MACHINE LEARNING NOTEBOOK: PREDICTING CALIFORNIA HOUSE PRICES

This notebook provides a step-by-step guide to building a machine learning model for a regression task. We will use the California Housing dataset to predict the median house value in a given area.
The process will cover:

1. **Data Loading and Exploration**: Understanding the dataset's structure and contents.
2. **Data Preprocessing**: Preparing the data for the model, including splitting and scaling.
3. **Feature Selection**: Identifying the most relevant features to improve model performance and reduce complexity.
4. **Model Training**: Training different regression models on the preprocessed data.
5. **Model Evaluation**: Assessing model performance using key regression metrics.

1. Data Loading and Exploration

First, we'll load the necessary libraries and the dataset. We'll use pandas for data manipulation, sklearn for machine learning tools, and matplotlib/seaborn for visualization.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score

# Set plot style
sns.set_style("whitegrid")

# Load the dataset
housing = fetch_california_housing(as_frame=True)
df = housing.frame
df['target'] = housing.target

# Display the first 5 rows of the dataset
print("### Dataset Head")
print(df.head())

# Display dataset information and summary statistics
print("\n### Dataset Info")
```

```
df.info()

print("\n### Descriptive Statistics")
print(df.describe())

# Visualize the distribution of the target variable
plt.figure(figsize=(10, 6))
sns.histplot(df['target'], kde=True, bins=50)
plt.title('Distribution of Median House Value')
plt.xlabel('Median House Value')
plt.ylabel('Frequency')
plt.show()

# Visualize feature correlations
plt.figure(figsize=(12, 10))
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f")
plt.title('Feature Correlation Matrix')
plt.show()
```

2. Data Preprocessing

Before training a model, it's crucial to prepare the data. This involves separating features (X) from the target (y), splitting the data into training and testing sets, and scaling the features. Scaling is important for many algorithms (like Linear Regression) that are sensitive to the magnitude of input variables.

```
# Separate features and target
X = df.drop('target', axis=1)
y = df['target']

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

print(f"Shape of X_train: {X_train.shape}")
print(f"Shape of X_test: {X_test.shape}")
print(f"Shape of y_train: {y_train.shape}")
print(f"Shape of y_test: {y_test.shape}")

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit the scaler on the training data and transform both training and
test data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
print("\nData has been split and scaled successfully.")
```

3. Feature Selection

Feature selection helps in reducing the dimensionality of the dataset, which can prevent overfitting, improve model performance, and decrease training time. We'll use SelectKBest with the f_regression scoring function to select the top 5 most important features.

```
# Initialize SelectKBest to select the top 5 features
selector = SelectKBest(score_func=f_regression, k=5)

# Fit the selector on the scaled training data
selector.fit(X_train_scaled, y_train)

# Get a boolean mask of the selected features
selected_features_mask = selector.get_support()

# Get the names of the selected features
selected_features = X.columns[selected_features_mask]

print("Top 5 selected features:")
print(selected_features)

# Transform the training and test data to include only the selected
features
X_train_selected = selector.transform(X_train_scaled)
X_test_selected = selector.transform(X_test_scaled)

print(f"\nShape of X_train_selected: {X_train_selected.shape}")
print(f"Shape of X_test_selected: {X_test_selected.shape}")
```

4. Model Training

We will train two different regression models: a simple LinearRegression model and a more complex RandomForestRegressor. This allows us to compare the performance of a basic linear model against a powerful ensemble model. We will train each model on two datasets: the full scaled dataset and the feature-selected dataset.

```
# Initialize the models
lr_model = LinearRegression()
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# --- Training on the FULL dataset ---
print("### Training Models on Full Dataset")
lr_model.fit(X_train_scaled, y_train)
rf_model.fit(X_train_scaled, y_train)
```

```
# --- Training on the FEATURE-SELECTED dataset ---
print("\n### Training Models on Feature-Selected Dataset")
lr_model_sel = LinearRegression()
rf_model_sel = RandomForestRegressor(n_estimators=100,
random_state=42)

lr_model_sel.fit(X_train_selected, y_train)
rf_model_sel.fit(X_train_selected, y_train)

print("\nModels trained successfully.")
```

5. Model Evaluation

Evaluation is a critical step to determine how well our models perform on unseen data. We'll use the following metrics:

- **Mean Absolute Error (MAE)**: The average of the absolute differences between predictions and actual values. It's in the same unit as the target variable.
- **Mean Squared Error (MSE)**: The average of the squared differences. It penalizes larger errors more heavily.
- **R-squared (R^2)**: A value between 0 and 1 that represents the proportion of the variance in the target variable that is predictable from the features. A higher value indicates a better fit.

<!-- end list -->

```
def evaluate_model(model, X_test, y_test, model_name, dataset_type):
    """
    Predicts on the test set and prints evaluation metrics.
    """
    y_pred = model.predict(X_test)

    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, y_pred)

    print(f"--- {model_name} on {dataset_type} Dataset ---")
    print(f"  Mean Absolute Error (MAE): {mae:.4f}")
    print(f"  Mean Squared Error (MSE): {mse:.4f}")
    print(f"  Root Mean Squared Error (RMSE): {rmse:.4f}")
    print(f"  R-squared (R2): {r2:.4f}")
    print("-" * 40)

# Evaluate Linear Regression model
evaluate_model(lr_model, X_test_scaled, y_test, "Linear Regression",
"Full")
evaluate_model(lr_model_sel, X_test_selected, y_test, "Linear
Regression", "Feature-Selected")
```

```
# Evaluate Random Forest Regressor model
evaluate_model(rf_model, X_test_scaled, y_test, "Random Forest",
"Full")
evaluate_model(rf_model_sel, X_test_selected, y_test, "Random Forest",
"Feature-Selected")
```

Analysis of Results

Based on the evaluation metrics, we can draw the following conclusions:

- **Random Forest Regressor** consistently outperforms the Linear Regression model, regardless of whether feature selection was applied. This is expected, as Random Forest is a more powerful, non-linear model that can capture more complex relationships in the data.
- **Linear Regression** performance improved slightly after feature selection (e.g., lower MSE and higher R^2). This indicates that removing less important features helped the model find a better linear relationship.
- **Random Forest** performance slightly degraded after feature selection. For complex models like Random Forest, more features can sometimes provide richer information, and the model is robust enough to handle the noise from less important features. This suggests that for this specific dataset and model, including all features was beneficial.

In conclusion, for this problem, the RandomForestRegressor model trained on the full dataset provides the most accurate predictions. This notebook demonstrates the end-to-end process of building and evaluating a machine learning model, from data preparation to final performance assessment.