System Requirements Specification Index

For Machine learning Algorithm No 5

1.0

Machine Learning Assessment: SVM and Multiple Linear Regression assessment

Machine Learning Assessment: Car Resale Price Prediction and Credit Risk Classification

Machine Learning Assessment Instructions

Overview

This assessment consists of two Python files with skeleton code that you need to implement:

1. `car.py` - Car resale price prediction using multiple linear regression

2. `main.py` - Credit risk classification using SVM

Dataset Information

Car Resale Dataset (`car_resale.csv`)

This dataset contains information about cars and their resale prices.

Key columns:

- `age`: Age of the car in years

- `original_price`: Original price of the car

- `mileage`: Total distance traveled by the car

- `fuel_type`: Type of fuel (Petrol, Diesel, Electric)

- `num_owners`: Number of previous owners

- `brand_score`: Score representing the brand value (1-10)

- `resale_price`: Resale price of the car (target variable)

Credit Risk Dataset (`credit_risk_dataset.csv`)

This dataset contains information about loan applicants and whether they defaulted on their loans.

Key columns:

- `person_age`: Age of the applicant

- `person_income`: Annual income of the applicant

- `person_home_ownership`: Home ownership status (RENT, OWN, MORTGAGE)

- `person_emp_length`: Employment length in years

- `loan_intent`: Purpose of the loan (PERSONAL, EDUCATION, MEDICAL, VENTURE, etc.)

- `loan_grade`: Loan grade (A, B, C, etc.)

- `loan_amnt`: Loan amount

- `loan_int_rate`: Interest rate of the loan

- `loan_status`: Whether the applicant defaulted on the loan (0/1) - This is the target variable

- `loan_percent_income`: Loan amount as a percentage of income

- `cb_person_default_on_file`: Whether the person has defaulted before (Y/N)

- `cb_person_cred_hist_length`: Credit history length in years

Task Overview

Your task is to implement the functions in both files according to the TODO comments. Each function has detailed instructions on what it should do. The skeleton code is designed to fail the tests initially, and your implementation should make the tests pass.

Detailed Implementation Instructions

Car Resale Price Prediction (`car.py`) - Multiple Linear Regression

This part of the assessment focuses on implementing a multiple linear regression model to predict car resale prices based on several features (age, original price, mileage, fuel type, number of owners, and brand score).

Function 1: `load_and_prepare_data(path="car_resale.csv")`

Purpose: Load and preprocess the car resale dataset.

Implementation Steps:

1. Use `pd.read_csv(path)` to load the dataset

2. Fill missing values in 'fuel_type' with 'Petrol' and 'brand_score' with its median

3. Use `LabelEncoder()` to encode the 'fuel_type' categorical feature

4. Print a success message: " Data loaded and preprocessed."

5. Return the processed DataFrame

Function 2: `explore_data(df)`

Purpose: Explore the dataset and print key statistics.

Implementation Steps:

1. Calculate the maximum resale price from the DataFrame

2. Calculate the average resale price from the DataFrame

3. Print these statistics with appropriate formatting:

python

```
print(f"\n Max Resale Price: ₹{max_price}")
print(f" Avg Resale Price: ₹{mean_price:.2f}")
```

Function 3: `prediction_demo(model, X_sample)`

Purpose: Demonstrate a prediction using the trained model.

Implementation Steps:

1. Use the model to predict the resale price for the given sample

2. Print the prediction with appropriate formatting:

python

print(f"\n Predicted Resale Price: ₹{int(prediction[0])}")

Function 4: `cost_function(y_true, y_pred)`

Purpose: Calculate the Mean Squared Error between true and predicted values.

Implementation Steps:

1. Implement the MSE calculation using numpy:

return np.mean((y_true - y_pred) ** 2)

Function 5: `train_and_evaluate(X_train, y_train, X_test, y_test, path="car_resale_model.pkl")`

Purpose: Train a linear regression model, save it, and evaluate its performance.

Implementation Steps:

1. Create and train a MultilinearRegression model

2. Save the model to the specified path using joblib

3. Print a success message

4. Make predictions on the test set

5. Calculate the cost using the custom cost function

6. Print the cost and sample predictions:

print(f"\n Model trained and saved to '{path}'")

print(f"\n Custom MSE: {cost:.2f}")

print(" Sample Predictions:", y_pred[:5])

Credit Risk Classification (`main.py`) - SVM Classification

This part of the assessment focuses on implementing a Support Vector Machine (SVM) classifier to predict credit risk (loan default) based on various applicant and loan features.

Function 1: `load_and_prepare_data(path="credit_risk_dataset.csv")`

Purpose: Load and prepare the credit risk dataset.

Implementation Steps:

1. Load the CSV file from the given path

2. Fill missing values using forward fill method (`df.fillna(method="ffill", inplace=True)`)

3. Encode categorical features using LabelEncoder:

python

categorical_cols = ['person_home_ownership', 'loan_intent', 'loan_grade', 'cb_person_default_on_file']

for col in categorical_cols:

le = LabelEncoder()

df[col] = le.fit_transform(df[col])

4. Print a success message: " Data loaded and preprocessed."

5. Return the processed DataFrame

Function 2: `apply_smote(X, y)`

Purpose: Apply SMOTE to balance the classes in the dataset.

Implementation Steps:

1. Create a SMOTE object with random_state=42

2. Apply fit_resample to get balanced X and y

3. Print a success message: " SMOTE applied."

4. Return the resampled X and y

Function 3: `hypothesis(model, X)`

Purpose: Calculate the decision function scores for the SVM model.

Implementation Steps:

1. Use the model's decision_function method to get raw scores

2. Return the decision scores

Function 4: `hinge_loss(y_true, scores)`

Purpose: Calculate the hinge loss for SVM predictions.

Implementation Steps:

1. Convert binary labels to signed format (-1, 1)

2. Calculate the hinge loss using the formula: max(0, 1 - y * scores)

3. Return the mean loss

Function 5: `train_and_predict(df)`

Purpose: Train an SVM model and make predictions.

Implementation Steps:

1. Split the data into features (X) and target (y)

2. Scale the features using StandardScaler

3. Apply SMOTE to balance the classes

4. Split the data into training and testing sets

5. Create and train an SVM model

6. Calculate decision scores using the hypothesis function

7. Make predictions

8. Calculate the hinge loss

9. Print sample predictions and the loss:

print(" Sample Predictions:", y_pred[:10])

print(" Hinge Loss (Custom Cost):", loss)

Testing Your Implementation

After implementing the functions according to the instructions, you can run the test cases to verify your implementation:

Python3 -m unittest

Submission Guidelines

1. Complete all the required functions in `Main.py` and `car.py`

2. Ensure all tests pass

3. Submit your code files

Execution Steps to Follow:

- All actions like build, compile, running application, running test cases will be through Command Terminal.
- To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal
- This editor Auto Saves the code
- If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use **CTRL+Shift+B** -command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.
- These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.
- To setup environment:
  You can run the application without importing any packages
- To launch application:
  Python3 main .py

  Python3 car.py
- To run Test cases:
  python3 -m unittest

- Before Final Submission also, you need to use **CTRL+Shift+B** - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.

Screen shot to run the program

```
OK
coder@dighe20250227t070305rz1fj5p3:/home/myproject/dighegmailcom_20250227T070305$ python3 <<scriptname>>.py
```

To run the application

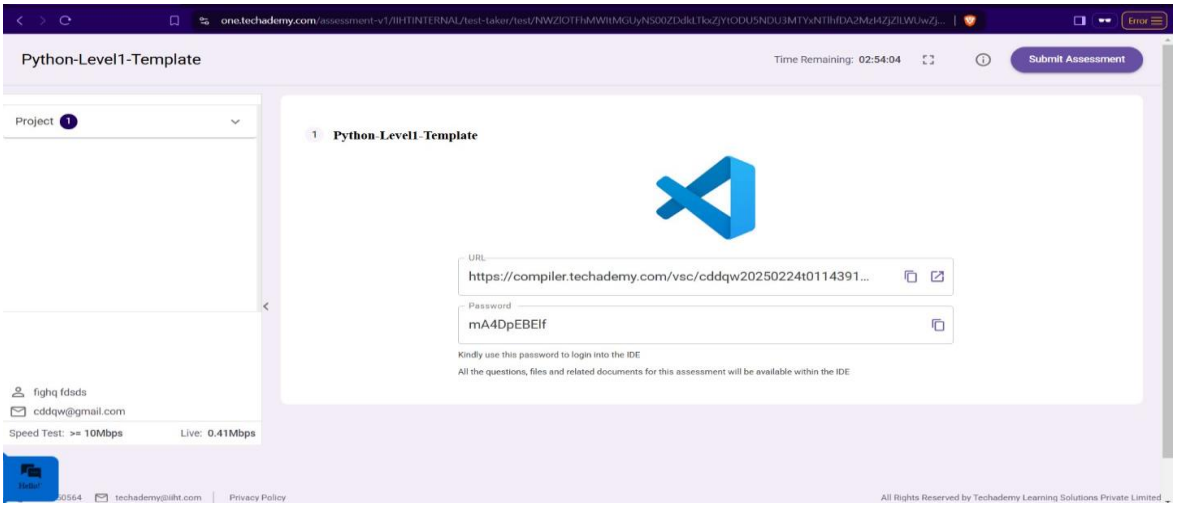
Python3 main .py

Python3 car.py

```
coder@dighe20250227t070305rz1fj5p3:/home/myproject/dighegmailcom_20250227T070305$ python3 -m unittest
TestBoundary = Passed
.TestExceptional = Passed
.TestCalculateTotalDonations = Failed
.TestCalculateTotalStockValue = Failed
.TestCheckFrankWhiteDonated = Failed
```

To run the testcase python3 -

m unittest



- **Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on “Submit Assessment” after you are done with code.**