Student Information

Name: Ramesh Chandra Soren
Enrollment No: 2022CSB086
Department: Computer Science and Technology

Overfitting and Underfitting Analysis using Regression/Classification Models

1. Dataset Selection and Preparation

1.1. Datasets

Download the following datasets:

- Mobile Price Classification Dataset
- Housing Price Dataset
- Melbourne Housing Snapshot Dataset

1.2. Feature Analysis

Analyze the features of each dataset and select the relevant attributes for prediction or classification tasks.

2. Data Preprocessing

2.1. Handling Missing Values

Use appropriate imputation techniques to handle missing values in the datasets.

2.2. Normalization

Normalize the datasets if necessary to ensure that features are on a similar scale.

3. Model Development

3.1. Data Splitting

Split each dataset into training and testing sets. Consider using stratified k-fold cross-validation to ensure a balanced split.

3.2. Model Development

Develop regression models (e.g., linear regression, multiple regression) or classification models to predict the target variable.

3.3. Parameter Estimation and Predictions

For each model:

- Estimate the parameters.
- Generate predictions on both training and testing sets.

4. Overfitting/Underfitting Analysis

4.1. Loss Curves

Plot the training and validation loss curves to visualize and identify overfitting or underfitting scenarios.

4.2. Model Evaluation

Evaluate the models using metrics such as:

- Mean Squared Error (MSE)
- R^2 Score

Assess the performance on both training and testing sets.

4.3. Performance Comparison

Compare the performance of different models and discuss observations related to overfitting and underfitting.

5. Reporting

5.1. Summary of Results

Summarize the results and insights from your analysis.

5.2. Observations

Highlight any patterns or trends observed during the study.

5.3. Recommendations

Provide recommendations for improving model performance and addressing overfitting or underfitting issues.

By conducting this comprehensive analysis, you will gain a deeper understanding of how different regression models perform on various datasets and the common pitfalls associated with overfitting and underfitting in machine learning.

##Following is the code of Housing data set

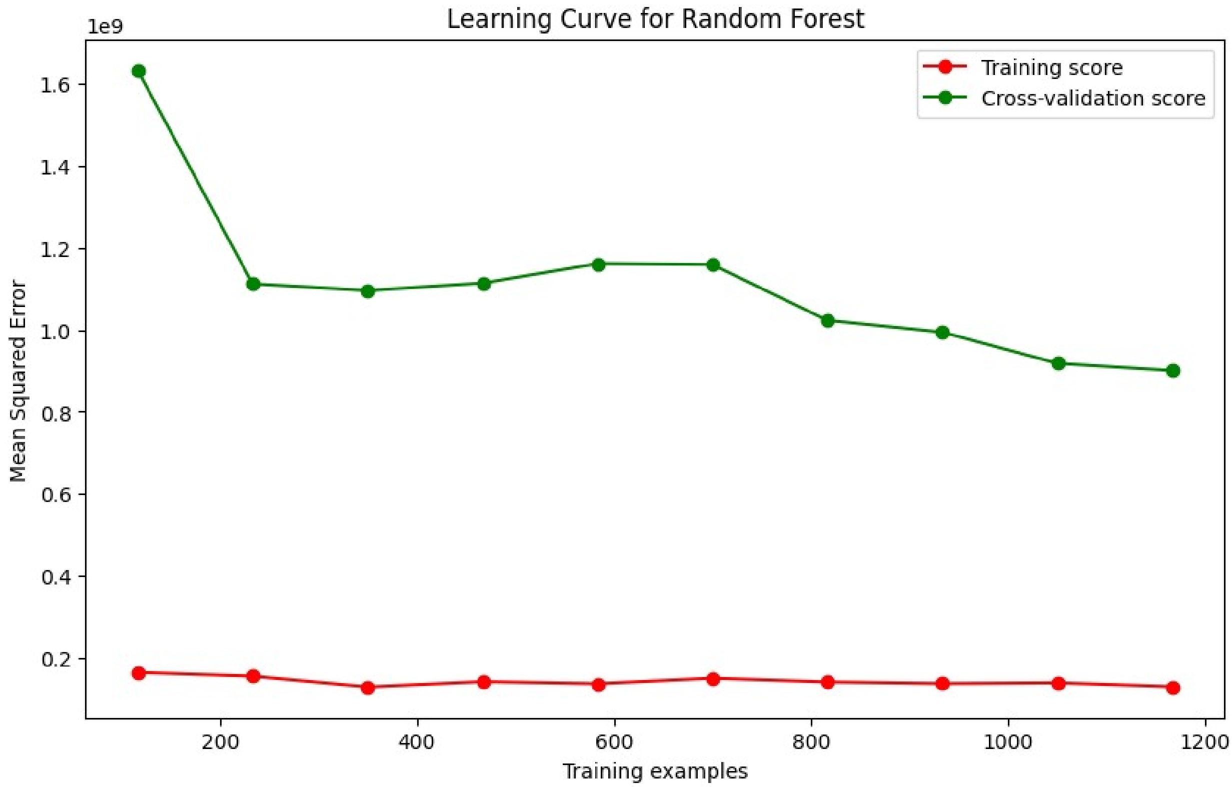
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Load the data
data = pd.read_csv('train.csv')

# Assume 'SalePrice' is the target variable
X = data.drop('SalePrice', axis=1)
y = data['SalePrice']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Identify numeric and categorical columns
numeric_features = X.select_dtypes(include=['int64',
'float64']).columns
categorical_features = X.select_dtypes(include=['object']).columns

# Create preprocessing pipelines
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant',
fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Combine preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
```

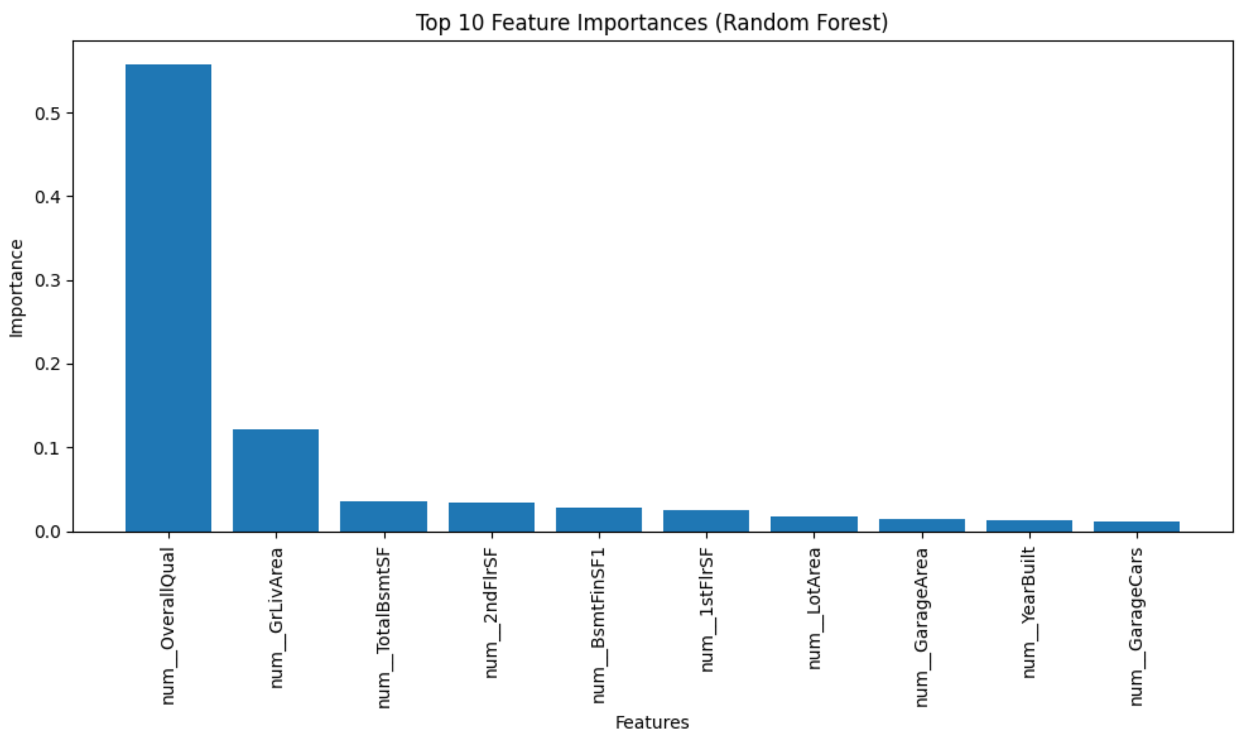
```python
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Create model pipelines
models = {
    'Linear Regression': Pipeline([
        ('preprocessor', preprocessor),
        ('regressor', LinearRegression())
    ]),
    'Decision Tree': Pipeline([
        ('preprocessor', preprocessor),
        ('regressor', DecisionTreeRegressor(random_state=42))
    ]),
    'Random Forest': Pipeline([
        ('preprocessor', preprocessor),
        ('regressor', RandomForestRegressor(random_state=42))
    ])
}

# Train and evaluate models
results = {}

for name, model in models.items():
    model.fit(X_train, y_train)

    train_pred = model.predict(X_train)
    test_pred = model.predict(X_test)

    train_mse = mean_squared_error(y_train, train_pred)
    test_mse = mean_squared_error(y_test, test_pred)
    train_r2 = r2_score(y_train, train_pred)
    test_r2 = r2_score(y_test, test_pred)

    results[name] = {
        'train_mse': train_mse,
        'test_mse': test_mse,
        'train_r2': train_r2,
        'test_r2': test_r2
    }

    print(f"\n{name}:")
    print(f"Train MSE: {train_mse:.2f}, Test MSE: {test_mse:.2f}")
    print(f"Train R2: {train_r2:.2f}, Test R2: {test_r2:.2f}")

# Plotting learning curves for Random Forest
def plot_learning_curve(estimator, X, y, title):
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=5, n_jobs=-1,
        train_sizes=np.linspace(0.1, 1.0, 10),
```

```
        scoring='neg_mean_squared_error')

    train_scores_mean = -train_scores.mean(axis=1)
    test_scores_mean = -test_scores.mean(axis=1)

    plt.figure(figsize=(10, 6))
    plt.title(title)
    plt.xlabel("Training examples")
    plt.ylabel("Mean Squared Error")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
label="Cross-validation score")
    plt.legend(loc="best")
    plt.show()

from sklearn.model_selection import learning_curve

plot_learning_curve(models['Random Forest'], X, y, "Learning Curve for
Random Forest")

# Feature importance for Random Forest
feature_importance = models['Random
Forest'].named_steps['regressor'].feature_importances_
feature_names = models['Random
Forest'].named_steps['preprocessor'].get_feature_names_out()

feature_importance_df = pd.DataFrame({'feature': feature_names,
'importance': feature_importance})
feature_importance_df =
feature_importance_df.sort_values('importance',
ascending=False).head(10)

plt.figure(figsize=(10, 6))
plt.bar(feature_importance_df['feature'],
feature_importance_df['importance'])
plt.title('Top 10 Feature Importances (Random Forest)')
plt.xlabel('Features')
plt.ylabel('Importance')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```

```
Linear Regression:
Train MSE: 357356635.50, Test MSE: 4269660731.07
Train R2: 0.94, Test R2: 0.44

Decision Tree:
Train MSE: 0.00, Test MSE: 1741699157.42
```

```
Train R2: 1.00, Test R2: 0.77

Random Forest:
Train MSE: 123863904.63, Test MSE: 870348388.49
Train R2: 0.98, Test R2: 0.89
```





Top 10 Feature Importances (Random Forest)
Importance
0.0
0.1
0.2
0.3
0.4
0.5
num__OverallQual
num__GrLivArea
num__TotalBsmtSF
num__2ndFlrSF
num__BsmtFinSF1
num__1stFlrSF
num__LotArea
num__GarageArea
num__YearBuilt
num__GarageCars
Features