:.4f}")
```

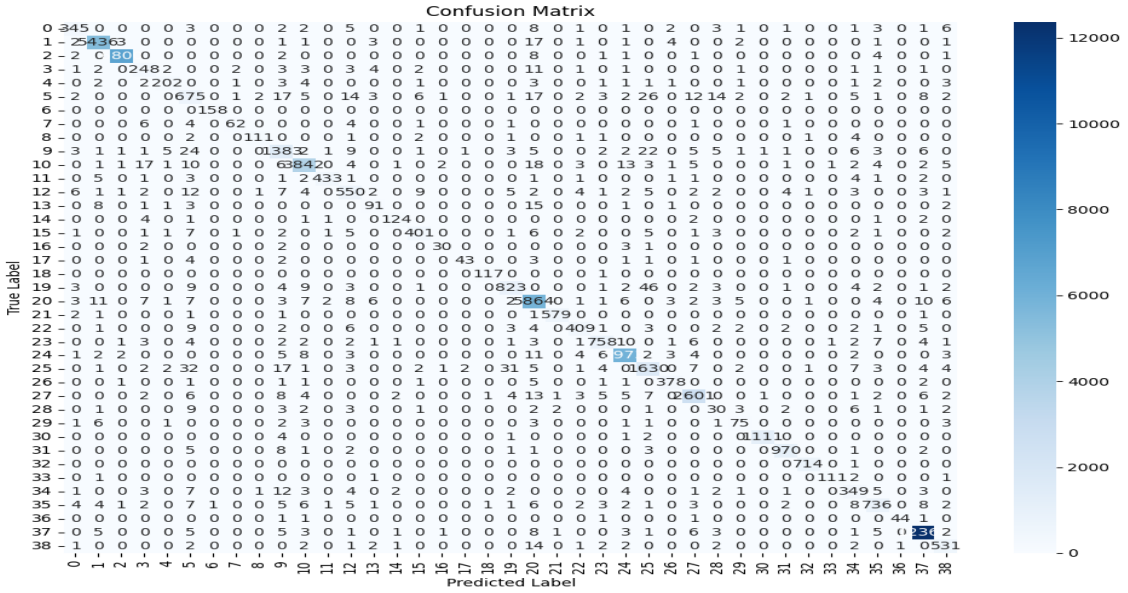
Kết quả

Hiệu suất mô hình đạt được thông qua các số liệu chính như sau:

- **Độ chính xác:** 0.97.
- **Báo cáo phân loại:** Độ chính xác, độ thu hồi và F1-score được hiển thị trong báo cáo phân loại.

accuracy			0.97	58752
macro avg	0.92	0.90	0.91	58752
weighted avg	0.97	0.97	0.97	58752

- Ma trận nhầm lẫn:



8 Mô hình: LightGBM

8.1 Giới thiệu về mô hình LightGBM

LightGBM (Light Gradient Boosting Machine) là một thuật toán học máy mạnh mẽ được phát triển bởi Microsoft, nổi bật với khả năng xử lý dữ liệu lớn và tốc độ huấn luyện nhanh. Mô hình này sử dụng phương pháp boosting để cải thiện độ chính xác của dự đoán bằng cách kết hợp nhiều cây quyết định.

8.2 Các bước thực hiện

1. **Huấn luyện mô hình LightGBM**: Mô hình được huấn luyện với các tham số được điều chỉnh.

```
lgb_model = lgb.LGBMClassifier(
    n_estimators=1000,
```

```
        max_depth=15,
        learning_rate=0.01,
        min_child_samples=20,
        subsample=0.8,
        random_state=42
    )
    lgb_model.fit(X_train, y_train)
```

2. **Đánh giá mô hình**: Mô hình được đánh giá bằng độ chính xác và báo cáo phân loại.

```
    y_pred = lgb_model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(classification_report(y_test, y_pred))
```

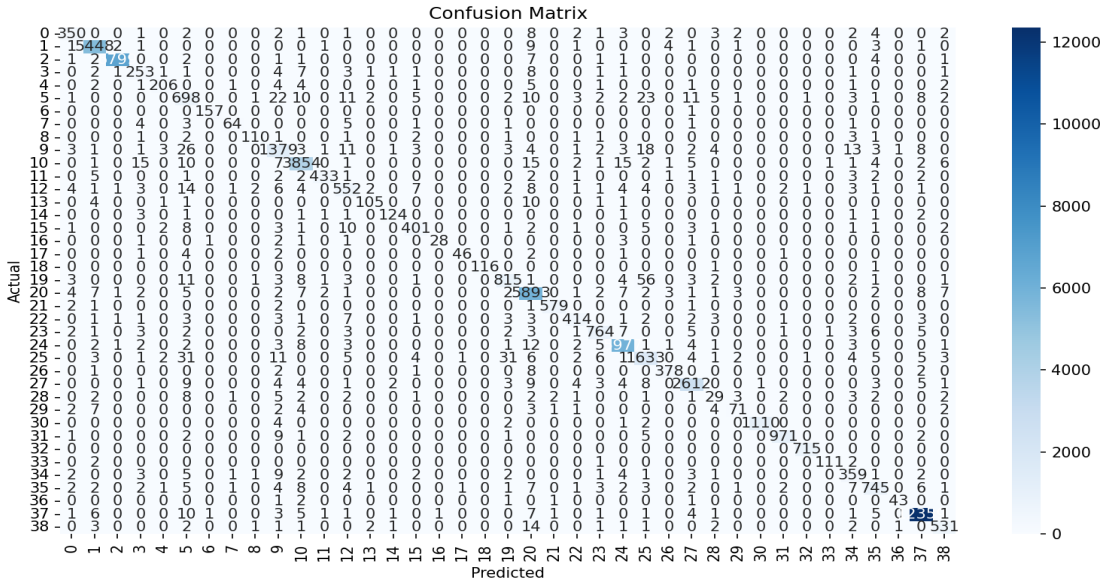
Kết quả

Hiệu suất mô hình đạt được thông qua các số liệu chính như sau:

- **Độ chính xác:** 0.97.
- **Báo cáo phân loại:** Độ chính xác, độ thu hồi và F1-score được hiển thị trong báo cáo phân loại.

accuracy			0.97	58752
macro avg	0.94	0.91	0.92	58752
weighted avg	0.97	0.97	0.97	58752

- Ma trận nhầm lẫn:



9 Mô hình: Decision Tree

9.1 Giới thiệu về Mô hình Decision Tree

Mô hình Decision Tree là một trong những thuật toán học máy phổ biến, được sử dụng để phân loại và hồi quy. Mô hình này hoạt động bằng cách chia nhỏ dữ liệu thành các nhánh dựa trên các điều kiện khác nhau, giúp đưa ra quyết định cuối cùng. Trong bài báo cáo này, chúng tôi sẽ trình bày quy trình xây dựng và đánh giá mô hình Decision Tree cho bài toán phân loại thương hiệu xe hơi.

9.2 Các bước thực hiện

Huấn luyện mô hình

Mô hình Decision Tree được huấn luyện với các tham số đã được điều chỉnh.

```
dt = DecisionTreeClassifier(
    criterion='entropy',
    max_depth=30,
    min_samples_split=10,
    min_samples_leaf=5,
    max_features='sqrt',
    random_state=42,
    class_weight='balanced'
)
dt.fit(X_train, y_train)
```

Đánh giá mô hình

Mô hình được đánh giá bằng cách tính toán độ chính xác và in ra báo cáo phân loại.

```
y_pred = dt.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(classification_report(y_test, y_pred))
```

Tạo hình ảnh trực quan

Các hình ảnh trực quan được tạo ra để minh họa tầm quan trọng của các đặc trưng và ma trận nhầm lẫn.

```
plot_tree_visualization(dt, X_train.columns)
plot_feature_importance(dt, X_train.columns)
plot_confusion_matrix(y_test, y_pred)
```

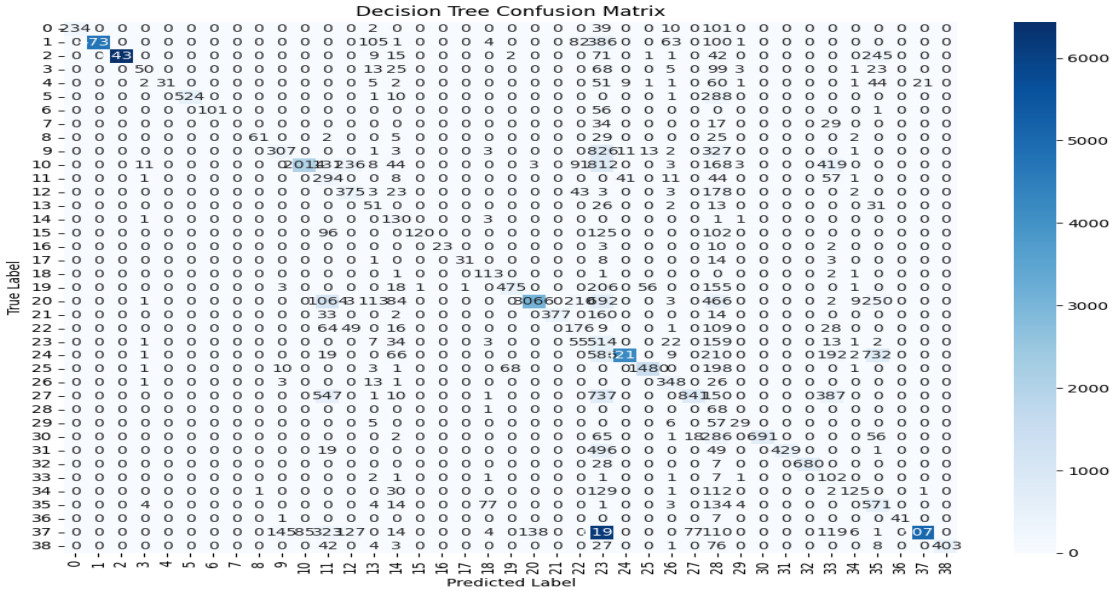
Kết quả

Hiệu suất mô hình đạt được thông qua các số liệu chính như sau:

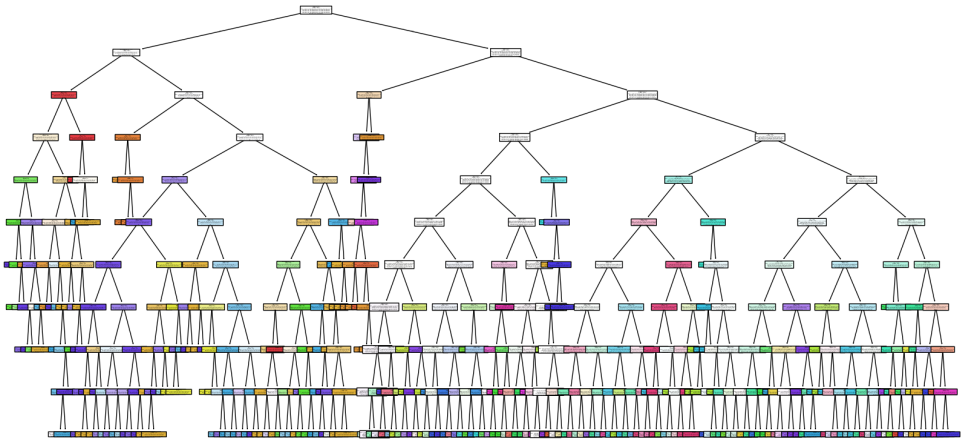
- **Độ chính xác:** `0.77`.
- **Báo cáo phân loại:** Độ chính xác, độ thu hồi và F1-score được hiển thị trong báo cáo phân loại.

accuracy			0.77	58752
macro avg	0.54	0.74	0.59	58752
weighted avg	0.84	0.77	0.79	58752

- Ma trận nhầm lẫn:



- Hình minh họa cây quyết định:



Hình 9.2.1: Caption

10 Mô hình:Bagging Classifier

10.1 Giới thiệu về Mô hình Bagging Classifier

Mô hình Bagging Classifier là một kỹ thuật học máy mạnh mẽ, được sử dụng để cải thiện độ chính xác của các mô hình phân loại bằng cách kết hợp nhiều mô hình cơ sở. Kỹ thuật này hoạt động bằng cách huấn luyện nhiều mô hình trên các mẫu ngẫu nhiên của dữ liệu huấn luyện và sau đó kết hợp các dự đoán của chúng để đưa ra dự đoán cuối

cùng. Trong báo cáo này, chúng tôi sẽ trình bày quy trình xây dựng và đánh giá mô hình Bagging Classifier cho bài toán phân loại thương hiệu xe hơi.

10.2 Các bước thực hiện

Tạo mô hình cơ sở

Mô hình Decision Tree được sử dụng làm mô hình cơ sở cho Bagging.

```
base_estimator = DecisionTreeClassifier(
    max_depth=10,
    min_samples_split=5,
    min_samples_leaf=2,
    random_state=42
)
```

Huấn luyện Bagging Classifier

Mô hình Bagging Classifier được huấn luyện với các tham số đã được điều chỉnh.

```
bagging = BaggingClassifier(
    estimator=base_estimator,
    n_estimators=100,
    max_samples=0.8,
    max_features=0.8,
    bootstrap=True,
    bootstrap_features=True,
    n_jobs=-1,
    random_state=42
)
bagging.fit(X_train, y_train)
```

Đánh giá mô hình

Mô hình được đánh giá bằng cách tính toán độ chính xác và in ra báo cáo phân loại.

```
y_pred = bagging.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(classification_report(y_test, y_pred))
```

Tạo ma trận nhầm lẫn

Ma trận nhầm lẫn được tạo ra để đánh giá hiệu suất của mô hình.

```
plot_confusion_matrix(y_test, y_pred)
```

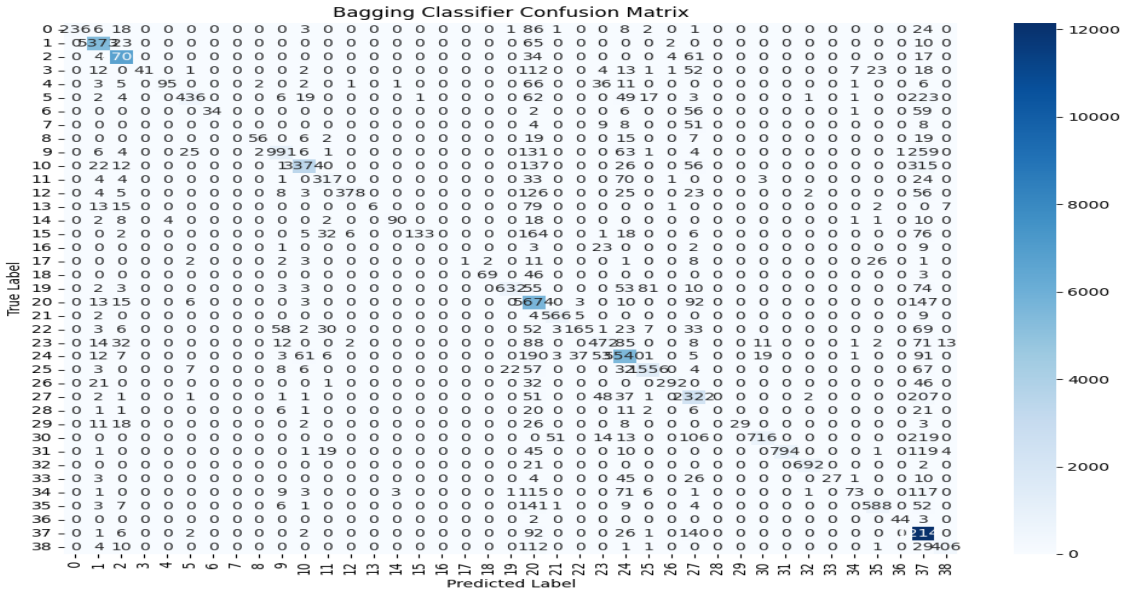
Kết quả

Hiệu suất mô hình đạt được thông qua các số liệu chính như sau:

- **Độ chính xác:** `0.87`.
- **Báo cáo phân loại:** Độ chính xác, độ thu hồi và F1-score được hiển thị trong báo cáo phân loại.

accuracy			0.87	58752
macro avg	0.85	0.57	0.64	58752
weighted avg	0.88	0.87	0.86	58752

- Ma trận nhầm lẫn:



V Đóng gói sản phẩm

Ta lưu các mô hình đã được huấn luyện vào file pkl.

```
with open('adaboost_model_improve.pkl', 'wb') as f:
    pickle.dump(model_data, f)
```

Ở đây ta dùng thư viện stream lit để tạo giao diện web và tải lên mô hình đã được lưu trong các file pkl ở trên.

-Giao diện :