A3_st126488

October 2, 2025

[1]:
```python
# A3 - Predicting Car Prices (Classification with CV and MLflow) - IMPROVED
# Name: Alston Alvares      Student ID: st126488

import os
import json
import joblib
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import time
import mlflow
import warnings
from datetime import datetime
from mlflow.exceptions import MlflowException


from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.metrics import confusion_matrix, classification_report,␣
 ↪accuracy_score

# Import the custom Logistic Regression model
from custom_classifier import LogisticRegression

# Suppress warnings for a cleaner output
warnings.filterwarnings('ignore')

# --- FIX: Add MLflow Server Authentication ---
# IMPORTANT: Replace 'YOUR_USERNAME' and 'YOUR_PASSWORD' with your actual␣
 ↪credentials
os.environ['MLFLOW_TRACKING_USERNAME'] = 'admin'
os.environ['MLFLOW_TRACKING_PASSWORD'] = 'password'
```

```python
# --- MLflow Setup ---
# Objective 1: Change the tracking_uri to the remote server
mlflow.set_tracking_uri("http://mlflow.ml.brain.cs.ait.ac.th/")

# Objective 1: Set the experiment name as per the requirement
experiment_name = "st126488-a3"
mlflow.set_experiment(experiment_name)
print(f" MLflow configured. Tracking to experiment: '{experiment_name}' on the remote server.")


# --- Custom Classification Metrics (remains the same) ---
def custom_classification_report(y_true, y_pred, class_names):
    # This function remains unchanged...
    cm = confusion_matrix(y_true, y_pred)
    n_classes = len(class_names)
    accuracy = np.sum(np.diag(cm)) / np.sum(cm)
    precisions, recalls, f1_scores, supports = [], [], [], []
    for i in range(n_classes):
        TP = cm[i, i]; FP = np.sum(cm[:, i]) - TP; FN = np.sum(cm[i, :]) - TP
        precision = TP / (TP + FP) if (TP + FP) > 0 else 0
        recall = TP / (TP + FN) if (TP + FN) > 0 else 0
        f1 = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0
        support = np.sum(cm[i, :])
        precisions.append(precision); recalls.append(recall); f1_scores.append(f1); supports.append(support)
    macro_precision = np.mean(precisions); macro_recall = np.mean(recalls); macro_f1 = np.mean(f1_scores)
    weighted_precision = np.sum(np.array(precisions) * np.array(supports)) / np.sum(supports)
    weighted_recall = np.sum(np.array(recalls) * np.array(supports)) / np.sum(supports)
    weighted_f1 = np.sum(np.array(f1_scores) * np.array(supports)) / np.sum(supports)
    print("="*55); print("         Custom Classification Report (from Scratch)"); print("="*55); print(f"{'':<12}{'precision':<12}{'recall':<12}{'f1-score':<12}{'support'}"); print("-"*55)
    for i in range(n_classes): print(f"{class_names[i]:<12}{precisions[i]:<12.2f}{recalls[i]:<12.2f}{f1_scores[i]:<12.2f}{supports[i]}")
    print("-"*55); print(f"accuracy {accuracy:.2f}"); print(f"\nmacro avg    {macro_precision:<12.2f}{macro_recall:<12.2f}{macro_f1:<12.2f}"); print(f"weighted avg  {weighted_precision:<12.2f}{weighted_recall:<12.2f}{weighted_f1:<12.2f}"); print("="*55)
```

```
# --- Data Loading & Preprocessing ---
DATA_PATH = "Cars.csv"
df = pd.read_csv(DATA_PATH)
df.columns = df.columns.str.lower()
owner_map = {'First Owner': 1, 'Second Owner': 2, 'Third Owner': 3, 'Fourth &␣
↪Above Owner': 4, 'Test Drive Car': 5}
df['owner'] = df['owner'].astype(str).str.strip().replace(owner_map)
df = df[~df['fuel'].isin(['CNG', 'LPG']) & (df['owner'] != 5)]
df['mileage'] = pd.to_numeric(df['mileage'].astype(str).str.extract(r'([\d\.
↪]+)')[0], errors='coerce')
df['brand'] = df['name'].astype(str).str.split().str[0]

# --- FEATURE ENGINEERING ---
current_year = datetime.now().year
df['car_age'] = current_year - df['year']
# Create a new feature for kilometers per year to better represent car usage
# Add 1 to car_age to prevent division by zero for new cars
df['km_per_year'] = df['km_driven'] / (df['car_age'] + 1)


# Define feature set with the new 'km_per_year' feature
X = df[['car_age', 'km_driven', 'mileage', 'owner', 'brand', 'km_per_year']]
y_continuous = df['selling_price']

df_model = pd.concat([X, y_continuous], axis=1).dropna()
X = df_model.drop(columns=['selling_price']).reset_index(drop=True)
y_continuous = df_model['selling_price'].reset_index(drop=True)
y, bin_edges = pd.qcut(y_continuous, q=4, retbins=True, labels=False)

# --- Class Imbalance Check ---
print("--- Class Distribution Check ---")
print("Number of samples in each class after using pd.qcut:")
print(y.value_counts().sort_index())
plt.figure(figsize=(8, 5))
sns.countplot(x=y, palette='viridis')
plt.title('Distribution of Car Price Classes')
plt.xlabel('Price Class')
plt.ylabel('Number of Cars')
plt.show()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42, stratify=y)


# --- Feature Preprocessing Definition ---
# Update num_cols to include the new 'km_per_year' feature
```

```python
num_cols = ['car_age', 'km_driven', 'mileage', 'owner', 'km_per_year']
cat_cols = ['brand']
preprocessor = ColumnTransformer([
    ('num', Pipeline([('imputer', SimpleImputer(strategy='median')), ('scaler', StandardScaler())]), num_cols),
    ('cat', OneHotEncoder(handle_unknown='ignore', sparse_output=False), cat_cols)
], remainder='drop')


# --- Hyperparameter Tuning with Cross-Validation and MLflow ---
print("\n  Starting Fast & Efficient Hyperparameter Tuning (Fixed Regularization)...")
start_time = time.time()

# --- Hyperparameters Search Space (Fixed as per user request) ---
learning_rates = [0.01, 0.001, 0.0001]
fixed_lambda = 0.1 # Use a fixed lambda value for Ridge regularization
num_iterations = 30000 # Balanced for speed and performance

best_cv_accuracy = -1
best_params = {}
all_results = []

# --- Use standard 5-fold CV for speed ---
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

for lr in learning_rates:
    with mlflow.start_run(run_name=f"lr_{lr}_lambda_{fixed_lambda}"):
        mlflow.log_params({"learning_rate": lr, "lambda": fixed_lambda, "num_iterations": num_iterations})
        fold_accuracies = []
        print(f"\n--- CV for lr={lr} (lambda={fixed_lambda}) ---")

        for fold, (train_idx, val_idx) in enumerate(skf.split(X_train, y_train)):
            X_train_fold, X_val_fold = X_train.iloc[train_idx], X_train.iloc[val_idx]
            y_train_fold, y_val_fold = y_train.iloc[train_idx], y_train.iloc[val_idx]

            X_train_fold_processed = preprocessor.fit_transform(X_train_fold)
            X_val_fold_processed = preprocessor.transform(X_val_fold)

            # Instantiate model with reg_type='l2' and the fixed lambda
```

```
            model = LogisticRegression(lr=lr, num_iter=num_iterations,␣
↪reg_type='l2', lambda_=fixed_lambda)
            model.fit(X_train_fold_processed, y_train_fold.to_numpy())

            y_val_pred = model.predict(X_val_fold_processed)
            accuracy = accuracy_score(y_val_fold, y_val_pred)
            fold_accuracies.append(accuracy)

        mean_cv_accuracy = np.mean(fold_accuracies)
        std_cv_accuracy = np.std(fold_accuracies)
        print(f"  => Average CV Accuracy over 5 folds: {mean_cv_accuracy:.4f}␣
↪(+/- {std_cv_accuracy:.4f})")
        mlflow.log_metrics({"mean_cv_accuracy": mean_cv_accuracy,␣
↪"std_cv_accuracy": std_cv_accuracy})
        all_results.append({'lr': lr, 'mean_cv_accuracy': mean_cv_accuracy})

        if mean_cv_accuracy > best_cv_accuracy:
            best_cv_accuracy = mean_cv_accuracy
            best_params = {'lr': lr} # Best params now only contains learning␣
↪rate
            print(f"  New best CV accuracy found: {best_cv_accuracy:.4f}")
            mlflow.set_tag("best_run", "True")

end_time = time.time()
print("\n" + "="*55); print("        Hyperparameter Tuning Complete!");␣
↪print(f"Total time taken: {(end_time - start_time) / 60:.2f} minutes");␣
↪print("="*55)
results_df = pd.DataFrame(all_results)
print("\nTuning Results Summary:"); print(results_df.
↪sort_values(by='mean_cv_accuracy', ascending=False))
print(f"\n  Best CV Parameters: {best_params}")
print(f"  Best Mean CV Accuracy: {best_cv_accuracy:.4f}")


# --- Final Model Training, Logging, and Registration (Objective 2) ---
print("\n--- Training Final Model and Registering with MLflow ---")

# Start a new run to log the final, best model
with mlflow.start_run(run_name=f"final_model_fixed_lambda_{fixed_lambda}") as␣
↪run:
    # Log the best parameters found during the search
    final_params = {**best_params, "lambda": fixed_lambda}
    print("Logging best parameters...")
    mlflow.log_params(final_params)

    # Define the final pipeline with the best hyperparameters
```

```
    final_pipeline = Pipeline([
        ('preprocessor', preprocessor),
        ('classifier', LogisticRegression(
            lr=best_params['lr'],
            num_iter=num_iterations,
            reg_type='l2', # Ensure Ridge penalty is used
            lambda_=fixed_lambda, # Use the fixed lambda value
            verbose=True
        ))
    ])

    # Train on the entire training set
    print("Training final model on full training data...")
    final_pipeline.fit(X_train, y_train)

    # Log model traces (learning curve) to MLflow
    print("\n--- Logging Model Training Trace to MLflow ---")
    final_model = final_pipeline.named_steps['classifier']
    for i, loss in enumerate(final_model.loss_history):
        # Log each loss value with a step. Step is i*100 because we record␣
↪every 100 iterations.
        mlflow.log_metric("training_loss", loss, step=i*100)

    # Evaluate on the hold-out test set and log metrics
    print("\nEvaluating final model on the test set...")
    y_pred_final = final_pipeline.predict(X_test)
    final_accuracy = accuracy_score(y_test, y_pred_final)
    mlflow.log_metric("final_test_accuracy", final_accuracy)

    print(f"\nFinal Model Test Accuracy: {final_accuracy:.4f}")
    custom_classification_report(y_test, y_pred_final, [f'Class {i}' for i in␣
↪range(y.nunique())])


    # --- Objective 2: Register the model and set alias ---
    model_name = "st126488-a3-model"
    print(f"\nLogging and registering the model as '{model_name}'...")

    # Step 1: Log the model and infer the signature
    mlflow.sklearn.log_model(
        sk_model=final_pipeline,
        artifact_path="model",
        input_example=X_train.head() # Add an input example to infer the␣
↪signature
    )

    # Step 2: Register the model from the artifact path of the current run
```

```
    run_id = run.info.run_id
    model_uri = f"runs:/{run_id}/model"

    registered_model_info = mlflow.register_model(
        model_uri=model_uri,
        name=model_name
    )
    new_version = registered_model_info.version

    # Add a delay and handle errors
    print("Waiting 5 seconds for the model registry to update...")
    time.sleep(5)

    try:
        # Use the new set_registered_model_alias function
        alias = "Staging"
        print(f"Setting alias '{alias}' for new version {new_version}...")
        client = mlflow.tracking.MlflowClient()
        client.set_registered_model_alias(
            name=model_name,
            alias=alias,
            version=new_version
        )
        print(f"  Successfully set alias '{alias}' for version {new_version} of model '{model_name}'.")

    except MlflowException as e:
        print(f"  WARNING: Model registration was successful, but setting the alias failed.")
        print(f"   This could be a temporary server glitch or a permissions issue.")
        print(f"   Please manually set the 'Staging' alias for version {new_version} of '{model_name}' in the MLflow UI.")
        print(f"   Error details: {e}")


# --- Save model locally for Dash app ---
DEPLOY_DIR = "deploy_assets"
os.makedirs(DEPLOY_DIR, exist_ok=True)
model_path = os.path.join(DEPLOY_DIR, "best_classifier_pipeline.joblib")
joblib.dump(final_pipeline, model_path)
print(f"\n  Final model pipeline saved locally to: {model_path}")

assets = {"num_cols": num_cols, "cat_cols": cat_cols, "class_names": [f'Class {i}' for i in range(y.nunique())], "price_bin_edges": list(bin_edges)}
assets_path = os.path.join(DEPLOY_DIR, "assets.json")
with open(assets_path, "w") as f: json.dump(assets, f, indent=4)
```

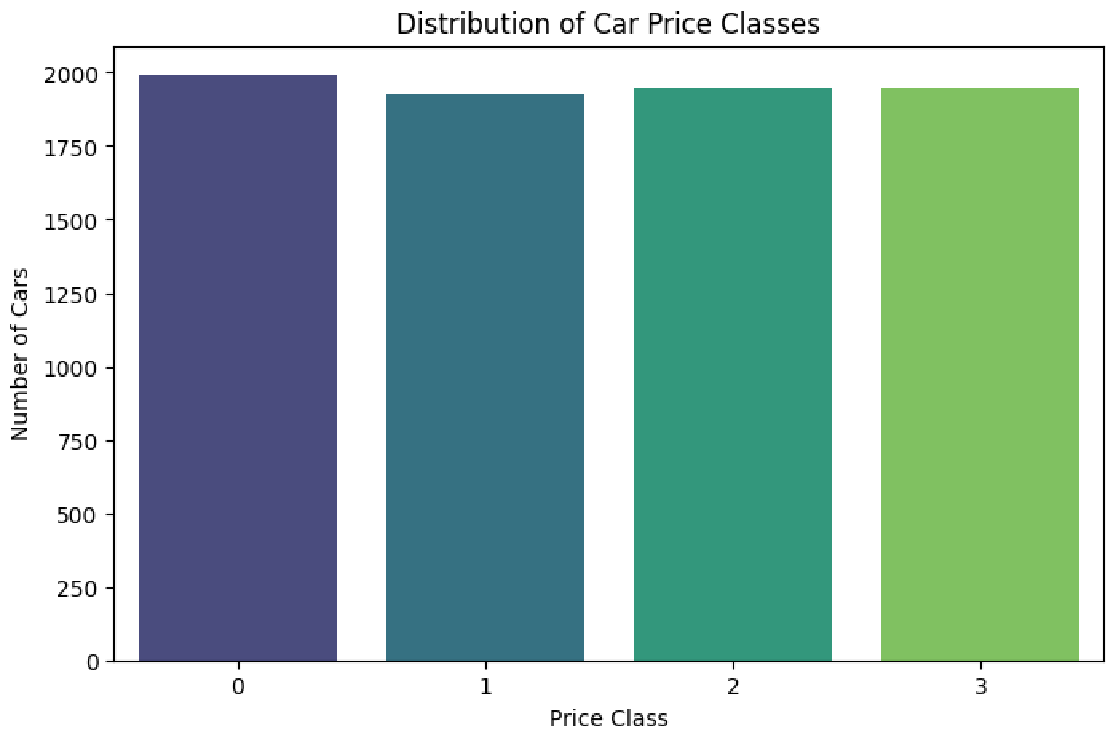
```python
print(f" Deployment assets saved locally to: {assets_path}")


# --- Inference Testing Section (Simplified Output) ---
print("\n" + "="*55)
print("         Inference Testing on Sample Data")
print("="*55)

loaded_model = joblib.load(model_path)
# Update test cases to include the new feature
test_cases = {
    "Budget Car": {'car_age': 14, 'km_driven': 150000, 'mileage': 18.0, 'owner':
 ↪ 3, 'brand': 'Maruti', 'km_per_year': 150000 / (14 + 1)},
    "Mid-Range Car": {'car_age': 6, 'km_driven': 60000, 'mileage': 22.0,␣
 ↪'owner': 2, 'brand': 'Honda', 'km_per_year': 60000 / (6 + 1)},
    "Premium Car": {'car_age': 2, 'km_driven': 20000, 'mileage': 15.0, 'owner':␣
 ↪1, 'brand': 'BMW', 'km_per_year': 20000 / (2 + 1)}
}

for name, data in test_cases.items():
    input_df = pd.DataFrame([data])
    predicted_class_index = loaded_model.predict(input_df)[0]
    predicted_class_name = assets['class_names'][predicted_class_index]
    print(f"\n--- Testing: {name} ---")
    print("Input Features:")
    print(input_df.to_string(index=False))
    print("\nPrediction:")
    print(f"  -> Predicted Category: {predicted_class_name}")
```

```
 MLflow configured. Tracking to experiment: 'st126488-a3' on the remote server.
--- Class Distribution Check ---
Number of samples in each class after using pd.qcut:
selling_price
0    1991
1    1927
2    1949
3    1947
Name: count, dtype: int64
```




```
 Starting Fast & Efficient Hyperparameter Tuning (Fixed Regularization)…

--- CV for lr=0.01 (lambda=0.1) ---
  => Average CV Accuracy over 5 folds: 0.5829 (+/- 0.0134)
 New best CV accuracy found: 0.5829
 View run lr_0.01_lambda_0.1 at: http://mlflow.ml.brain.cs.ait.ac.th/#/experime
nts/974791038746408189/runs/006770f7adf64522a71ac87cabf203d7
 View experiment at:
http://mlflow.ml.brain.cs.ait.ac.th/#/experiments/974791038746408189

--- CV for lr=0.001 (lambda=0.1) ---
  => Average CV Accuracy over 5 folds: 0.5362 (+/- 0.0118)
 View run lr_0.001_lambda_0.1 at: http://mlflow.ml.brain.cs.ait.ac.th/#/experim
ents/974791038746408189/runs/c35df113d71f4ebb872a99d5d49f2c9d
 View experiment at:
http://mlflow.ml.brain.cs.ait.ac.th/#/experiments/974791038746408189

--- CV for lr=0.0001 (lambda=0.1) ---
  => Average CV Accuracy over 5 folds: 0.4714 (+/- 0.0114)
 View run lr_0.0001_lambda_0.1 at: http://mlflow.ml.brain.cs.ait.ac.th/#/experi
ments/974791038746408189/runs/199fc92710714a0c839e1ebdf2793212
 View experiment at:
http://mlflow.ml.brain.cs.ait.ac.th/#/experiments/974791038746408189
```

```
=======================================================
        Hyperparameter Tuning Complete!
Total time taken: 3.69 minutes
=======================================================

Tuning Results Summary:
       lr  mean_cv_accuracy
0  0.0100          0.582946
1  0.0010          0.536234
2  0.0001          0.471445

  Best CV Parameters: {'lr': 0.01}
  Best Mean CV Accuracy: 0.5829

--- Training Final Model and Registering with MLflow ---
Logging best parameters…
Training final model on full training data…
Iteration 0, Loss: 1.3882
Iteration 1000, Loss: 1.0591
Iteration 2000, Loss: 1.0127
Iteration 3000, Loss: 0.9903
Iteration 4000, Loss: 0.9765
Iteration 5000, Loss: 0.9667
Iteration 6000, Loss: 0.9592
Iteration 7000, Loss: 0.9532
Iteration 8000, Loss: 0.9482
Iteration 9000, Loss: 0.9439
Iteration 10000, Loss: 0.9401
Iteration 11000, Loss: 0.9368
Iteration 12000, Loss: 0.9337
Iteration 13000, Loss: 0.9310
Iteration 14000, Loss: 0.9285
Iteration 15000, Loss: 0.9263
Iteration 16000, Loss: 0.9242
Iteration 17000, Loss: 0.9222
Iteration 18000, Loss: 0.9204
Iteration 19000, Loss: 0.9187
Iteration 20000, Loss: 0.9172
Iteration 21000, Loss: 0.9157
Iteration 22000, Loss: 0.9143
Iteration 23000, Loss: 0.9130
Iteration 24000, Loss: 0.9117
Iteration 25000, Loss: 0.9106
Iteration 26000, Loss: 0.9095
Iteration 27000, Loss: 0.9084
Iteration 28000, Loss: 0.9074
Iteration 29000, Loss: 0.9064
```

```
--- Logging Model Training Trace to MLflow ---

Evaluating final model on the test set…

2025/10/02 19:36:27 WARNING mlflow.models.model: `artifact_path` is deprecated.
Please use `name` instead.


Final Model Test Accuracy: 0.5630
=======================================================
         Custom Classification Report (from Scratch)
=======================================================
            precision   recall      f1-score    support
-------------------------------------------------------
Class 0     0.75        0.77        0.76        398
Class 1     0.45        0.41        0.43        385
Class 2     0.41        0.47        0.44        390
Class 3     0.65        0.61        0.63        390
-------------------------------------------------------
accuracy 0.56

macro avg     0.56        0.56        0.56
weighted avg  0.57        0.56        0.56
=======================================================

Logging and registering the model as 'st126488-a3-model'…

Downloading artifacts: 100%|      | 7/7 [00:00<00:00, 2501.29it/s]
Registered model 'st126488-a3-model' already exists. Creating a new version of
this model…
2025/10/02 19:36:36 WARNING mlflow.tracking._model_registry.fluent: Run with id
c8ace762d18c432fb0985e06ad0b23c3 has no artifacts at artifact path 'model',
registering model based on models:/m-478c704966ce4d52aa92a33ab934ead2 instead
2025/10/02 19:36:36 INFO mlflow.store.model_registry.abstract_store: Waiting up
to 300 seconds for model version to finish creation. Model name:
st126488-a3-model, version 12
Created version '12' of model 'st126488-a3-model'.

Waiting 5 seconds for the model registry to update…
Setting alias 'Staging' for new version 12…
 Successfully set alias 'Staging' for version 12 of model 'st126488-a3-model'.
 View run final_model_fixed_lambda_0.1 at: http://mlflow.ml.brain.cs.ait.ac.th/
#/experiments/974791038746408189/runs/c8ace762d18c432fb0985e06ad0b23c3
 View experiment at:
http://mlflow.ml.brain.cs.ait.ac.th/#/experiments/974791038746408189

 Final model pipeline saved locally to:
deploy_assets\best_classifier_pipeline.joblib
 Deployment assets saved locally to: deploy_assets\assets.json
```

```
==========================================================
         Inference Testing on Sample Data
==========================================================

--- Testing: Budget Car ---
Input Features:
 car_age  km_driven  mileage  owner  brand  km_per_year
      14     150000     18.0      3 Maruti      10000.0

Prediction:
  -> Predicted Category: Class 0

--- Testing: Mid-Range Car ---
Input Features:
 car_age  km_driven  mileage  owner brand  km_per_year
       6      60000     22.0      2 Honda  8571.428571

Prediction:
  -> Predicted Category: Class 3

--- Testing: Premium Car ---
Input Features:
 car_age  km_driven  mileage  owner brand  km_per_year
       2      20000     15.0      1   BMW  6666.666667

Prediction:
  -> Predicted Category: Class 3
```