learn-co-curriculum / dsc-tuning-regression-trees-lab Public

View license

1 star 148 forks

Star Watch

Code Issues Pull requests Actions Projects Security Insights

solution

This branch is 11 commits ahead, 11 commits behind master.

hoffm386 make max_depth an int … on Aug 22 15

View code

README.md

Regression Trees and Model Optimization - Lab

Introduction

In this lab, we'll see how to apply regression analysis using CART trees while making use of some hyperparameter tuning to improve our model.

Objectives

In this lab you will:

- Perform the full process of cleaning data, tuning hyperparameters, creating visualizations, and evaluating decision tree models
- Determine the optimal hyperparameters for a decision tree model and evaluate the performance of decision tree models

Ames Housing dataset

The dataset is available in the file `'ames.csv'`.

- Import the dataset and examine its dimensions:

```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('ggplot')
%matplotlib inline

# Load the Ames housing dataset
data = pd.read_csv('ames.csv')

# Print the dimensions of data
print(data.shape)

# Check out the info for the dataframe
print(data.info())

# Show the first 5 rows
data.head()
```

```
(1460, 81)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
 13  Condition1     1460 non-null   object
 14  Condition2     1460 non-null   object
```

```
 15  BldgType       1460 non-null   object
 16  HouseStyle     1460 non-null   object
 17  OverallQual    1460 non-null   int64
 18  OverallCond    1460 non-null   int64
 19  YearBuilt      1460 non-null   int64
 20  YearRemodAdd   1460 non-null   int64
 21  RoofStyle      1460 non-null   object
 22  RoofMatl       1460 non-null   object
 23  Exterior1st    1460 non-null   object
 24  Exterior2nd    1460 non-null   object
 25  MasVnrType     1452 non-null   object
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object
 28  ExterCond      1460 non-null   object
 29  Foundation     1460 non-null   object
 30  BsmtQual       1423 non-null   object
 31  BsmtCond       1423 non-null   object
 32  BsmtExposure   1422 non-null   object
 33  BsmtFinType1   1423 non-null   object
 34  BsmtFinSF1     1460 non-null   int64
 35  BsmtFinType2   1422 non-null   object
 36  BsmtFinSF2     1460 non-null   int64
 37  BsmtUnfSF      1460 non-null   int64
 38  TotalBsmtSF    1460 non-null   int64
 39  Heating        1460 non-null   object
 40  HeatingQC      1460 non-null   object
 41  CentralAir     1460 non-null   object
 42  Electrical     1459 non-null   object
 43  1stFlrSF       1460 non-null   int64
 44  2ndFlrSF       1460 non-null   int64
 45  LowQualFinSF   1460 non-null   int64
 46  GrLivArea      1460 non-null   int64
 47  BsmtFullBath   1460 non-null   int64
 48  BsmtHalfBath   1460 non-null   int64
 49  FullBath       1460 non-null   int64
 50  HalfBath       1460 non-null   int64
 51  BedroomAbvGr   1460 non-null   int64
 52  KitchenAbvGr   1460 non-null   int64
 53  KitchenQual    1460 non-null   object
 54  TotRmsAbvGrd   1460 non-null   int64
 55  Functional     1460 non-null   object
 56  Fireplaces     1460 non-null   int64
 57  FireplaceQu    770 non-null    object
 58  GarageType     1379 non-null   object
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object
 61  GarageCars     1460 non-null   int64
 62  GarageArea     1460 non-null   int64
 63  GarageQual     1379 non-null   object
```

```
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
None
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley
0	1	60	RL	65.0	8450	Pave	NaN
1	2	20	RL	80.0	9600	Pave	NaN
2	3	60	RL	68.0	11250	Pave	NaN
3	4	70	RL	60.0	9550	Pave	NaN
4	5	60	RL	84.0	14260	Pave	NaN

5 rows × 81 columns

Identify features and target data

In this lab, we will use using 3 predictive continuous features:

Features

- `LotArea` : Lot size in square feet
- `1stFlrSF` : Size of first floor in square feet
- `GrLivArea` : Above grade (ground) living area square feet

Target

- `SalePrice` ', the sale price of the home, in dollars

- Create DataFrames for the features and the target variable as shown above

- Inspect the contents of both the features and the target variable

```
# Features and target data
target = data['SalePrice']
features = data[['LotArea', '1stFlrSF', 'GrLivArea']]
print(target.describe())
print("")
features.describe()
```

```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
```

```
    text-align: right;
}
```

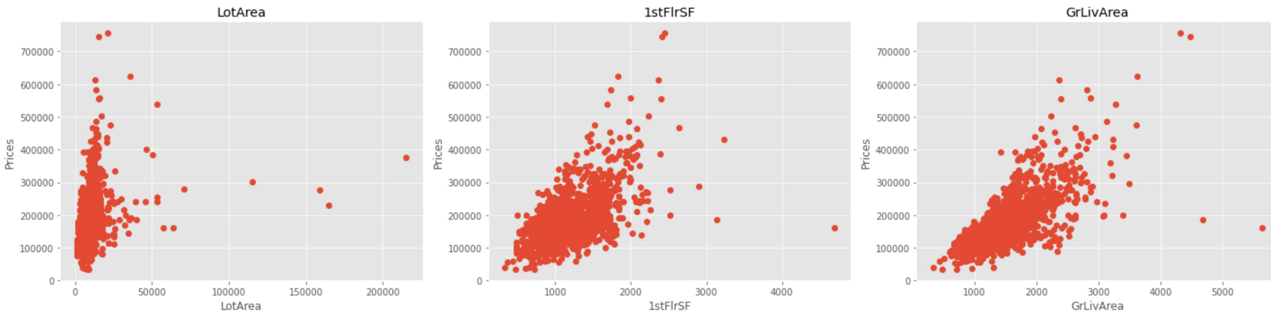
</style>

	LotArea	1stFlrSF	GrLivArea
count	1460.000000	1460.000000	1460.000000
mean	10516.828082	1162.626712	1515.463699
std	9981.264932	386.587738	525.480383
min	1300.000000	334.000000	334.000000
25%	7553.500000	882.000000	1129.500000
50%	9478.500000	1087.000000	1464.000000
75%	11601.500000	1391.250000	1776.750000
max	215245.000000	4692.000000	5642.000000

Inspect correlations

- Use scatter plots to show the correlation between the chosen features and the target variable
- Comment on each scatter plot

```
# Create scatter plots for each feature vs. target
import matplotlib.pyplot as plt
plt.figure(figsize=(20, 5))
for i, col in enumerate(features.columns):
    plt.subplot(1, 3, i+1)
    plt.plot(data[col], target, 'o')
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel('Prices')
    plt.tight_layout()
```



Create evaluation metrics

- Import `r2_score` and `mean_squared_error` from `sklearn.metrics`
- Create a function `performance(true, predicted)` to calculate and return both the R-squared score and Root Mean Squared Error (RMSE) for two equal-sized arrays for the given true and predicted values
  - Depending on your version of sklearn, in order to get the RMSE score you will need to either set `squared=False` or you will need to take the square root of the output of the `mean_squared_error` function - check out the documentation or this helpful and related StackOverflow post
  - The benefit of calculating RMSE instead of the Mean Squared Error (MSE) is that RMSE is in the same units at the target - here, this means that RMSE will be in dollars, calculating how far off in dollars our predictions are away from the actual prices for homes, on average

```
# Import metrics
from sklearn.metrics import r2_score, mean_squared_error

# Define the function
def performance(y_true, y_predict):
    """
    Calculates and returns the two performance scores between
    true and predicted values - first R-Squared, then RMSE
    """

    # Calculate the r2 score between 'y_true' and 'y_predict'
    r2 = r2_score(y_true, y_predict)

    # Calculate the root mean squared error between 'y_true' and 'y_predict'
    rmse = mean_squared_error(y_true, y_predict, squared=False)

    # If using an older version of sklearn:
    # rmse = np.sqrt(mean_squared_error(y_true, y_predict))

    # Return the score
```

```
    return [r2, rmse]

# Test the function
score = performance([3, -0.5, 2, 7, 4.2], [2.5, 0.0, 2.1, 7.8, 5.3])
score
# [0.9228556485355649, 0.6870225614927066]
```

```
[0.9228556485355649, 0.6870225614927066]
```

Split the data into training and test sets

- Split `features` and `target` datasets into training/test data (80/20)
- For reproducibility, use `random_state=42`

```
from sklearn.model_selection import train_test_split

# Split the data into training and test subsets
x_train, x_test, y_train, y_test = train_test_split(
    features, target, test_size=0.2, random_state=42)
```

Grow a vanilla regression tree

- Import the `DecisionTreeRegressor` class
- Run a baseline model for later comparison using the datasets created above
- Generate predictions for test dataset and calculate the performance measures using the function created above
- Use `random_state=45` for tree instance
- Record your observations

```
# Import DecisionTreeRegressor
from sklearn.tree import DecisionTreeRegressor

# Instantiate DecisionTreeRegressor
# Set random_state=45
regressor = DecisionTreeRegressor(random_state=45)

# Fit the model to training data
regressor.fit(x_train, y_train)
```

```
# Make predictions on the test data
y_pred = regressor.predict(x_test)

# Calculate performance using the performance() function
score = performance(y_test, y_pred)
score

# [0.5961521990414137, 55656.48543887347] - R2, RMSE
```

```
[0.5961521990414137, 55656.48543887347]
```

Hyperparameter tuning (I)

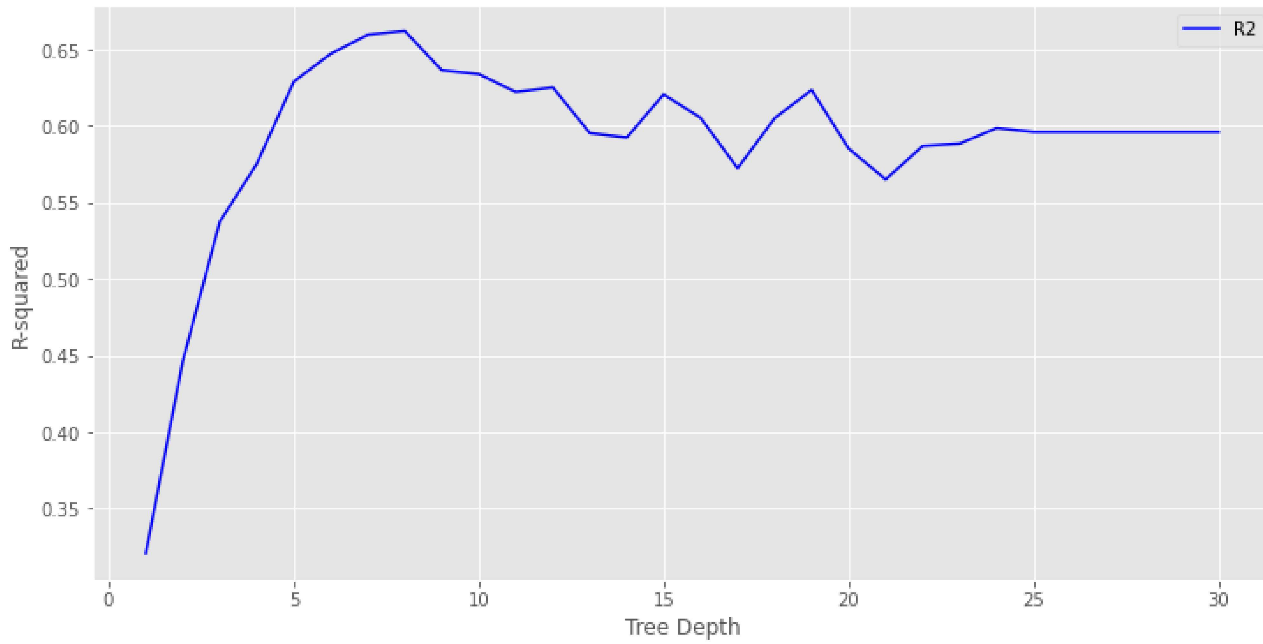
- Find the best tree depth using depth range: 1-30
- Run the regressor repeatedly in a `for` loop for each depth value
- Use `random_state=45` for reproducibility
- Calculate RMSE and r-squared for each run
- Plot both performance measures for all runs
- Comment on the output

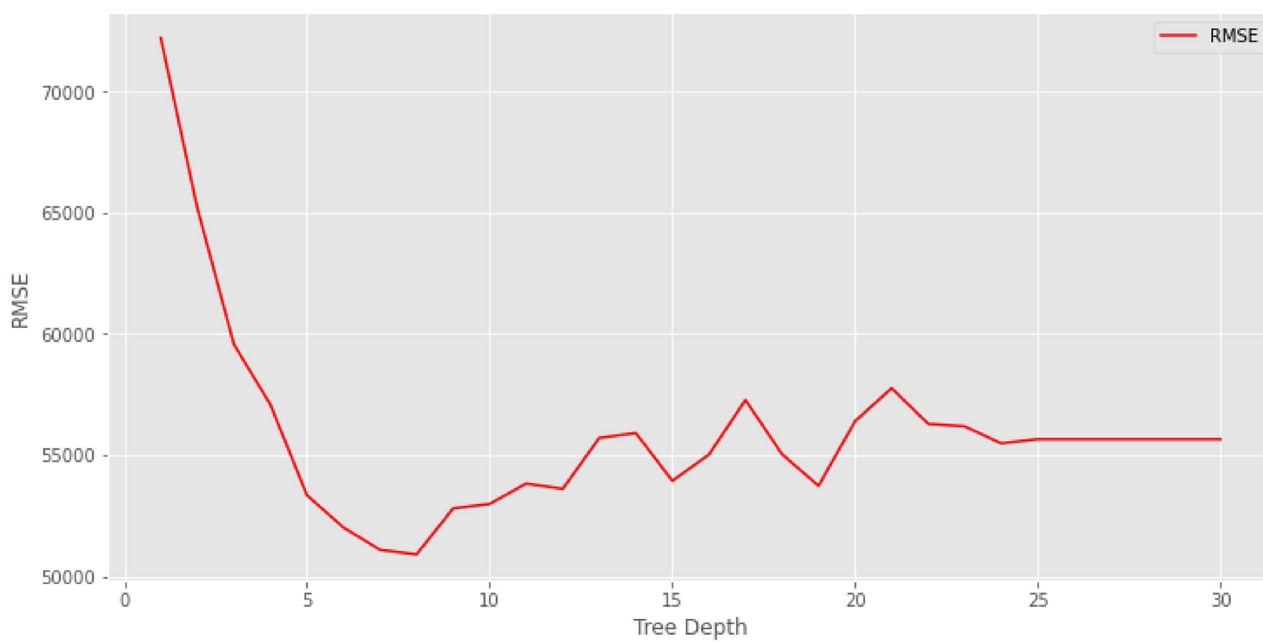
```
# Identify the optimal tree depth for given data
max_depths = list(range(1, 31))
mse_results = []
r2_results = []

for max_depth in max_depths:
    regressor = DecisionTreeRegressor(max_depth=max_depth,
                                      random_state=45)
    regressor.fit(x_train, y_train)
    y_pred = regressor.predict(x_test)
    score = performance(y_test, y_pred)
    r2_results.append(score[0])
    mse_results.append(score[1])

plt.figure(figsize=(12, 6))
plt.plot(max_depths, r2_results, 'b', label='R2')
plt.xlabel('Tree Depth')
plt.ylabel('R-squared')
plt.legend()
plt.show()
plt.figure(figsize=(12, 6))
plt.plot(max_depths, mse_results, 'r', label='RMSE')
```

```
plt.xlabel('Tree Depth')
plt.ylabel('RMSE')
plt.legend()
plt.show()
```





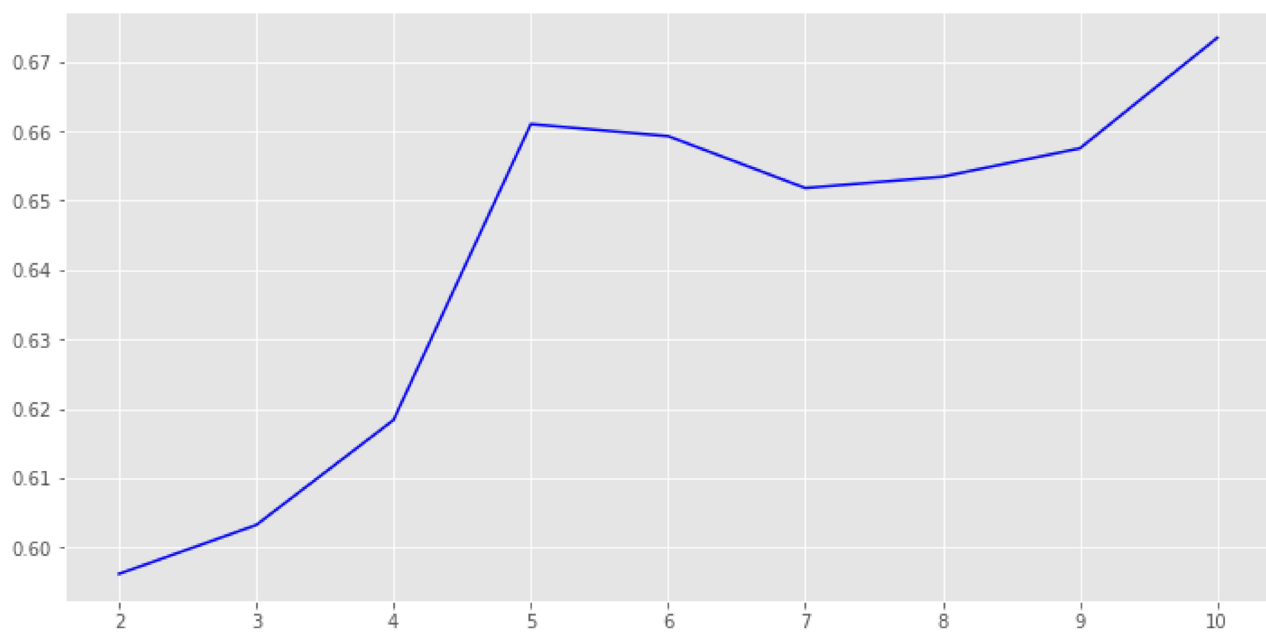
Hyperparameter tuning (II)

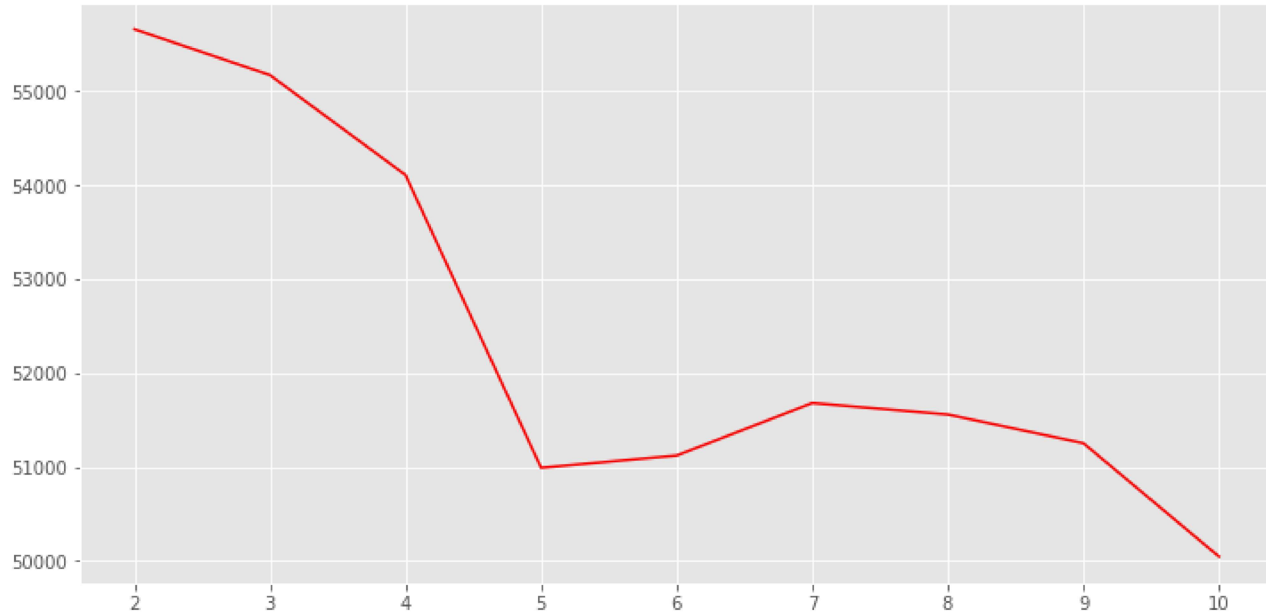
- Repeat the above process for `min_samples_split`
- Use a range of values from 2-10 for this hyperparameter
- Use `random_state=45` for reproducibility
- Visualize the output and comment on results as above

```python
# Identify the optimal minimum split size for given data
min_samples_splits = np.arange(2, 11)
mse_results = []
r2_results = []

for min_samples_split in min_samples_splits:
    regressor = DecisionTreeRegressor(min_samples_split=int(min_samples_split),
                                      random_state=45)
    regressor.fit(x_train, y_train)
    y_pred = regressor.predict(x_test)
    score = performance(y_test, y_pred)
    r2_results.append(score[0])
    mse_results.append(score[1])

plt.figure(figsize=(12, 6))
plt.plot(min_samples_splits, r2_results, 'b', label='R2')
plt.show()
plt.figure(figsize=(12, 6))
plt.plot(min_samples_splits, mse_results, 'r', label='RMSE')
plt.show()
```





Run the *optimized* model

- Use the best values for `max_depth` and `min_samples_split` found in previous runs and run an optimized model with these values
- Calculate the performance and comment on the output

```
regressor = DecisionTreeRegressor(min_samples_split=5, max_depth=7, random_state=45)
regressor.fit(x_train, y_train)
y_pred = regressor.predict(x_test)
score = performance(y_test, y_pred)
score[0], score[1], regressor
```

```
(0.6721318710553857,
 50148.33498676983,
 DecisionTreeRegressor(max_depth=7, min_samples_split=5, random_state=45))
```

Level up (Optional)

- How about bringing in some more features from the original dataset which may be good predictors?
- Also, try tuning more hyperparameters like `max_features` to find a more optimal version of the model

Summary

In this lab, we looked at applying a decision-tree-based regression analysis on the Ames Housing dataset. We saw how to train various models to find the optimal values for hyperparameters.

Releases

No releases published

Packages

No packages published

Contributors 6

Languages

- Jupyter Notebook 100.0%