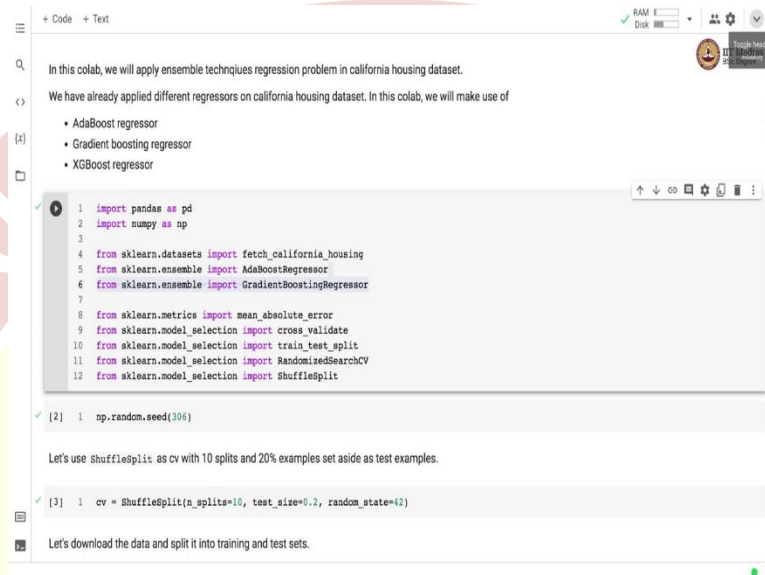


IIT Madras
ONLINE DEGREE

Machine Learning Practice
Professor. Ashish Tendulkar
Indian Institute of Technology, Madras
AdaBoost and GradientBoost Regressor on California Housing

(Refer Slide Time: 00:10)



```
1 import pandas as pd
2 import numpy as np
3
4 from sklearn.datasets import fetch_california_housing
5 from sklearn.ensemble import AdaBoostRegressor
6 from sklearn.ensemble import GradientBoostingRegressor
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

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

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

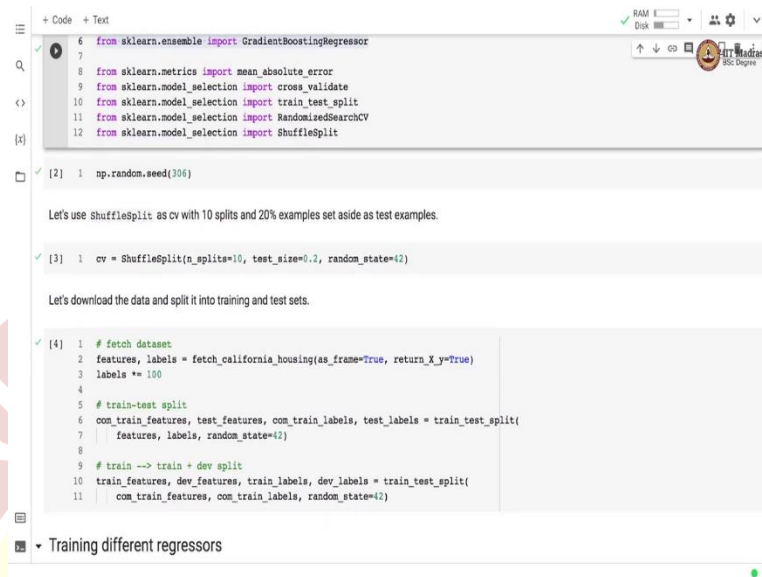
[3] 1 cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=42)

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

Namaste! Welcome to the next video of Machine Learning Practice Course. In this video, will apply boosting technique for California housing dataset, which is a dataset with regression problem. In this problem, we tried to predict the price of the house based on various attributes. We have already applied different regressions on California Housing dataset.

In this collab will make use of boosting regressors mainly AdaBoost, gradient boosting and XGBoost, AdaBoost and GradientBoostingRegressor are available in sklearn.ensemble module. Whereas, XGBoost regressor is not available directly in sklearn. But there is a separate library for XGBoost. And we will talk about XGBoost towards end of this collab.

(Refer Slide Time: 01:07)



```
+ Code + Text
6 from sklearn.ensemble import GradientBoostingRegressor
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

[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)

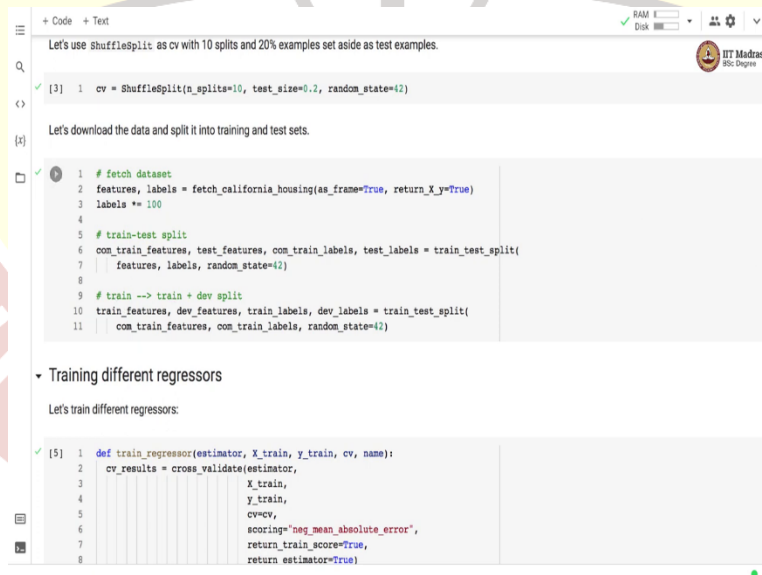
Let's download the data and split it into training and test sets.

[4] 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)

Training different regressors
```

We start by assigning a random seed of 306. And this random seed helps us to get the reproducible result in the collab. We will use ShuffleSplit as a cross-validation with 10 splits and 20% examples set aside as test examples. Let us download the data and split it into training and test sets.

(Refer Slide Time: 01:32)



```
+ Code + Text
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's download the data and split it into training and test sets.

[4] 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)

Training different regressors
Let's train different regressors:

[5] 1 def train_regressor(estimator, X_train, y_train, cv, name):
2     cv_results = cross_validate(estimator,
3                               X_train,
4                               y_train,
5                               cv=cv,
6                               scoring="neg_mean_absolute_error",
7                               return_train_score=True,
8                               return_estimator=True)
```

So, we fetch California Housing dataset and divide that into training and test as well as the training data is divided into train and dev.

(Refer Slide Time: 01:43)

```
+ Code + Text
[4] 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)

[5] Training different regressors
Let's train different regressors:

[6] 1 def train_regressor(estimator, X_train, y_train, cv, name):
2     cv_results = cross_validate(estimator,
3                                 X_train,
4                                 y_train,
5                                 cv=cv,
6                                 scoring="neg_mean_absolute_error",
7                                 return_train_score=True,
8                                 return_estimator=True)
9
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.')

[6] 1 #title AdaBoostRegressor
2     train_regressor(
3         AdaBoostRegressor(), com_train_features,
4         com_train_labels, cv, 'AdaBoostRegressor')
```

Then we trained different regressors and for training different regressors we define a function called `train_regressor` that takes the regression estimator, the feature matrix label vector cross-validation strategy and name of the regressor as inputs. It performs the cross-validation based training of the estimator using the negative mean absolute error as a scoring function. It uses cross-validation strategy as specified by `cv` parameter of the function argument. After the model is trained, we calculate the training error under `test_error` and print it out in these statements.

(Refer Slide Time: 02:28)

```
+ Code + Text
[5] 7     return_train_score=True,
8         return_estimator=True)
9
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.')

[6] 1 #title AdaBoostRegressor
2     train_regressor(
3         AdaBoostRegressor(), com_train_features,
4         com_train_labels, cv, 'AdaBoostRegressor')

On an average, AdaBoostRegressor makes an error of 73.363k +/- 6.031k on the training set.
On an average, AdaBoostRegressor makes an error of 73.623k +/- 6.057k on the test set.

[7] 1 #title GradientBoostingRegressor
2     train_regressor(
3         GradientBoostingRegressor(), com_train_features, com_train_labels,
4         cv, 'GradientBoostingRegressor')

On an average, GradientBoostingRegressor makes an error of 35.394k +/- 0.273k on the training set.
On an average, GradientBoostingRegressor makes an error of 36.773k +/- 0.723k on the test set.

XGBoost
[8] 1 !pip install xgboost
```

```
+ Code + Text
[5] 7 def __init__(base_estimator=None, *, n_estimators=50,
8     learning_rate=1.0, loss='linear', random_state=None)
9
10 cv
11 cv
12 An AdaBoost regressor.
13 pr An AdaBoost (T) regressor is a meta-estimator that begins by fitting a
14 pr regressor on the original dataset and then fits additional copies of the
15 pr regressor on the same dataset but where the weights of instances are
16 pr adjusted according to the error of the current prediction. As such,
17 pr subsequent regressors focus more on difficult cases.
18 pr This class implements the algorithm known as AdaBoost.R2 [2].
19 pr Read more in the User Guide <adaboost>.
20
21 1 #title AdaBoostRegressor
22 2 train_regressor(
23     AdaBoostRegressor(), com_train_features,
24     com_train_labels, cv, 'AdaBoostRegressor')
25
26 On an average, AdaBoostRegressor makes an error of 73.263k +/- 6.031k on the training set.
27 On an average, AdaBoostRegressor makes an error of 73.623k +/- 6.057k on the test set.
28
29 [7] 1 #title GradientBoostingRegressor
30 2 train_regressor(
31     GradientBoostingRegressor(), com_train_features, com_train_labels,
32     'GradientBoostingRegressor')
33
34 On an average, GradientBoostingRegressor makes an error of 35.394k +/- 0.273k on the training set.
35 On an average, GradientBoostingRegressor makes an error of 36.773k +/- 0.723k on the test set.
36
37 XGBoost
38 [8] 1 !pip install xgboost
```

```
+ Code + Text
[5] 7 return_train_score=True,
8     return_estimator=True)
9
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 '
15       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 t')
17
18 [6] 1 #title AdaBoostRegressor
19 2 train_regressor(
20     AdaBoostRegressor(), com_train_labels, cv, 'AdaBoostRegressor')
21
22 On an average, AdaBoostRegressor makes an error of 73.263k +/- 6.031k on the training set.
23 On an average, AdaBoostRegressor makes an error of 73.623k +/- 6.057k on the test set.
24
25 [7] 1 #title GradientBoostingRegressor
26 2 train_regressor(
27     GradientBoostingRegressor(),
28     'GradientBoostingRegressor')
29
30 On an average, GradientBoostingRegressor makes an error of 35.394k +/- 0.273k on the training set.
31 On an average, GradientBoostingRegressor makes an error of 36.773k +/- 0.723k on the test set.
32
33 1 ?GradientBoostingRegressor
34 2
35
36 XGBoost
37 completed at 11:52
```

Help X

Init signature:
GradientBoostingRegressor(*args, **kwargs)
Docstring:
Gradient Boosting for regression.

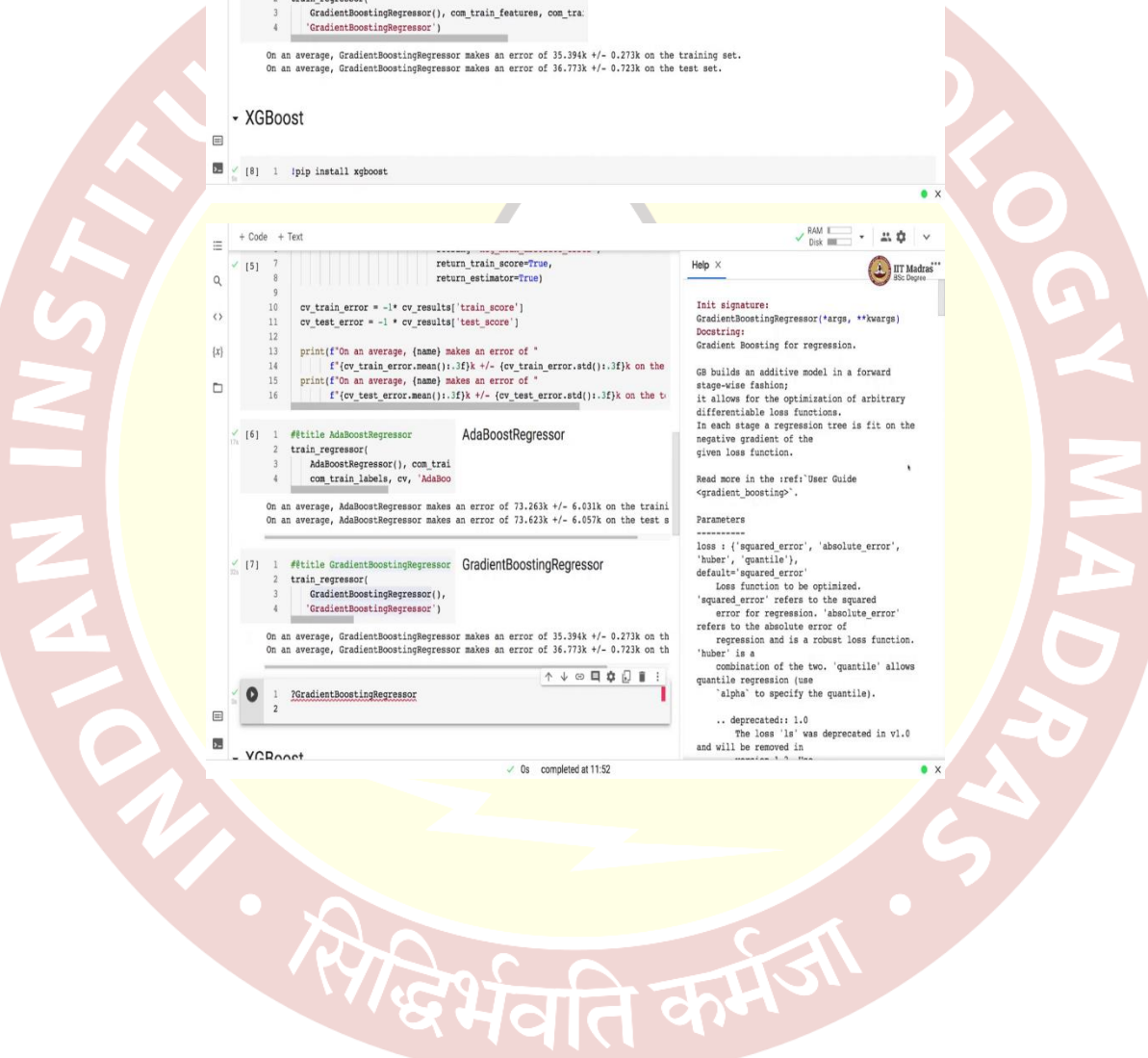
GB builds an additive model in a forward stage-wise fashion;
it allows for the optimization of arbitrary differentiable loss functions.
In each stage a regression tree is fit on the negative gradient of the given loss function.

Read more in the :ref:`User Guide <gradient_boosting>`.

Parameters

loss : {'squared_error', 'absolute_error', 'huber', 'quantile'}, default='squared_error'
Loss function to be optimized.
'squared_error' refers to the squared error for regression. 'absolute_error' refers to the absolute error of regression and is a robust loss function. 'huber' is a combination of the two. 'quantile' allows quantile regression (use 'alpha' to specify the quantile).

.. deprecated:: 1.0
The loss 'l1' was deprecated in v1.0 and will be removed in v1.1.




```

[5] 7         return_train_score=True,
      8         return_estimator=True)
      9
     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 t
     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 t

[6] 1 ##title AdaBoostRegressor      AdaBoostRegressor
     2 train_regressor(
     3     AdaBoostRegressor(), com_train_labels, cv, 'AdaBoo
     4     com_train_labels, cv, 'AdaBoo

On an average, AdaBoostRegressor makes an error of 73.263k +/- 6.031k on the traini
On an average, AdaBoostRegressor makes an error of 73.623k +/- 6.057k on the test s

[7] 1 ##title GradientBoostingRegressor      GradientBoostingRegressor
     2 train_regressor(
     3     GradientBoostingRegressor(),
     4     'GradientBoostingRegressor')

On an average, GradientBoostingRegressor makes an error of 35.394k +/- 0.273k on th
On an average, GradientBoostingRegressor makes an error of 36.773k +/- 0.723k on th

1 ?GradientBoostingRegressor
2
XGBoost

```

Help X

score by Friedman, "squared_error" for mean squared error, and "mae" for the mean absolute error. The default value of "friedman_mse" is generally the best as it can provide a better approximation in some cases.

.. deprecated:: 0.18

.. deprecated:: 0.24

"criterion='mae'" is deprecated and will be removed in version 1.1 (renaming of 0.26). The correct way of minimizing the absolute error is to use "loss='absolute_error'" instead.

.. deprecated:: 1.0

Criterion 'mse' was deprecated in v1.0 and will be removed in version 1.2. Use "criterion='squared_error'" which is equivalent.

min_samples_split : int or float, default=2

The minimum number of samples required to split an internal node:

- If int, then consider 'min_samples_split' as the minimum number.
- If float, then 'min_samples_split' is a fraction and 'ceil(min_samples_split * n_samples)' are the minimum number of samples for each split.

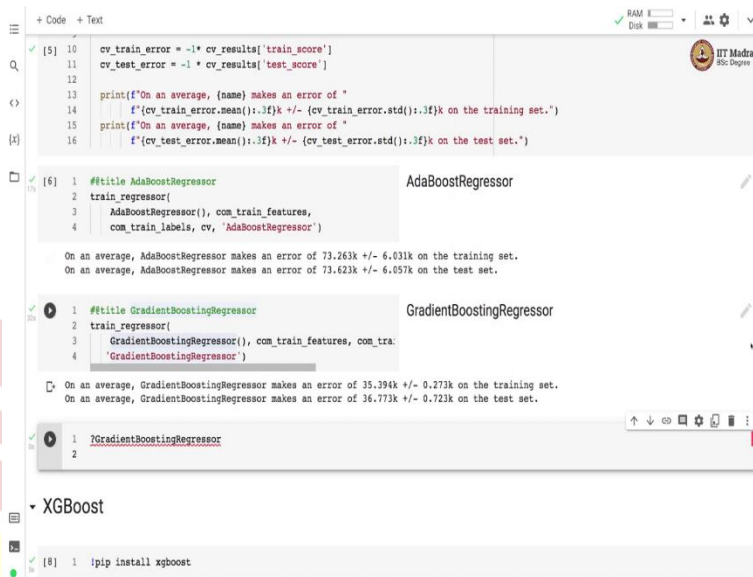
Let us run the AdaBoostRegressor and after running the AdaBoostRegressor we find that AdaBoostRegressor makes an error of 73.26k on the training set, and almost similar type of error on the test set the standard deviation is 6.03 for training and 6.05 for the test set. So, AdaBoostRegressor is defined using AdaBoostRegressor.

This is the instantiation of AdaBoostRegressor and we have made instantiation with default parameters and default parameters you can see here, number of estimators is = 50 and learning _rate is 1. And it uses linear loss. Next we define our GradientBoostingRegressor we instantiate GradientBoostingRegressor again with a bunch of default parameters.

And in case of GradientBoostingRegressor if you want to find out what are the default parameters, we can simply type the ? GradientBoostingRegressor and run this particular cell. And we can see the documentation for GradientBoostingRegressor. So, the default loss in case of GradientBoostingRegressor is squared error loss.

The default learning _rate is 0.1, and by default it trains 100 estimators. It uses friedman _mse as a criterion for making split in the decision tree. So, this is how you can access the documentation of the API that you are interested in.

(Refer Slide Time: 04:14)



```
[5]: 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.')

[6]: 1 #title AdaBoostRegressor
2 train_regressor(
3     AdaBoostRegressor(), com_train_features,
4     com_train_labels, cv, 'AdaBoostRegressor')

On an average, AdaBoostRegressor makes an error of 73.263k +/- 6.031k on the training set.
On an average, AdaBoostRegressor makes an error of 73.623k +/- 6.057k on the test set.

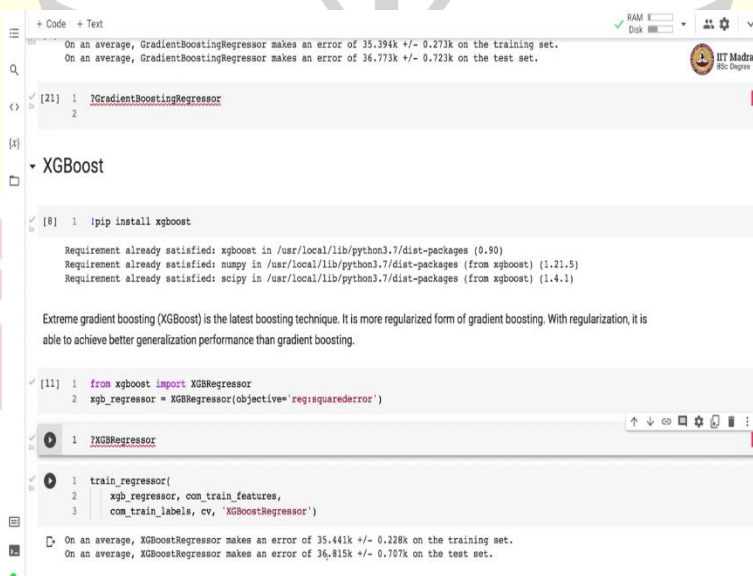
[7]: 1 #title GradientBoostingRegressor
2 train_regressor(
3     GradientBoostingRegressor(), com_train_features, com_train_labels, cv, 'GradientBoostingRegressor')

On an average, GradientBoostingRegressor makes an error of 35.394k +/- 0.273k on the training set.
On an average, GradientBoostingRegressor makes an error of 36.773k +/- 0.723k on the test set.

[8]: 1 ?GradientBoostingRegressor
2
```

So, you can see that after training the model with GradientBoostingRegressor it makes an error of 35.39k with the standard deviation of 0.273, whereas on the test set, it makes an error of 36.77 with standard deviation of 0.72. So, you can see that using gradient boosting has improved the performance of the regressor compared to the AdaBoostRegressor.

(Refer Slide Time: 04:42)



```
[9]: 1 ?GradientBoostingRegressor
2

[10]: 1 ?XGBRegressor

[11]: 1 from xgboost import XGBRegressor
2 xgb_regressor = XGBRegressor(objective='reg:squarederror')

[12]: 1 train_regressor(
2     xgb_regressor, com_train_features,
3     com_train_labels, cv, 'XGBRegressor')

On an average, XGBRegressor makes an error of 35.441k +/- 0.228k on the training set.
On an average, XGBRegressor makes an error of 36.815k +/- 0.707k on the test set.
```

And finally, we trained XGBoost regressor model. So, we installed xgboost regressor and xgboost is Extreme gradient boosting it is the latest boosting technique. It is more regularized for gradient

boosting it is very similar to gradient boosting except that it is more regularized form and with regularization, it is able to achieve better generalization performance than gradient boosting.

So, xgboost regressor is available in xgboost module. So, from xgboost we import XGBRegressor we instantiate XGBRegressor with objective function as squared error loss. And here you can again, check out the documentation of XGBRegressor. And when we train our model with xgboost regressor the performance is very similar to gradient boosting regressor.

So, remember that we have trained xgboost regressor with default parameters and with default parameters, it is giving performance very similar to gradient boosting. So, what we have not done is we have not tried to fine tune the parameters of these different regressors. So, another exercise that I am leaving you with is to basically perform hyper-parameter tuning of these regressors and see what kind of accuracy or what kind of errors you can get on the training set.

The idea is to reduce the error as much as possible, so that our regressor is able to make better predictions on the unseen data. So, in this video, we use boosting techniques for solving regression problem. We demonstrated the boosting regressor on California Housing dataset task. And we found that the gradient boosting and xgboost regressor in with the default parameters obtained very comparable performance, whereas AdaBoostRegressor was having worst of the performance among these three regressors.