The Superior University, Lahore

Assignment-I (Fall 2023)

Course Title:	Programming for AI				Course Code:	CAI601410	Credit Hours: 4
Instructor:	Prof. Rasikh Ali				Programme Name:	BSDS	
Semester:	4th	Batch:	F23	Section:	BSDSM-4A	Date:	1st February, 2025
Time Allowed:					Maximum Marks:		
Student's Name:	Asaad Iqbal				Reg. No.	SU92-BSDSM-F23-020	

Lab-Task 1

1: House Price Prediction

Task 1

House Price Prediction Model Documentation

Introduction

This document provides a step-by-step explanation of the house price prediction model using machine learning techniques in Python. The model uses a **Random Forest Regressor** to predict house prices based on various features from the dataset, while avoiding the use of a validation split by utilizing **cross**-validation.

Step 1: Importing Libraries

[79]:
```
import pandas as pd
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.preprocessing import LabelEncoder
```

- **pandas**: Used for data manipulation and handling CSV files.
- **RandomForestRegressor**: A machine learning model used for regression tasks.
- **mean_absolute_error**: Measures the model's performance.
- **LabelEncoder**: Encodes categorical features into numeric values.
- **cross_val_score**: Performs cross-validation to assess the model's performance.

Step 2: Loading dataset

[48]:
```
df = pd.read_csv('sample_submission.csv')
df1 = pd.read_csv('train.csv')
df2 = pd.read_csv('test.csv')
print("All Data sets are loaded")
```

```
All Data sets are loaded
```

- Reads the datasets (train.csv, test.csv, and sample_submission.csv) into Pandas DataFrames.
- df.head(), df1.head() and df2.head() provide first 5 rows of each dataset.
- df.info(), df1.info() and df2.info() provide details about the data structure, including column names, data types, and missing values.

Step 3: Handle missing values

[61]:
```
def fill_missing_values(data):
    for col in data.columns:
        if data[col].dtype == 'O':  # Categorical
            data[col] = data[col].fillna(data[col].mode()[0])
        elif data[col].dtype == 'float64':  # Float
            data[col] = data[col].fillna(data[col].mean())
        elif data[col].dtype == 'int64':  # Integer
            data[col] = data[col].fillna(data[col].median())

fill_missing_values(df1)
fill_missing_values(df2)
```

- Defines the fill_missing_values() function to handle missing values in the dataset:
    a. **Categorical ('O')**: Filled with the most frequent value (mode).
    b. **Float ('float64')**: Filled with the column mean.
    c. **Integer ('int64')**: Filled with the column median.
- Applies this function to both df1 (training data) and df2 (test data) to ensure no missing values remain.

Step 4: Selecting features and target variables

[73]:
```
cols = ['OverallQual', 'GrLivArea', 'YearBuilt', 'TotalBsmtSF', 'GarageArea', 'ExterCond', 'SalePrice']
df_selected = df1[cols].copy()
```

[74]:
```
df_selected = df_selected.dropna(subset=['SalePrice'])
```

- Selects important columns that are related to house pricing.
- Drops rows with missing SalePrice values since they are the target variable.

Step 5: Encoding target variables

```
[75]: label_encoder = LabelEncoder()
      df_selected['ExterCond'] = label_encoder.fit_transform(df_selected['ExterCond'])
```

- ExterCond (Exterior Condition) is a categorical feature that needs to be converted into numerical format.
- LabelEncoder is used to convert the categorical feature into numeric values.

Step 6: Training the model on all data

```
[76]: X = df_selected.drop(columns='SalePrice')
      y = df_selected['SalePrice']
```

```
[77]: model = RandomForestRegressor(n_estimators=100, random_state=42)
      model.fit(X, y)
```

```
[77]: RandomForestRegressor
      RandomForestRegressor(random_state=42)
```

- Separates the features (X) and the target variable (y).
- **RandomForestRegressor** is used with 100 trees (estimators).
- The model is trained using the entire dataset without splitting it, as no validation set is used in this workflow.

Step 7: Validating the model

```
[80]: cv_scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_absolute_error')
      mean_cv_score = -cv_scores.mean()
      print(f"Cross-validated Mean Absolute Error: {mean_cv_score}")

      Cross-validated Mean Absolute Error: 20579.668644121248
```

Step 8: Preparing Test Data for prediction

```
[81]: feature_cols = ['OverallQual', 'GrLivArea', 'YearBuilt', 'TotalBsmtSF', 'GarageArea', 'ExterCond']
      df2_processed = df2[feature_cols].copy()
      df2_processed['ExterCond'] = label_encoder.transform(df2_processed['ExterCond'])
```

- Selects the same features used for training from the test dataset (df2).
- Applies the **LabelEncoder** to transform categorical values in the ExterCond column in the test data.

Step 9: Making predictions

```
[82]: y_pred_full = model.predict(df2_processed)
      assert len(y_pred_full) == len(df2), f"Expected {len(df2)} predictions, but found {len(y_pred_full)}"
```

- The trained model makes predictions on the processed test data (df2_processed).

- An assertion is used to ensure that the number of predictions matches the number of test data points.

Step 10: Saving Predictions to csv

```
[83]: predictions_df = pd.DataFrame({
          'Id': df2['Id'].values,
          'SalePrice': y_pred_full
      })
      predictions_df.to_csv('house_price_predictions.csv', index=False)
      print("Submission file created successfully!")
```

```
Submission file created successfully!
```

- Creates a new DataFrame containing the Id from df2 (test dataset) and the predicted house prices (y_pred_full).
- Saves the predictions as a CSV file (house_price_predictions.csv).
- Prints a success message confirming the submission file is created.

Kaggle Competition Result