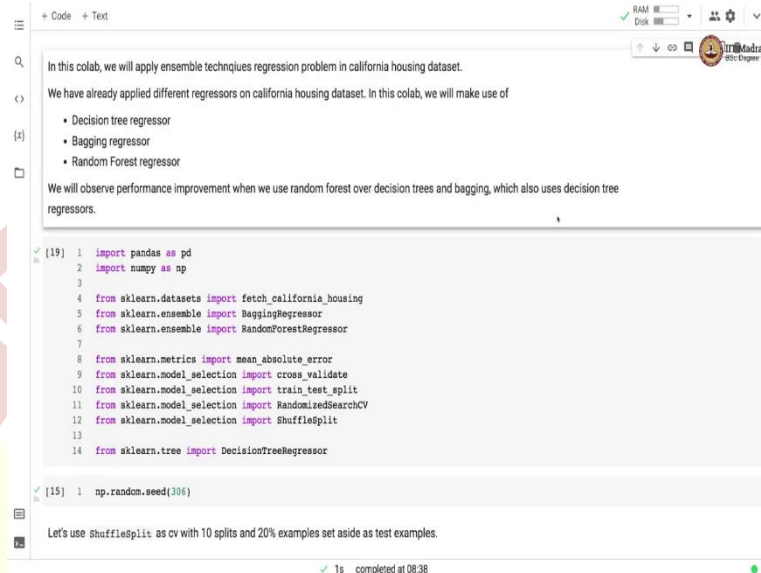


IIT Madras

ONLINE DEGREE

Machine Learning Practice
Professor. Ashish Tendulkar
Indian Institute of Technology, Madras
Bagging and Random Forest Regressor on California Housing Dataset

(Refer Slide Time: 0:10)



```
[19]: 1 import pandas as pd
      2 import numpy as np
      3
      4 from sklearn.datasets import fetch_california_housing
      5 from sklearn.ensemble import BaggingRegressor
      6 from sklearn.ensemble import RandomForestRegressor
      7
      8 from sklearn.metrics import mean_absolute_error
      9 from sklearn.model_selection import cross_validate
     10 from sklearn.model_selection import train_test_split
     11 from sklearn.model_selection import RandomizedSearchCV
     12 from sklearn.model_selection import ShuffleSplit
     13
     14 from sklearn.tree import DecisionTreeRegressor
```

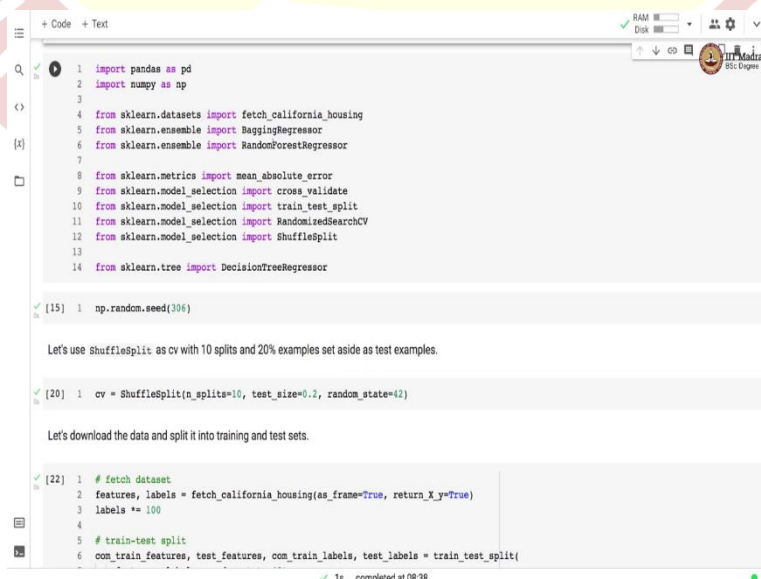
[15]: 1 np.random.seed(306)

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

Namaste! Welcome to the next video of Machine Learning Practice Course. In this video, we will apply ensemble technique to a regression problem in California Housing Dataset. We are already applied different regressors on California housing dataset in this course. In this collab, we will make use of decision tree regressor, bagging regressor and random forest regressor.

We will observe performance improvement when we use random forest over decision tree and bagging and bagging also uses decision tree regressor internally.

(Refer Slide Time: 0:51)



```
[15]: 1 np.random.seed(306)
```

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

```
[20]: 1 cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=42)
```

Let's download the data and split it into training and test sets.

```
[22]: 1 # fetch dataset
      2 features, labels = fetch_california_housing(as_frame=True, return_X_y=True)
      3 labels -= 100
      4
      5 # train-test split
      6 com_train_features, test_features, com_train_labels, test_labels = train_test_split(
```

Then we use `mean_absolute_error` as a metric. And for model selection we are going to use `cross_validate`, `train_test_split`, `RandomizedSearchCV` and `ShuffleSplit`.

```
+ Code + Text
12 from sklearn.model_selection import ShuffleSplit
13
14 from sklearn.tree import DecisionTreeRegressor

[15] 1 np.random.seed(396)

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

[20] 1 cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=42)

Let's download the data and split it into training and test sets.

[22] 1 # fetch dataset
2 features, labels = fetch_california_housing(as_frame=True, return_X_y=True)
3 labels == 100
4
5 # train-test split
6 com_train_features, test_features, com_train_labels, test_labels = train_test_split(
7     features, labels, random_state=42)
8
9 # train --> train + dev split
10 train_features, dev_features, train_labels, dev_labels = train_test_split(
11     com_train_features, com_train_labels, random_state=42)
```

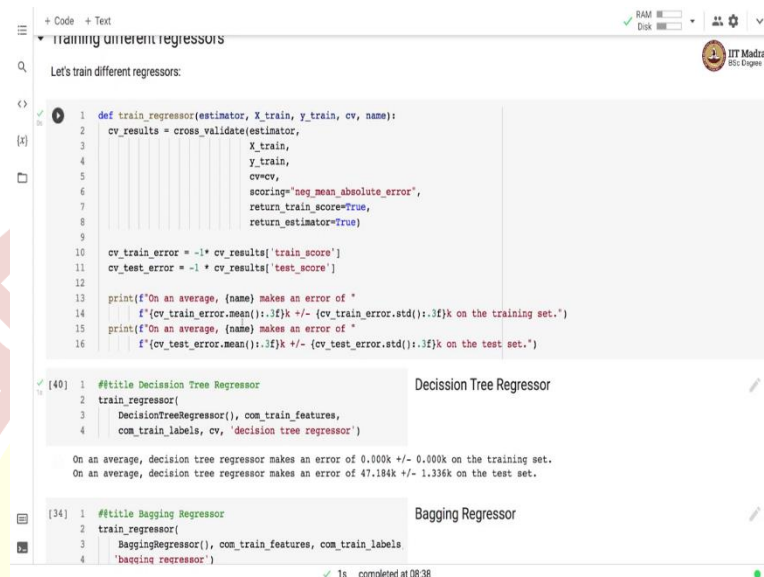
```
+ Code + Text
[20] 1 cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=42)

Let's download the data and split it into training and test sets.

1 # fetch dataset
2 features, labels = fetch_california_housing(as_frame=True, return_X_y=True)
3 labels -= 100
4
5 # train-test split
6 com_train_features, test_features, com_train_labels, test_labels = train_test_split(
7     features, labels, random_state=42)
8
9 # train -> train + dev split
10 train_features, dev_features, train_labels, dev_labels = train_test_split(
11     com_train_features, com_train_labels, random_state=42)
```

Let us download the data and split it into training and test set. So, we downloaded the data with `fetch_california_housing`, we \times label by 100. So, that we get values in 1000 of dollars, then we perform training and test split and we also perform training and their split.

(Refer Slide Time: 02:06)



```
def train_regressor(estimator, X_train, y_train, cv, name):
    cv_results = cross_validate(estimator,
                                X_train,
                                y_train,
                                cv=cv,
                                scoring="neg_mean_absolute_error",
                                return_train_score=True,
                                return_estimator=True)

    cv_train_error = -1 * cv_results['train_score']
    cv_test_error = -1 * cv_results['test_score']

    print(f"On an average, {name} makes an error of "
          f"{cv_train_error.mean():.3f}k +/- {cv_train_error.std():.3f}k on the training set.")
    print(f"On an average, {name} makes an error of "
          f"{cv_test_error.mean():.3f}k +/- {cv_test_error.std():.3f}k on the test set.")

[40]: #title Decision Tree Regressor
      train_regressor(
        DecisionTreeRegressor(), com_train_features,
        com_train_labels, cv, 'decision tree regressor')

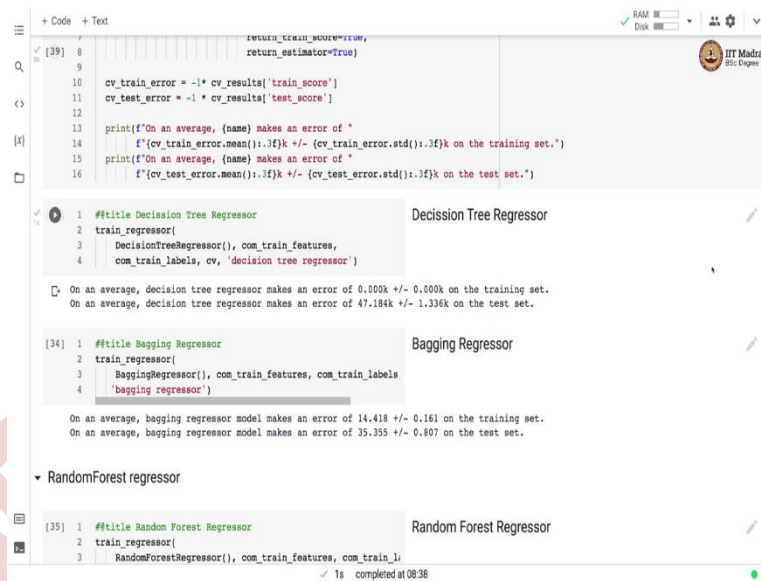
On an average, decision tree regressor makes an error of 0.000k +/- 0.000k on the training set.
On an average, decision tree regressor makes an error of 47.184k +/- 1.336k on the test set.

[34]: #title Bagging Regressor
      train_regressor(
        BaggingRegressor(), com_train_features, com_train_labels,
        'bagging regressor')
```

In order to train different regressors, we define a `train_regressor` function that performs training of the regressor with cross-validation, it takes estimator as an argument along with training feature matrix, training label vector, the cross-validation, the scoring which is negative mean absolute error, and we said return `train_score` and return estimator both these flags to true.

Then we get the training error by \times the `train_score` by -1 and test_error by \times test_score by -1. Remember the score here is negative mean absolute error and hence we \times it by -1. Then we print the error on training and test sets.

(Refer Slide Time: 02:58)



```
[39] 8 return_estimator=True,
9 return_estimator=True)
10 cv_train_error = -1 * cv_results['train_score']
11 cv_test_error = -1 * cv_results['test_score']
12
13 print(f"On an average, {name} makes an error of "
14       f"{cv_train_error.mean():.3f}k +/- {cv_train_error.std():.3f}k on the training set.")
15 print(f"On an average, {name} makes an error of "
16       f"{cv_test_error.mean():.3f}k +/- {cv_test_error.std():.3f}k on the test set.")

[34] 1 #title Decision Tree Regressor
2 train_regressor(
3     DecisionTreeRegressor(), com_train_features,
4     com_train_labels, cv, 'decision tree regressor')

On an average, decision tree regressor makes an error of 0.000k +/- 0.000k on the training set.
On an average, decision tree regressor makes an error of 47.184k +/- 1.336k on the test set.

[34] 1 #title Bagging Regressor
2 train_regressor(
3     BaggingRegressor(), com_train_features, com_train_labels,
4     'bagging regressor')

On an average, bagging regressor model makes an error of 14.418 +/- 0.161 on the training set.
On an average, bagging regressor model makes an error of 35.355 +/- 0.807 on the test set.

Random Forest regressor

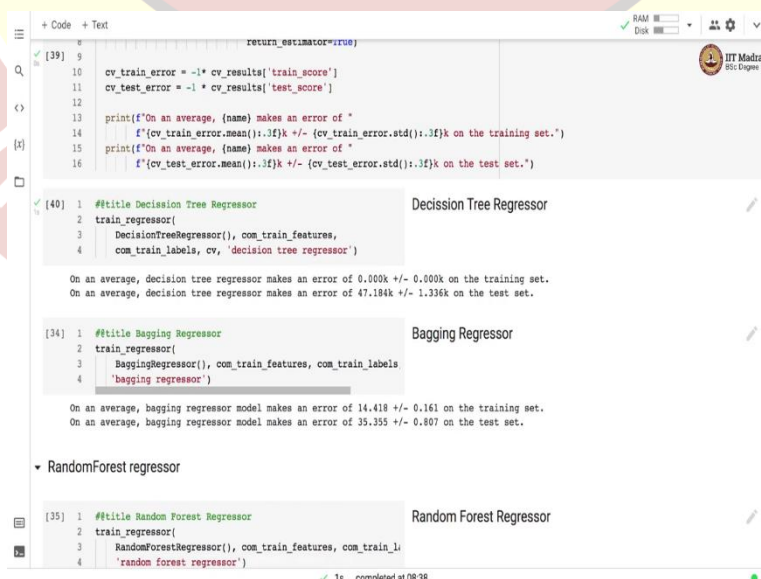
[35] 1 #title Random Forest Regressor
2 train_regressor(
3     RandomForestRegressor(), com_train_features, com_train_labels,
4     'random forest regressor')

1s completed at 08:38
```

We are going to train 3 regressors, Decision Tree Regressor, Bagging Regressor and Random Forest Regressor. After defining this function, this is quite a straightforward task we call the function `train_regressor` by passing the appropriate estimator which is `DecisionTreeRegressor` here the training feature matrix and training label vector along with the cross-validation scheme and the name of the regressor so that we can print it appropriately in the output.

So, you can see here that the `DecisionTreeRegressor` makes 0 error on the training set, whereas it makes error of 47.18k on the test set with standard deviation of 1.33k.

(Refer Slide Time: 03:45)



```
[39] 8 return_estimator=True,
9 return_estimator=True)
10 cv_train_error = -1 * cv_results['train_score']
11 cv_test_error = -1 * cv_results['test_score']
12
13 print(f"On an average, {name} makes an error of "
14       f"{cv_train_error.mean():.3f}k +/- {cv_train_error.std():.3f}k on the training set.")
15 print(f"On an average, {name} makes an error of "
16       f"{cv_test_error.mean():.3f}k +/- {cv_test_error.std():.3f}k on the test set.")

[40] 1 #title Decision Tree Regressor
2 train_regressor(
3     DecisionTreeRegressor(), com_train_features,
4     com_train_labels, cv, 'decision tree regressor')

On an average, decision tree regressor makes an error of 0.000k +/- 0.000k on the training set.
On an average, decision tree regressor makes an error of 47.184k +/- 1.336k on the test set.

[34] 1 #title Bagging Regressor
2 train_regressor(
3     BaggingRegressor(), com_train_features, com_train_labels,
4     'bagging regressor')

On an average, bagging regressor model makes an error of 14.418 +/- 0.161 on the training set.
On an average, bagging regressor model makes an error of 35.355 +/- 0.807 on the test set.

Random Forest regressor

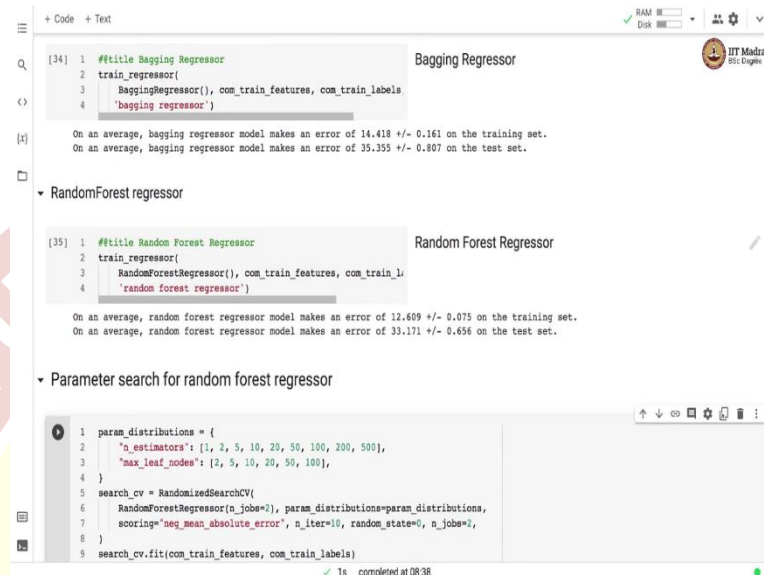
[35] 1 #title Random Forest Regressor
2 train_regressor(
3     RandomForestRegressor(), com_train_features, com_train_labels,
4     'random forest regressor')

1s completed at 08:38
```

So, you can see that this is an example of over fitted model because it obtains a 0 error on the training, but quite high error on the test set. And this precise problem of over fitting is solved

with bagging and random forest. Then we trained BaggingRegressor and when we train the BaggingRegressor, model makes error 14.41 on the training set, but the error on the test set is reduced from 47.18 to 35.35.

(Refer Slide Time: 04:20)



The screenshot shows a Jupyter Notebook with two code cells. The first cell, titled 'Bagging Regressor', contains the following code:

```
[34] 1 #title Bagging Regressor
2 train_regressor(
3     BaggingRegressor(), com_train_features, com_train_labels
4     'bagging regressor')
```

Below the code, the output shows the mean cross-validated score for the BaggingRegressor model:

```
On an average, bagging regressor model makes an error of 14.418 +/- 0.161 on the training set.
On an average, bagging regressor model makes an error of 35.355 +/- 0.807 on the test set.
```

The second cell, titled 'Random Forest Regressor', contains the following code:

```
[35] 1 #title Random Forest Regressor
2 train_regressor(
3     RandomForestRegressor(), com_train_features, com_train_labels
4     'random forest regressor')
```

Below the code, the output shows the mean cross-validated score for the RandomForestRegressor model:

```
On an average, random forest regressor model makes an error of 12.609 +/- 0.075 on the training set.
On an average, random forest regressor model makes an error of 33.171 +/- 0.656 on the test set.
```

The third cell, titled 'Parameter search for random forest regressor', contains the following code:

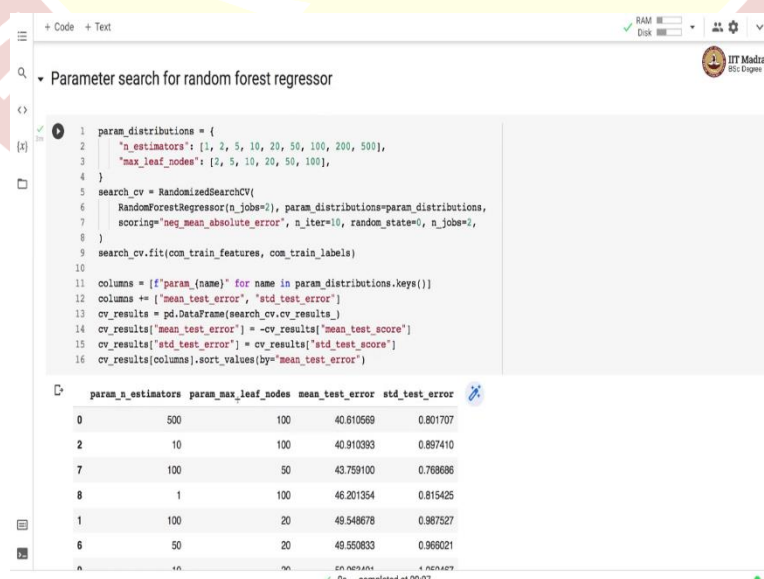
```
1 param_distributions = {
2     "n_estimators": [1, 2, 5, 10, 20, 50, 100, 200, 500],
3     "max_leaf_nodes": [2, 5, 10, 20, 50, 100],
4 }
5 search_cv = RandomizedSearchCV(
6     RandomForestRegressor(n_jobs=-1), param_distributions=param_distributions,
7     scoring="neg_mean_absolute_error", n_iter=10, random_state=0, n_jobs=-1,
8 )
9 search_cv.fit(com_train_features, com_train_labels)
```

The output shows the results of the parameter search:

```
1s completed at 08:38
```

And when we apply a RandomForestRegressor the errors get reduced both on the training set as well as on the test set. Now on the training set we have error of 12.609 whereas on the test set we have error of 33.17. So, you can see that as we use BaggingRegressor and RandomForestRegressor the error has gone down on the test set with marginal increase in the training error. And this is the exact point that we were talking in the theory class that bagging helps us to reduce the variance or over fitting in the base classifiers.

(Refer Slide Time: 05:05)



The screenshot shows a Jupyter Notebook with a code cell titled 'Parameter search for random forest regressor'. The code is as follows:

```
1 param_distributions = {
2     "n_estimators": [1, 2, 5, 10, 20, 50, 100, 200, 500],
3     "max_leaf_nodes": [2, 5, 10, 20, 50, 100],
4 }
5 search_cv = RandomizedSearchCV(
6     RandomForestRegressor(n_jobs=-1), param_distributions=param_distributions,
7     scoring="neg_mean_absolute_error", n_iter=10, random_state=0, n_jobs=-1,
8 )
9 search_cv.fit(com_train_features, com_train_labels)
10
11 columns = ["param_{}".format(name) for name in param_distributions.keys()]
12 columns += ["mean_test_error", "std_test_error"]
13 cv_results = pd.DataFrame(search_cv.cv_results_)
14 cv_results["mean_test_error"] = -cv_results["mean_test_score"]
15 cv_results["std_test_error"] = cv_results["std_test_score"]
16 cv_results[columns].sort_values(by="mean_test_error")
```

The output shows the results of the parameter search as a table:

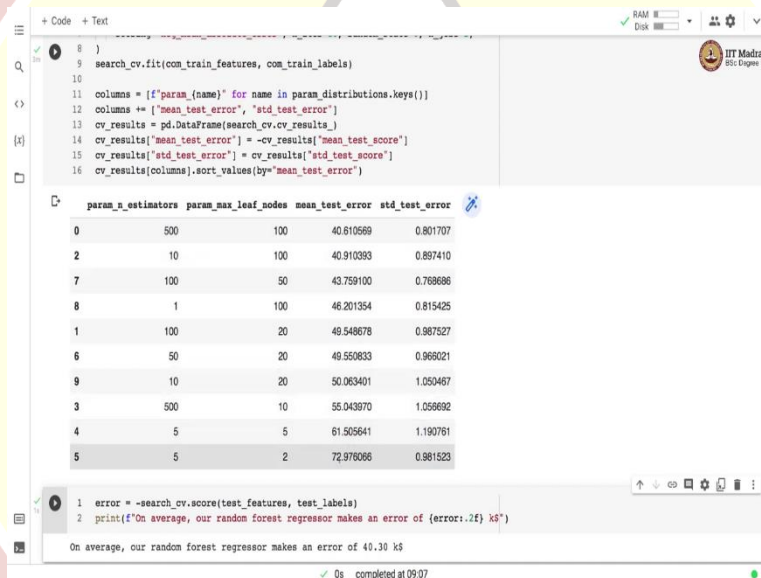
| | param_n_estimators | param_max_leaf_nodes | mean_test_error | std_test_error |
|---|--------------------|----------------------|-----------------|----------------|
| 0 | 500 | 100 | 40.810569 | 0.801707 |
| 2 | 10 | 100 | 40.910393 | 0.897410 |
| 7 | 100 | 50 | 43.759100 | 0.768686 |
| 8 | 1 | 100 | 46.201354 | 0.815425 |
| 1 | 100 | 20 | 49.548678 | 0.987327 |
| 6 | 50 | 20 | 49.550833 | 0.966021 |

The output also shows the execution time: 0s completed at 09:07.

We demonstrate how to perform parameter search for random forest regressor. There are 2 configurable parameters, number of estimators and maximum number of leaf nodes. We define a parameter distribution for number of estimators with number of estimators to be 1, 2, 5, 10, 20, 50, 100, 200 and 500. We said the maximum number of leaf nodes to 2, 5, 10, 20, 50 and 100. We perform RandomizedSearchCV with this parameter distribution, and RandomForestRegressor estimator.

We use negative mean absolute error as a scoring function. And we perform 10 iterations of RandomizedSearchCV. After performing the RandomizedSearchCV, we have displayed the mean _test _error and the standard deviation in the test _error for different parameter configurations.

(Refer Slide Time: 06:09)



```

8 )
9 search_cv.fit(com_train_features, com_train_labels)
10
11 columns = [f"param_{name}" for name in param_distributions.keys()]
12 columns += ["mean_test_error", "std_test_error"]
13 cv_results = pd.DataFrame(search_cv.cv_results_)
14 cv_results["mean_test_error"] = -cv_results["mean_test_score"]
15 cv_results["std_test_error"] = cv_results["std_test_score"]
16 cv_results[columns].sort_values(by="mean_test_error")

```

| | param_n_estimators | param_max_leaf_nodes | mean_test_error | std_test_error |
|---|--------------------|----------------------|-----------------|----------------|
| 0 | 500 | 100 | 40.810569 | 0.801707 |
| 2 | 10 | 100 | 40.910393 | 0.897410 |
| 7 | 100 | 50 | 43.759100 | 0.788686 |
| 8 | 1 | 100 | 46.201354 | 0.815425 |
| 1 | 100 | 20 | 49.548678 | 0.987527 |
| 6 | 50 | 20 | 49.550833 | 0.966021 |
| 9 | 10 | 20 | 50.063401 | 1.050467 |
| 3 | 500 | 10 | 55.043970 | 1.056692 |
| 4 | 5 | 5 | 61.505841 | 1.190761 |
| 5 | 5 | 2 | 72.976086 | 0.981523 |

```

1 error = -search_cv.score(test_features, test_labels)
2 print(f'On average, our random forest regressor makes an error of {error:.2f} k$')

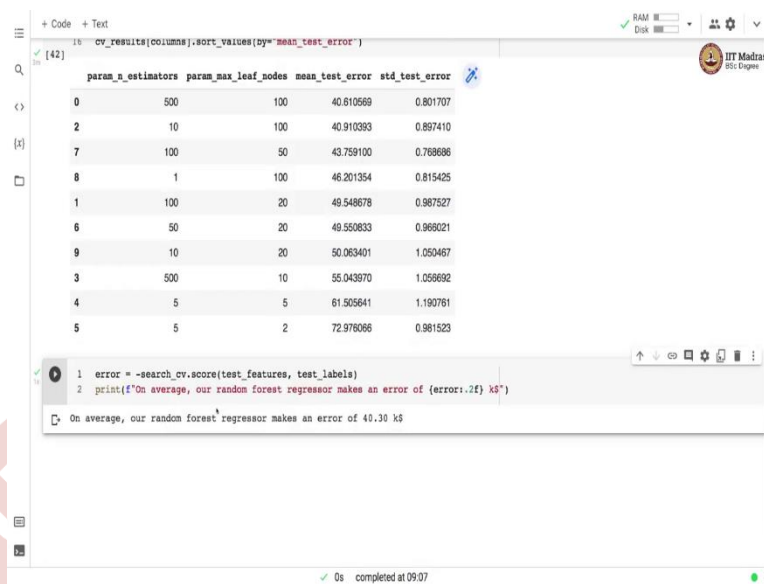
```

On average, our random forest regressor makes an error of 40.30 k\$

0s completed at 09:07

And we are assured this parameter configuration in the ascending order of the test _error. So, the best performance was obtained with number of estimators =500 and maximum number of leaf nodes =100. The maximum test _error was 72.97 and was obtained for number of estimators =5 and maximum number of leaf nodes =2.

(Refer Slide Time: 06:39)



| | param_n_estimators | param_max_leaf_nodes | mean_test_error | std_test_error |
|---|--------------------|----------------------|-----------------|----------------|
| 0 | 500 | 100 | 40.610669 | 0.801707 |
| 2 | 10 | 100 | 40.910393 | 0.897410 |
| 7 | 100 | 50 | 43.759100 | 0.768686 |
| 8 | 1 | 100 | 46.201354 | 0.815425 |
| 1 | 100 | 20 | 49.548678 | 0.987527 |
| 6 | 50 | 20 | 49.550833 | 0.969021 |
| 9 | 10 | 20 | 50.063401 | 1.050467 |
| 3 | 500 | 10 | 55.043970 | 1.056692 |
| 4 | 5 | 5 | 61.505641 | 1.190761 |
| 5 | 5 | 2 | 72.978066 | 0.981523 |

```
1 error = -search_cv.score(test_features, test_labels)
2 print(f'On average, our random forest regressor makes an error of {error:.2f} k$')
```

On average, our random forest regressor makes an error of 40.30 k\$

Next, we calculate the error on the test set. So, on an average, a RandomForestRegressor makes an error of 40.30k on the test set, and this RandomForestRegressor is obtained to RandomizedSearchCV. So, in this video, we studied how to apply bagging techniques for regression problem. Specifically, we applied Decision Tree, Bagging Regressor and Random Forest Regressor on California Housing Dataset.

We use DecisionTreeRegressor to understand what is the baseline regressor on the dataset and then use bagging and RandomForestRegressor to show that we are able to reduce the over fitting effect in the DecisionTreeRegressor using the bagging technique. We also demonstrated how to perform randomized search for the best set of parameters for RandomForestRegressor.