jupyter regression_assignment Last Checkpoint: 8 seconds ago

File Edit View Run Kernel Settings Help

Trusted

Markdown

JupyterLab Python 3 (ipykernel)

Loading and preprocessing

[3]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

#loading California Housing dataset
california_housing = fetch_california_housing(as_frame=True)
```

[5]:
```
#converting to pandas DataFrame
df = california_housing.frame

df.head()
```

[5]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

[7]:
```
#checking missing/null values
df.isnull().sum()
```

[7]:
```
MedInc        0
HouseAge      0
AveRooms      0
AveBedrms     0
Population    0
AveOccup      0
```

[7]:
```
#checking missing/null values
df.isnull().sum()
```

[7]:
```
MedInc        0
HouseAge      0
AveRooms      0
AveBedrms     0
Population    0
AveOccup      0
Latitude      0
Longitude     0
MedHouseVal   0
dtype: int64
```

[9]:
```
#feature scaling using standardization
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df.drop(columns=['MedHouseVal']))
```

[11]:
```
#creating new DataFrame with scaled features
df_scaled = pd.DataFrame(scaled_features, columns=df.columns[:-1])

#adding 'MedHouseVal' to the scaled dataframe
df_scaled['MedHouseVal'] = df['MedHouseVal']

#displaying first few rows of the preprocessed data
df_scaled.head()
```

[11]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	2.344766	0.982143	0.628559	-0.153758	-0.974429	-0.049597	1.052548	-1.327835	4.526
1	2.332238	-0.607019	0.327041	-0.263336	0.861439	-0.092512	1.043185	-1.322844	3.585
2	1.782699	1.856182	1.155620	-0.049016	-0.820777	-0.025843	1.038503	-1.332827	3.521
3	0.932968	1.856182	0.156966	-0.049833	-0.766028	-0.050329	1.038503	-1.337818	3.413
4	-0.012881	1.856182	0.344711	-0.032906	-0.759847	-0.085616	1.038503	-1.337818	3.422

Explanation:

- Loading into DataFrame : This makes data manipulation easier and allows for better visualization
- Missing Value Handling : Checked missing values to avoid errors during model training, This dataset has no missing values
- Feature Scaling : Standardizing ensures that each feature contributes equally to the model training process

jupyter regression_assignment Last Checkpoint: 32 seconds ago

File Edit View Run Kernel Settings Help

Trusted

Markdown

JupyterLab Python 3 (ipykernel)

Explanation:

- Loading into DataFrame : This makes data manipulation easier and allows for better visualization
- Missing Value Handling : Checked missing values to avoid errors during model training, This dataset has no missing values
- Feature Scaling : Standardizing ensures that each feature contributes equally to the model training process

Regression Algorithm implementation

[15]:
```
#splitting data into training and testing sets
X = df_scaled.drop(columns=['MedHouseVal'])
y = df_scaled['MedHouseVal']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

[17]:
```
#initializing models
models = {
    'Linear Regression' : LinearRegression(),
    'Decision Tree Regressor' : DecisionTreeRegressor(),
    'Random Forest Regressor' : RandomForestRegressor(),
    'Gradient Boosting regressor' : GradientBoostingRegressor(),
    'Support Vector Regressor' : SVR()
}
```

Explanation

1. Linear Regression

Explanation : Linear regression models the relationship between dependent and independent variables by fitting a linear equation to the observed data.

Suitability : It is suitable due to its simplicity and interpretability, especially when relationships are approximately linear.

2. Decision Tree Regressor

Explanation : This algorithm uses a tree-like model of decisions based on feature values, splitting data into subsets based on feature thresholds.

Suitability : It can capture non-linear relationships and interactions between features effectively.

3. Random Forest Regressor

Explanation : An ensemble method that builds multiple decision trees and merges them to improve accuracy and control overfitting.

jupyter regression_assignment Last Checkpoint: 48 seconds ago

File Edit View Run Kernel Settings Help

Trusted

Markdown

JupyterLab Python 3 (ipykernel)

Explanation

1. Linear Regression

Explanation : Linear regression models the relationship between dependent and independent variables by fitting a linear equation to the observed data.

Suitability : It is suitable due to its simplicity and interpretability, especially when relationships are approximately linear.

2. Decision Tree Regressor

Explanation : This algorithm uses a tree-like model of decisions based on feature values, splitting data into subsets based on feature thresholds.

Suitability : It can capture non-linear relationships and interactions between features effectively.

3. Random Forest Regressor

Explanation : An ensemble method that builds multiple decision trees and merges them to improve accuracy and control overfitting.

Suitability : It handles high-dimensional spaces well and is robust against overfitting, making it ideal for complex datasets.

4. Gradient Boosting Regressor

Explanation : This method builds trees sequentially, wherre each new tree corrects errors made by previously trained trees.

Suitability : It performs well on various datasets, especially when fine-tuned with hyperparameters.

5. Support Vector Regressor (SVR)

Explanation : SVR attempts to fit as many instances as possible within a margin of tolerance while minimizing the model complexity.

Suitability : It is effective in high-dimensional spaces and when dealing with non-linear relationships.

Model Evaluation and Comparison

[21]:
```
#train and evaluate models
results = {}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
```

jupyter regression_assignment Last Checkpoint: 1 minute ago

File Edit View Run Kernel Settings Help Trusted

Markdown JupyterLab Python 3 (ipykernel)

Model Evaluation and Comparison

[21]:
```
#train and evaluate models
results = {}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    results[name] = {'MSE' : mse, 'MAE' : mae, 'R²' : r2}

#display results
results_df = pd.DataFrame(results).T

results_df
```

[21]:

	MSE	MAE	R^2
Linear Regression	0.555892	0.533200	0.575788
Decision Tree Regressor	0.500692	0.457087	0.617912
Random Forest Regressor	0.254640	0.327505	0.805679
Gradient Boosting regressor	0.294018	0.371709	0.775629
Support Vector Regressor	0.355198	0.397763	0.728941

[22]:
```
#identifying the best and worst performing models
best_model = results_df['R²'].idxmax()
worst_model = results_df['R²'].idxmin()

print(f'Best Permorming Model : {best_model}\n', results_df.loc[best_model])
print('-----------------------------------------------')
print(f'Worst Performing Model : {worst_model}\n', results_df.loc[worst_model])
```

```
Best Permorming Model : Random Forest Regressor
 MSE    0.254640
MAE    0.327505
R²     0.805679
Name: Random Forest Regressor, dtype: float64
-----------------------------------------------
Worst Performing Model : Linear Regression
```

jupyter regression_assignment Last Checkpoint: 1 minute ago

File Edit View Run Kernel Settings Help

Trusted

Markdown

JupyterLab Python 3 (ipykernel)

Support Vector Regressor	0.355198	0.397763	0.728941

[22]:

```
#identifying the best and worst performing models
best_model = results_df['R²'].idxmax()
worst_model = results_df['R²'].idxmin()

print(f'Best Permorming Model : {best_model}\n', results_df.loc[best_model])
print('-----------------------------------------------')
print(f'Worst Performing Model : {worst_model}\n', results_df.loc[worst_model])
```

```
Best Permorming Model : Random Forest Regressor
 MSE    0.254640
MAE    0.327505
R²     0.805679
Name: Random Forest Regressor, dtype: float64
-----------------------------------------------
Worst Performing Model : Linear Regression
 MSE    0.555892
MAE    0.533200
R²     0.575788
Name: Linear Regression, dtype: float64
```

Best Performing Algorithm:

* Random Forest Regressor:

justification ==> It has the lowest MSE (0.258953), lowest MAE (0.330066), and highest R^2 score (0.802388). This indicates it explains a significant proportion of variance in the target variable and performs well in terms of prediction accuracy.

Worst Performing Algorithm:

* Linear Regression:

Reasoning ==> It has the highest MSE (0.555892) and MAE (0.533200), with the lowest R^2 score (0.575788). This suggests it struggles to capture the underlying patterns in the data compared to other models.