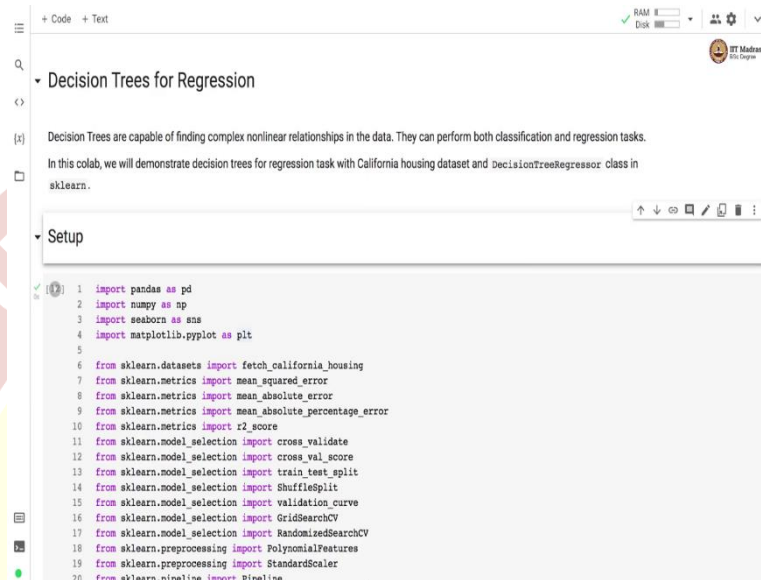


IIT Madras

ONLINE DEGREE

Machine Learning Practice
Professor Dr. Ashish Tendulkar
B. Sc in Programming and Data Science
Indian Institute of Technology, Madras
Decision Trees for Regression

(Refer Slide Time: 0:11)

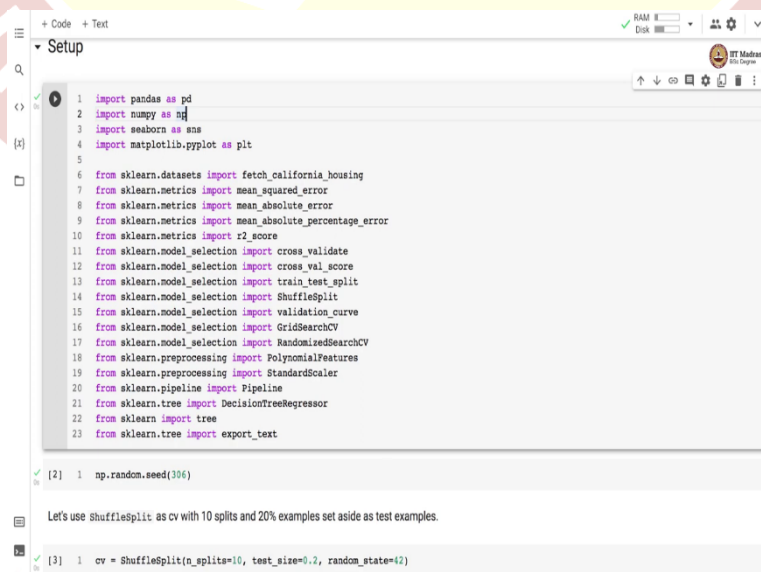


```
+ Code + Text
Decision Trees for Regression
Decision Trees are capable of finding complex nonlinear relationships in the data. They can perform both classification and regression tasks.
In this colab, we will demonstrate decision trees for regression task with California housing dataset and DecisionTreeRegressor class in sklearn.

Setup
1 import pandas as pd
2 import numpy as np
3 import seaborn as sns
4 import matplotlib.pyplot as plt
5
6 from sklearn.datasets import fetch_california_housing
7 from sklearn.metrics import mean_squared_error
8 from sklearn.metrics import mean_absolute_error
9 from sklearn.metrics import mean_absolute_percentage_error
10 from sklearn.metrics import r2_score
11 from sklearn.model_selection import cross_validate
12 from sklearn.model_selection import cross_val_score
13 from sklearn.model_selection import train_test_split
14 from sklearn.model_selection import ShuffleSplit
15 from sklearn.model_selection import validation_curve
16 from sklearn.model_selection import GridSearchCV
17 from sklearn.model_selection import RandomizedSearchCV
18 from sklearn.preprocessing import PolynomialFeatures
19 from sklearn.preprocessing import StandardScaler
20 from sklearn.pipeline import Pipeline
```

Namaste! welcome to the next video of Machine Learning Practice Course. In this video, we will demonstrate decision trees for regression. As you know, decision trees are capable of finding complex nonlinear relationships in the data, they can perform both classification and regression task. In this collab will demonstrate decision trees for regression tasks with California housing dataset, and DecisionTreeRegressor class in sklearn.

(Refer Slide Time: 0:42)



```
+ Code + Text
Setup
1 import pandas as pd
2 import numpy as np
3 import seaborn as sns
4 import matplotlib.pyplot as plt
5
6 from sklearn.datasets import fetch_california_housing
7 from sklearn.metrics import mean_squared_error
8 from sklearn.metrics import mean_absolute_error
9 from sklearn.metrics import mean_absolute_percentage_error
10 from sklearn.metrics import r2_score
11 from sklearn.model_selection import cross_validate
12 from sklearn.model_selection import cross_val_score
13 from sklearn.model_selection import train_test_split
14 from sklearn.model_selection import ShuffleSplit
15 from sklearn.model_selection import validation_curve
16 from sklearn.model_selection import GridSearchCV
17 from sklearn.model_selection import RandomizedSearchCV
18 from sklearn.preprocessing import PolynomialFeatures
19 from sklearn.preprocessing import StandardScaler
20 from sklearn.pipeline import Pipeline
21 from sklearn.tree import DecisionTreeRegressor
22 from sklearn import tree
23 from sklearn.tree import export_text

[2] 1 np.random.seed(306)

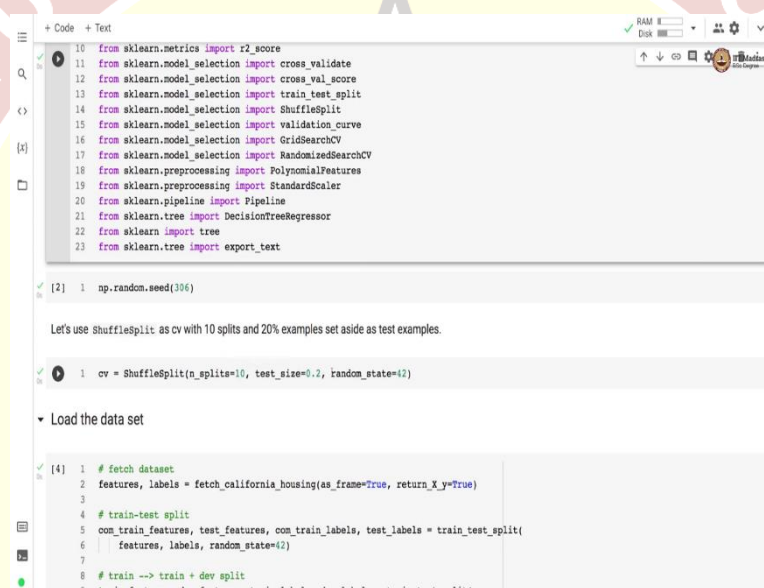
Let's use ShuffleSplit as cv with 10 splits and 20% examples set aside as test examples.

[3] 1 cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=42)
```

Let us begin by importing all necessary libraries, we import a bunch of basic Python libraries for data handling and plotting. Since we are going to use California housing dataset, we import `fetch_California_housing` API from `sklearn . dataset` then bunch of metrics like `mean _squared _error` `mean _absolute _error`, and `mean _absolute _percentage _error` and `r2 _score`.

Then there are a bunch of model selection utilities that are imported, followed by the pre-processing utilities, the pipeline utility, and here we are going to use `DecisionTreeRegressor` as a class, and there is `export tree` and `tree` API's that are imported for visualising the tree in graphical format as well as in the text format.

(Refer Slide Time: 01:35)



```
+ Code + Text
10 from sklearn.metrics import r2_score
11 from sklearn.model_selection import cross_validate
12 from sklearn.model_selection import cross_val_score
13 from sklearn.model_selection import train_test_split
14 from sklearn.model_selection import ShuffleSplit
15 from sklearn.model_selection import validation_curve
16 from sklearn.model_selection import GridSearchCV
17 from sklearn.model_selection import RandomizedSearchCV
18 from sklearn.preprocessing import PolynomialFeatures
19 from sklearn.preprocessing import StandardScaler
20 from sklearn.pipeline import Pipeline
21 from sklearn.tree import DecisionTreeRegressor
22 from sklearn import tree
23 from sklearn.tree import export_text

[2] 1 np.random.seed(306)

Let's use ShuffleSplit as cv with 10 splits and 20% examples set aside as test examples.

1 cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=42)

Load the data set

[4] 1 # fetch dataset
2 features, labels = fetch_california_housing(as_frame=True, return_X_y=True)
3
4 # train-test split
5 com_train_features, test_features, com_train_labels, test_labels = train_test_split(
6     features, labels, random_state=42)
7
8 # train -> train + dev split
9 train_features, dev_features, train_labels, dev_labels = train_test_split(
```

As usual, it is a good practice to set the seed and we set the random seed to 306. In this case, we use `ShuffleSplit` CV as a cross-validation with 10 splits, and 20% example set aside as test examples.

(Refer Slide Time: 01:54)

```
+ Code + Text
RAM 8
Disk 100%

Load the data set

1 # fetch dataset
2 features, labels = fetch_california_housing(as_frame=True, return_X_y=True)
3
4 # train-test split
5 com_train_features, test_features, com_train_labels, test_labels = train_test_split(
6     features, labels, random_state=42)
7
8 # train --> train + dev split
9 train_features, dev_features, train_labels, dev_labels = train_test_split(
10     com_train_features, com_train_labels, random_state=42)

Model setup

[5] 1 dt_reg_pipeline = Pipeline([("feature_scaling", StandardScaler()),
2     ("dt_reg", DecisionTreeRegressor(max_depth=3, random_state=42))])
3 dt_reg_cv_results = cross_validate(dt_reg_pipeline,
4     com_train_features,
5     com_train_labels,
6     cv=cv,
7     scoring="neg_mean_absolute_error",
8     return_train_score=True,
9     return_estimator=True)
10
11 dt_reg_train_error = -1 * dt_reg_cv_results['train_score']
12 dt_reg_test_error = -1 * dt_reg_cv_results['test_score']
13
14 print(f"Mean absolute error of linear regression model on the train set:\n"
15       f"{dt_reg_train_error.mean():.3f} +/- {dt_reg_train_error.std():.3f}")
```

Next, we load the dataset using fetch _California _housing API, we perform training test split, then we perform the further split of the training set into train and development sets.

(Refer Slide Time: 02:10)

```
+ Code + Text
RAM 8
Disk 100%

Model setup

[5] 1 dt_reg_pipeline = Pipeline([("feature_scaling", StandardScaler()),
2     ("dt_reg", DecisionTreeRegressor(max_depth=3, random_state=42))])
3 dt_reg_cv_results = cross_validate(dt_reg_pipeline,
4     com_train_features,
5     com_train_labels,
6     cv=cv,
7     scoring="neg_mean_absolute_error",
8     return_train_score=True,
9     return_estimator=True)
10
11 dt_reg_train_error = -1 * dt_reg_cv_results['train_score']
12 dt_reg_test_error = -1 * dt_reg_cv_results['test_score']
13
14 print(f"Mean absolute error of linear regression model on the train set:\n"
15       f"{dt_reg_train_error.mean():.3f} +/- {dt_reg_train_error.std():.3f}")
16 print(f"Mean absolute error of linear regression model on the test set:\n"
17       f"{dt_reg_test_error.mean():.3f} +/- {dt_reg_test_error.std():.3f}")

Mean absolute error of linear regression model on the train set:
0.590 +/- 0.005
Mean absolute error of linear regression model on the test set:
0.593 +/- 0.007

Visualizing the tree

One of the advantages of using a decision tree classifier is that the output is intuitive to understand and can be easily visualised.

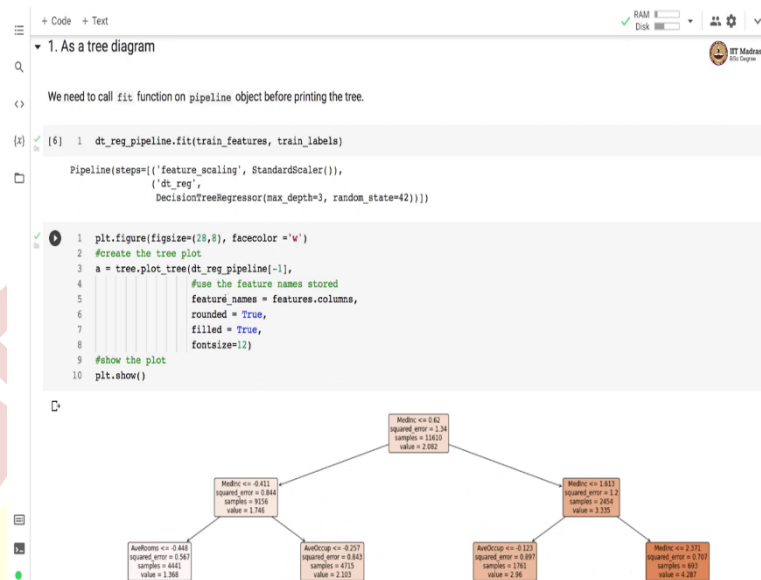
This can be done in two ways:
```

We set up a pipeline model, where we are using feature scaling as a pre-processing step. And for feature scaling, we are using StandardScaler. And then we are defining the DecisionTreeRegressor with max _depth = 3, we perform the training with cross _validate. And here we use all of the training data, we use ShuffleSplit CV and negative mean _absolute _error as a scoring mechanism.

So, after performing the cross _validate regression based training, what we obtain is the matrix on the train set and test set. So, we have the mean _absolute _error of 0.59 on the training set

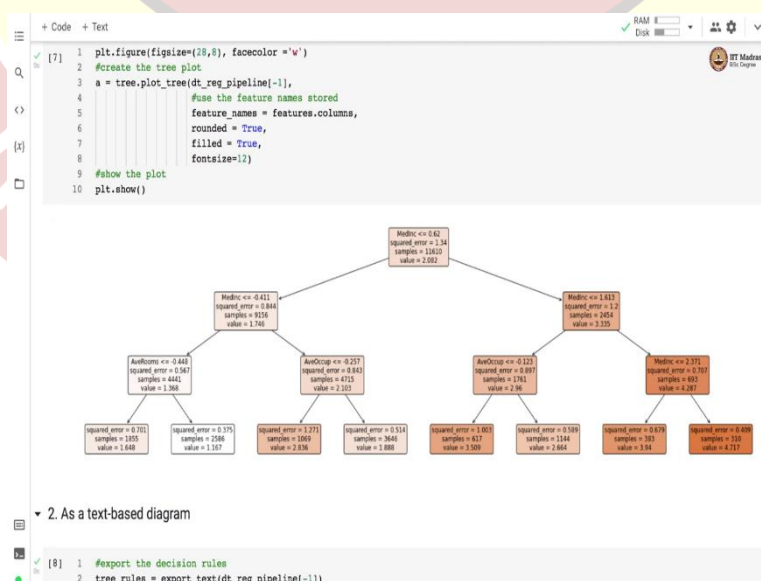
and 0.593 on the test set, the standard deviation is pretty small, which indicates that the mean `_absolute_error` is consistent across different validation sets.

(Refer Slide Time: 03:12)



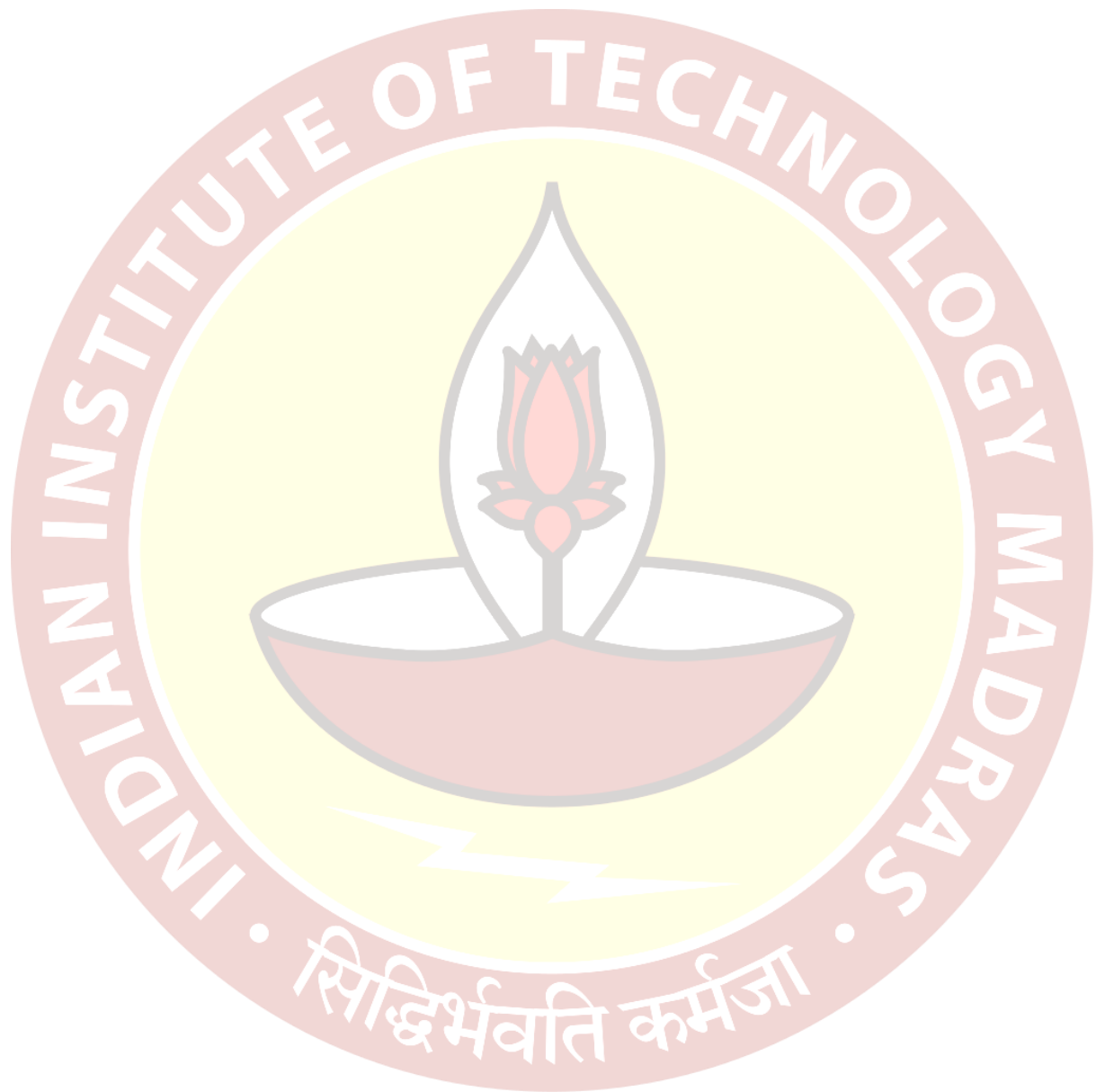
Let us visualize the tree that we have learned through this process. So there are two ways to visualize a tree, using a tree diagram and second is a text based diagram. So we need to call a fit function on the pipeline object before printing the tree otherwise, the print function returns errors. So, here after calling the fit on the pipeline, we use the estimator or specifically, we specify the tree for plot `_tree`. And here we also specify the feature `_names`.

(Refer Slide Time: 03:51)



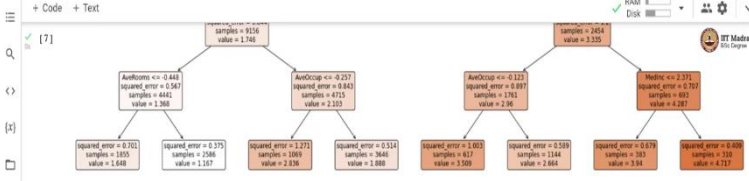
So, what you see on your screen is a tree with depth of 3. And in this tree we see the split criterion which is mid income or median income less than 0.62, there is a split on median

income also in the, at the second level of the tree. Then the third level splits are different for different branches, here we are splitting based on average room size, here based on average occupancy in these two cases. And here again based on median income. And we also print the squared error for each of the node and number of samples and values.



(Refer Slide Time: 04:38)

RAM 8 Disk 100% Jupyter

[7] 

2. As a text-based diagram

```
1 #export the decision rules
2 tree_rules = export_text(dt_reg_pipeline[-1])
3 #print the result
4 print(tree_rules)
```

```
--- feature_0 <= 0.62
|--- feature_0 <= -0.41
|   |--- feature_2 <= -0.45
|   |   |--- value: [1.65]
|   |--- feature_2 > -0.45
|   |   |--- value: [1.17]
|--- feature_0 > -0.41
|   |--- feature_5 <= -0.26
|   |   |--- value: [2.94]
|   |--- feature_5 > -0.26
|   |   |--- value: [1.89]
--- feature_0 > 0.62
|--- feature_0 <= 1.61
|   |--- feature_5 <= -0.12
|   |   |--- value: [3.51]
|   |--- feature_5 > -0.12
|   |   |--- value: [2.66]
--- feature_0 > 1.61
|--- feature_0 <= 2.37
|   |--- value: [3.94]
|--- feature_0 > 2.37
|   |--- value: [4.72]
```

Using the tree for prediction

```
1 test_labels_pred = dt_reg_pipeline.predict(test_features)
```

Evaluating the tree

```
14 mae = mean_absolute_error(test_labels, test_labels_pred)
15 mse = mean_squared_error(test_labels, test_labels_pred)
16 r2 = r2_score(test_labels, test_labels_pred)
17
18 print('The model performance for testing set')
19 print('-----')
20 print('MAE is ', mae)
21 print('MSE is ', mse)
22 print('R2 score is ', r2)
```

The model performance for testing set

```
-----
MAE is 0.45057629282664
MSE is 0.6417557936096145
R2 score is 0.515003769483743
```

Let us now try to improve the model by tuning the hyperparameters.

```
15 param_grid = {'dt_reg_max_depth': range(1, 20),
16               'dt_reg_min_samples_split': range(2, 8)}
17 dt_grid_search = GridSearchCV(dt_reg_pipeline,
18                               param_grid=param_grid,
19                               n_jobs=2,
```

We can also convert this visual representation into text representation by calling `export _test` function. And in `export _test`, we basically get the rule sets. So this is more like, if else conditions as you can convert this tree into this rule set. Then we use this tree for prediction by calling the `predict` function and passing the test feature matrix as input. Based on the predicted labels, we print a bunch of evaluation metrics like `mean _absolute _error`, `mean _squared _error` and `r _score`. The `mean _absolute _error` is 0.60, `mean _squared _error` is 0.64 and `r2 _score` is 0.51.

(Refer Slide Time: 05:27)



```
+ Code + Text
[9] 1 test_labels_pred = dt_reg_pipeline.predict(test_features)

<> Evaluating the tree
[x]
[14] 1 mae = mean_absolute_error(test_labels, test_labels_pred)
2 mse = mean_squared_error(test_labels, test_labels_pred)
3 r2 = r2_score(test_labels, test_labels_pred)
4
5 print('The model performance for testing set')
6 print("-----")
7 print('MAE is ', mae)
8 print('MSE is ', mse)
9 print('R2 score is ', r2)

The model performance for testing set
-----
MAE is  0.6005762942842664
MSE is  0.6417557936098145
R2 score is  0.5150037690483743

Let us now try to improve the model by tuning the hyperparameters.

[15] 1 param_grid = {'dt_reg_max_depth': range(1, 20),
2               'dt_reg_min_samples_split': range(2, 8)}
3 dt_grid_search = GridSearchCV(dt_reg_pipeline,
4                               param_grid=param_grid,
5                               n_jobs=2,
6                               cv=cv,
7                               scoring='neg_mean_absolute_error',
8                               return_train_score=True)
9 dt_grid_search.fit(com_train_features, com_train_labels)

+ Code + Text
[14] 1 r2 = r2_score(test_labels, test_labels_pred)
2
3 print('The model performance for testing set')
4 print("-----")
5 print('MAE is ', mae)
6 print('MSE is ', mse)
7 print('R2 score is ', r2)

The model performance for testing set
-----
MAE is  0.6005762942842664
MSE is  0.6417557936098145
R2 score is  0.5150037690483743

Let us now try to improve the model by tuning the hyperparameters.

[16] 1 param_grid = {'dt_reg_max_depth': range(1, 20),
2               'dt_reg_min_samples_split': range(2, 8)}
3 dt_grid_search = GridSearchCV(dt_reg_pipeline,
4                               param_grid=param_grid,
5                               n_jobs=2,
6                               cv=cv,
7                               scoring='neg_mean_absolute_error',
8                               return_train_score=True)
9 dt_grid_search.fit(com_train_features, com_train_labels)

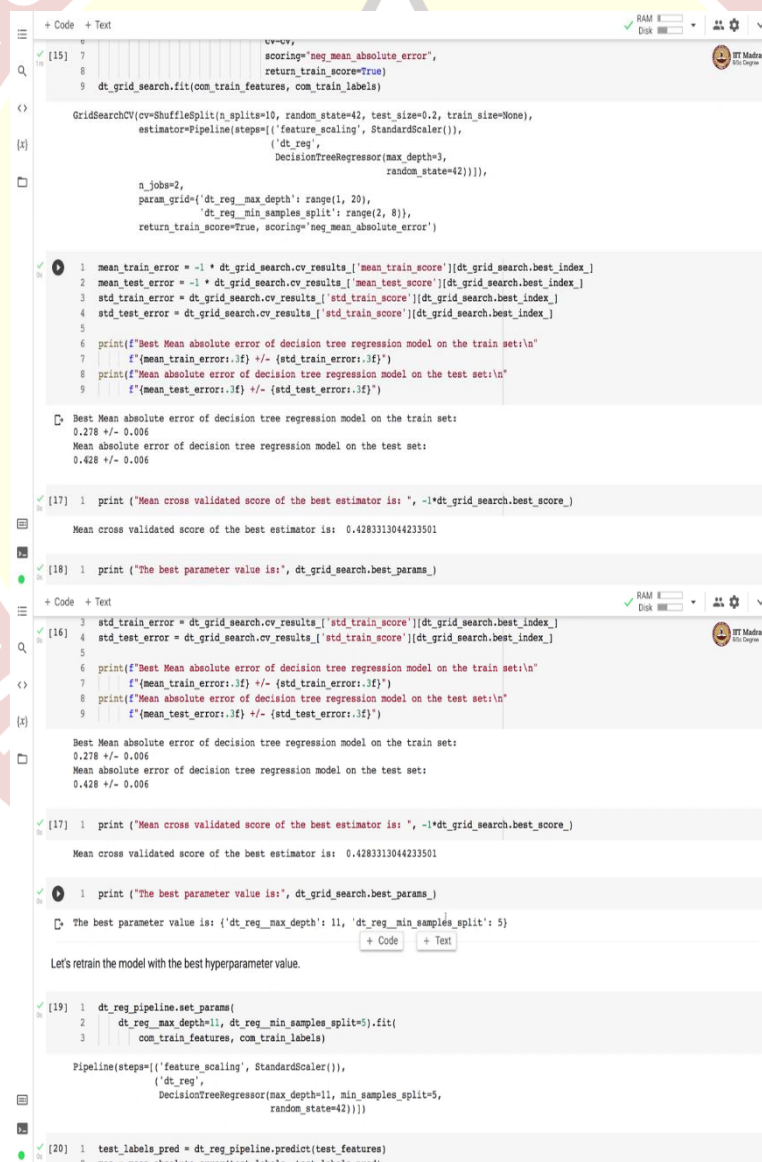
GridSearchCV(cv=ShuffleSplit(n_splits=10, random_state=42, test_size=0.2, train_size=None),
             estimator=Pipeline(steps=[('feature_scaling', StandardScaler()),
                                       ('dt_reg',
                                        DecisionTreeRegressor(max_depth=3,
                                                                random_state=42))]),
             param_grid=[{'dt_reg_max_depth': range(1, 20),
                           'dt_reg_min_samples_split': range(2, 8)}],
             return_train_score=True, scoring='neg_mean_absolute_error')
```

Now, let us try to improve this model by tuning the hyper-parameter. There are two hyper-parameters in case of trees, one is `max _depth` and second is `min _samples _split`. And these two hyper-parameters are of our interest for this particular exercise and we will try to tune them.

So here we define the parameter grid where we want to try max _depth from 1 to 20 and min _samples _split from 2 to 8. We define a GridSearchCV object. So here we are using GridSearchCV for hyper-parameter tuning, we specify the tree estimator and the parameter grid which is specified over here. And we are going to use again the ShuffleSplit CV as a way of performing the cross-validation.

We are using negative mean _absolute _error for scoring. We fit the GridSearchCV object with all of the training data, which includes the combined training features and combined training labels. So we are using the combined training data which has got training as well as development set.

(Refer Slide Time: 06:40)



```
+ Code + Text
[15] 7 cv=ShuffleSplit(n_splits=10, random_state=42, test_size=0.2, train_size=None),
8     scoring='neg_mean_absolute_error',
9     return_train_score=True)
10 dt_grid_search.fit(com_train_features, com_train_labels)

GridSearchCV(cv=ShuffleSplit(n_splits=10, random_state=42, test_size=0.2, train_size=None),
estimator=Pipeline(steps=[('feature_scaling', StandardScaler()),
('dt_reg',
DecisionTreeRegressor(max_depth=3,
random_state=42))]),
n_jobs=2,
param_grid={'dt_reg_max_depth': range(1, 20),
'dt_reg_min_samples_split': range(2, 8)},
return_train_score=True, scoring='neg_mean_absolute_error')

1 mean_train_error = -1 * dt_grid_search.cv_results_['mean_train_score'][dt_grid_search.best_index_]
2 mean_test_error = -1 * dt_grid_search.cv_results_['mean_test_score'][dt_grid_search.best_index_]
3 std_train_error = dt_grid_search.cv_results_['std_train_score'][dt_grid_search.best_index_]
4 std_test_error = dt_grid_search.cv_results_['std_test_score'][dt_grid_search.best_index_]
5
6 print(f'Best Mean absolute error of decision tree regression model on the train set:\n'
7       f'{mean_train_error:.3f} +/- {std_train_error:.3f}')
8 print(f'Mean absolute error of decision tree regression model on the test set:\n'
9       f'{mean_test_error:.3f} +/- {std_test_error:.3f}')

Best Mean absolute error of decision tree regression model on the train set:
0.278 +/- 0.006
Mean absolute error of decision tree regression model on the test set:
0.428 +/- 0.006

[17] 1 print ("Mean cross validated score of the best estimator is: ", -1*dt_grid_search.best_score_)

Mean cross validated score of the best estimator is: 0.4283313044233501

[18] 1 print ("The best parameter value is:", dt_grid_search.best_params_)

+ Code + Text
[16] 3 std_train_error = dt_grid_search.cv_results_['std_train_score'][dt_grid_search.best_index_]
4 std_test_error = dt_grid_search.cv_results_['std_test_score'][dt_grid_search.best_index_]
5
6 print(f'Best Mean absolute error of decision tree regression model on the train set:\n'
7       f'{mean_train_error:.3f} +/- {std_train_error:.3f}')
8 print(f'Mean absolute error of decision tree regression model on the test set:\n'
9       f'{mean_test_error:.3f} +/- {std_test_error:.3f}')

Best Mean absolute error of decision tree regression model on the train set:
0.278 +/- 0.006
Mean absolute error of decision tree regression model on the test set:
0.428 +/- 0.006

[17] 1 print ("Mean cross validated score of the best estimator is: ", -1*dt_grid_search.best_score_)

Mean cross validated score of the best estimator is: 0.4283313044233501

[18] 1 print ("The best parameter value is:", dt_grid_search.best_params_)

The best parameter value is: {'dt_reg_max_depth': 11, 'dt_reg_min_samples_split': 5}

Let's retrain the model with the best hyperparameter value.

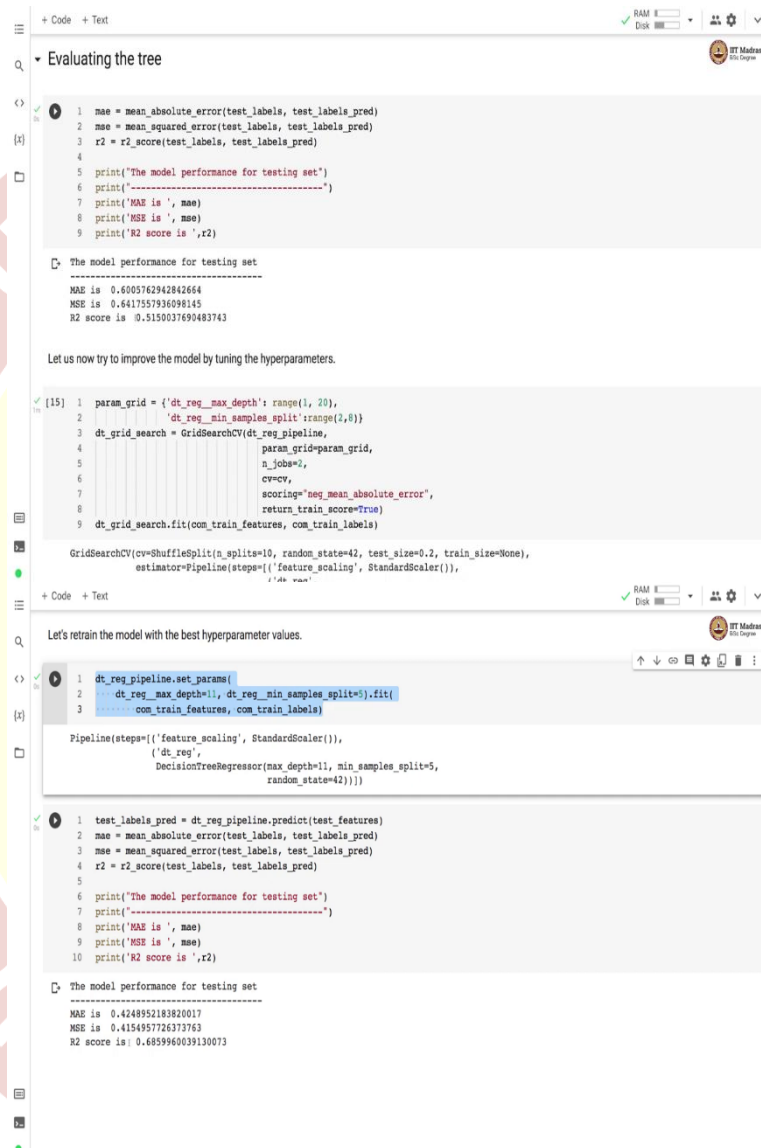
[19] 1 dt_reg_pipeline.set_params(
2     dt_reg_max_depth=11, dt_reg_min_samples_split=5).fit(
3     com_train_features, com_train_labels)

Pipeline(steps=[('feature_scaling', StandardScaler()),
('dt_reg',
DecisionTreeRegressor(max_depth=11, min_samples_split=5,
random_state=42))])

[20] 1 test_labels_pred = dt_reg_pipeline.predict(test_features)
2 max = mean_absolute_error(test_labels, test_labels_pred)
```

After performing the hyper-parameter search, we found out the best mean `_absolute _error` obtained on the training set as well as on the test set. So you can see that on the training set, the best mean `_absolute _error` that is obtained is 0.278. And on test set, it is 0.428. And the best parameter values for the tree are the `max _depth = 11` and `mean _samples _split = 5`.

(Refer Slide Time: 07:07)



```
+ Code + Text
Evaluating the tree

1 mae = mean_absolute_error(test_labels, test_labels_pred)
2 mse = mean_squared_error(test_labels, test_labels_pred)
3 r2 = r2_score(test_labels, test_labels_pred)
4
5 print('The model performance for testing set')
6 print('-----')
7 print('MAE is ', mae)
8 print('MSE is ', mse)
9 print('R2 score is ', r2)

The model performance for testing set
MAE is 0.6005762942842664
MSE is 0.6417557936098145
R2 score is 0.5150037690483743

Let us now try to improve the model by tuning the hyperparameters.

[15]: 1 param_grid = {'dt_reg_max_depth': range(1, 20),
2               'dt_reg_min_samples_split': range(2, 8)}
3 dt_grid_search = GridSearchCV(dt_reg_pipeline,
4                               param_grid=param_grid,
5                               n_jobs=2,
6                               cv=cv,
7                               scoring='neg_mean_absolute_error',
8                               return_train_score=True)
9 dt_grid_search.fit(com_train_features, com_train_labels)

GridSearchCV(cv=ShuffleSplit(n_splits=10, random_state=42, test_size=0.2, train_size=None),
             estimator=Pipeline(steps=[('feature_scaling', StandardScaler()),
                                       ('dt_reg',
                                        DecisionTreeRegressor(max_depth=11, min_samples_split=5,
                                                                random_state=42))]))

+ Code + Text
Let's retrain the model with the best hyperparameter values.

1 dt_reg_pipeline.set_params(
2     dt_reg_max_depth=11, dt_reg_min_samples_split=5).fit(
3     com_train_features, com_train_labels)

Pipeline(steps=[('feature_scaling', StandardScaler()),
                ('dt_reg',
                 DecisionTreeRegressor(max_depth=11, min_samples_split=5,
                                       random_state=42))])

1 test_labels_pred = dt_reg_pipeline.predict(test_features)
2 mae = mean_absolute_error(test_labels, test_labels_pred)
3 mse = mean_squared_error(test_labels, test_labels_pred)
4 r2 = r2_score(test_labels, test_labels_pred)
5
6 print('The model performance for testing set')
7 print('-----')
8 print('MAE is ', mae)
9 print('MSE is ', mse)
10 print('R2 score is ', r2)

The model performance for testing set
MAE is 0.4248952183820017
MSE is 0.4154957726373763
R2 score is 0.6859960039130073
```

We retrain our model with the best hyper-parameter values, these best hyper-parameter values are set over here, and then we call the fit function on our pipeline. And then we obtain the retrain pipeline with the best hyper-parameter values. We perform the prediction on the test set and calculate various metrics on the test set in order to evaluate the retrained pipeline.

Here we see that the mean `_absolute _error` has gone down to 0.42. And the `r2 _score` has improved to 0.68. So if we compare this with the values that we were having over here, here

the mean _absolute _error was 0.60 that is now reduced to 0.42. And r2 _score has jumped from 0.51 to 0.68.

So, in this video, we use decision trees for regression problem with California Housing dataset. We also demonstrate how to perform hyper-parameter search, and saw that after performing hyper-parameter search and retraining the pipeline, some of the metrics like r2 _score, mean _absolute _error, and mean _squared _errors improved.

