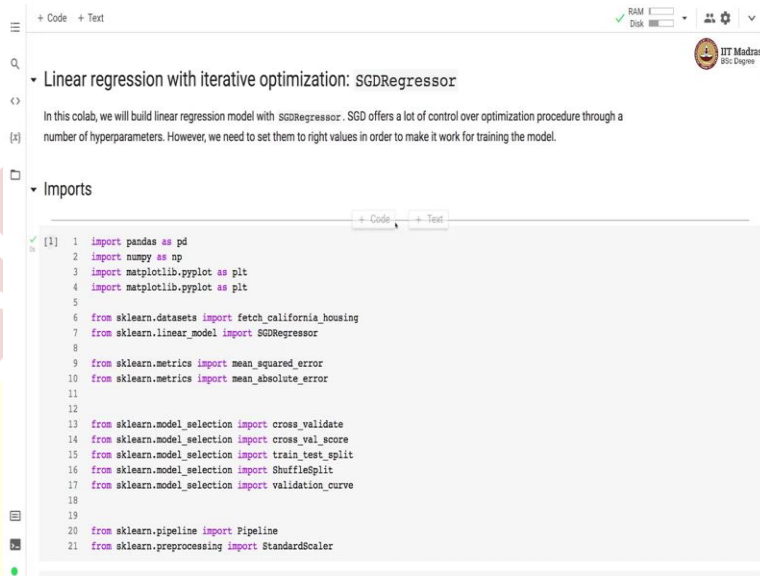


IIT Madras
ONLINE DEGREE

Machine Learning Program Indian Institute of Technology, Madras SGDRegressor Demonstration

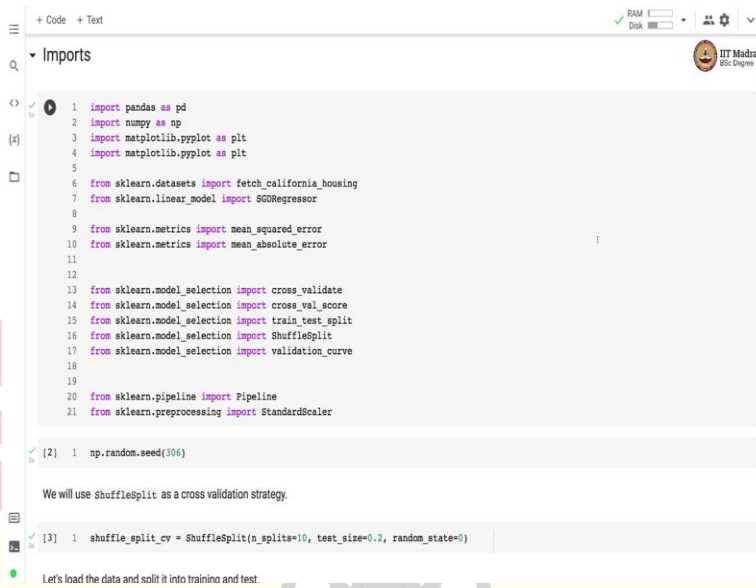
(Refer Slide Time: 0:05)



```
[1] 1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import matplotlib.pyplot as plt
5
6 from sklearn.datasets import fetch_california_housing
7 from sklearn.linear_model import SGDRegressor
8
9 from sklearn.metrics import mean_squared_error
10 from sklearn.metrics import mean_absolute_error
11
12
13 from sklearn.model_selection import cross_validate
14 from sklearn.model_selection import cross_val_score
15 from sklearn.model_selection import train_test_split
16 from sklearn.model_selection import ShuffleSplit
17 from sklearn.model_selection import validation_curve
18
19
20 from sklearn.pipeline import Pipeline
21 from sklearn.preprocessing import StandardScaler
```

Namaste welcome to the next video of the machine learning practice course. In this video, we will implement linear regression with SGDRegressor. SGDRegressor uses gradient descent or iterative optimization techniques for learning the parameters of the linear regression model. SGD offers a lot of control over optimization procedures through a number of hyperparameters. On the flip side, we need to set these hyperparameters to write values in order to make a SGDRegressor work for training the regression model. In this colab, we will study how to set these parameters right values.

(Refer Slide Time: 0:55)



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

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

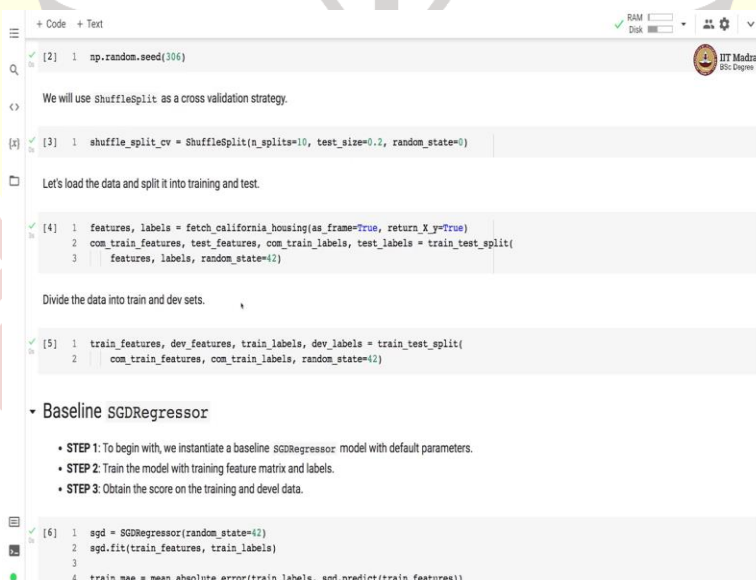
We will use ShuffleSplit as a cross validation strategy.

```
[3] 1 shuffle_split_cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=0)
```

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

We begin by importing the packages that are required for this demonstration. So, notably we are we are importing SGDRegressor from the sklearn.linear_model module. Then we have validation_curve for model selection and rest of the other packages are from the previous colab. Hence, we are not explaining them in detail here.

(Refer Slide Time: 1:32)



```
[4] 1 features, labels = fetch_california_housing(as_frame=True, return_X_y=True)
2 com_train_features, test_features, com_train_labels, test_labels = train_test_split(
3     features, labels, random_state=42)
```

Divide the data into train and dev sets.

```
[5] 1 train_features, dev_features, train_labels, dev_labels = train_test_split(
2     com_train_features, com_train_labels, random_state=42)
```

Baseline SGDRegressor

- STEP 1: To begin with, we instantiate a baseline SGDRegressor model with default parameters.
- STEP 2: Train the model with training feature matrix and labels.
- STEP 3: Obtain the score on the training and devel data.

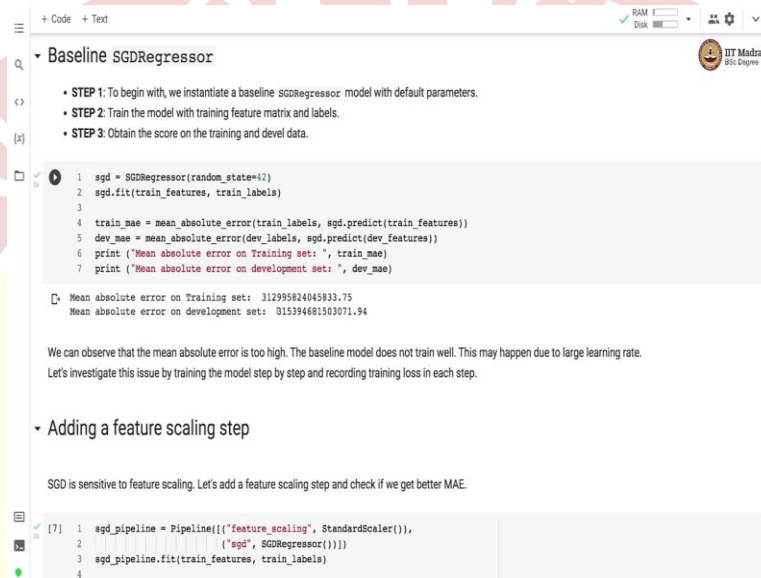
```
[6] 1 sgd = SGDRegressor(random_state=42)
2 sgd.fit(train_features, train_labels)
3
4 train_mae = mean_absolute_error(train_labels, sgd.predict(train_features))
```

As a usual practice, we set a random seed at the start of the notebook. Here again, we are using a shuffle split as a cross-validation strategy with 10 folds. And we will set aside 20 percent example set test examples. In each of the folds, we load the data, which is the California housing data set,

and split it into training and test. Further, what we do is divide the training data into two parts train and dev.

So, this particular dev part is what will be used for tuning the hyperparameters. So with this arrangement, we hold back the test data set for evaluating the performance on the SGDRegressor.

(Refer Slide Time: 2:30)



```
1 sgd = SGDRegressor(random_state=42)
2 sgd.fit(train_features, train_labels)
3
4 train_mae = mean_absolute_error(train_labels, sgd.predict(train_features))
5 dev_mae = mean_absolute_error(dev_labels, sgd.predict(dev_features))
6 print ('Mean absolute error on Training set: ', train_mae)
7 print ('Mean absolute error on development set: ', dev_mae)
```

Mean absolute error on Training set: 312995824045833.75
Mean absolute error on development set: 915394681503071.94

We can observe that the mean absolute error is too high. The baseline model does not train well. This may happen due to large learning rate.
Let's investigate this issue by training the model step by step and recording training loss in each step.

Adding a feature scaling step

SGD is sensitive to feature scaling. Let's add a feature scaling step and check if we get better MAE.

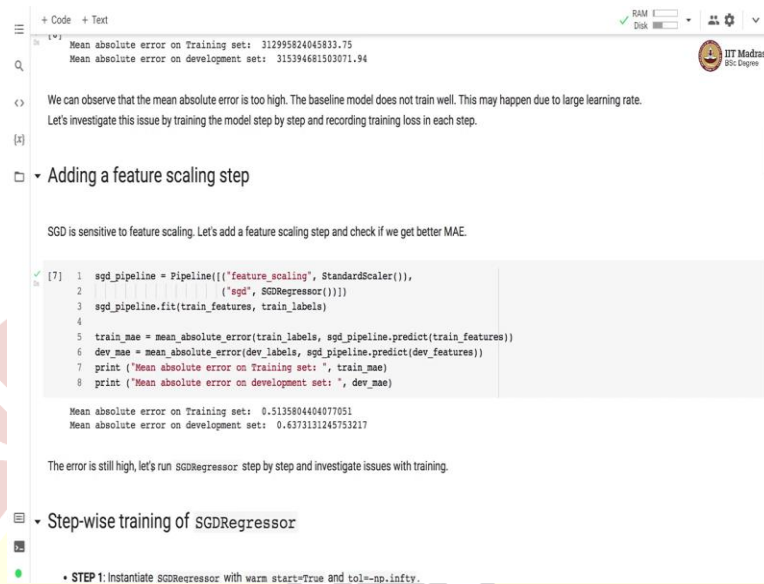
```
[7] 1 sgd_pipeline = Pipeline([['feature_scaling', StandardScaler()],
2                             ('sgd', SGDRegressor())])
3 sgd_pipeline.fit(train_features, train_labels)
4
```

So, now after loading the data and splitting it into three parts train dev and test, we will build the baseline SGDRegressor model. So to begin with, we instantiate the SGDRegressor with default parameters we only set the random state to 42. All other parameters are left to their default values. Then we train the SGDRegressor model with the features and labels from the training set. After fitting the model, we obtain the scores on training and development sets.

So here, we calculate the mean absolute error on the training set as well as on the development set. In order to calculate this error, we use the actual labels and predicted labels from the trained classifier. We repeat the same procedure on the development data set. And then we print this mean absolute error for both the datasets.

And you can observe that the mean absolute error is too high. The baseline model does not seem to train or does not seem to work that well. This may happen due to the large training rate. Let us investigate this issue by training the model step by step and recording training loss in each step.

(Refer Slide Time: 4:01)



```
+ Code + Text
Mean absolute error on Training set: 312995824045933.75
Mean absolute error on development set: 315394681503071.94

We can observe that the mean absolute error is too high. The baseline model does not train well. This may happen due to large learning rate.
Let's investigate this issue by training the model step by step and recording training loss in each step.

Adding a feature scaling step

SGD is sensitive to feature scaling. Let's add a feature scaling step and check if we get better MAE.

[7]: 1 sgd_pipeline = Pipeline([("feature_scaling", StandardScaler()),
2      ("sgd", SGDRegressor())])
3 sgd_pipeline.fit(train_features, train_labels)
4
5 train_mae = mean_absolute_error(train_labels, sgd_pipeline.predict(train_features))
6 dev_mae = mean_absolute_error(dev_labels, sgd_pipeline.predict(dev_features))
7 print ("Mean absolute error on Training set: ", train_mae)
8 print ("Mean absolute error on development set: ", dev_mae)

Mean absolute error on Training set: 0.5135804404077051
Mean absolute error on development set: 0.6373131245753217

The error is still high, let's run SGDRegressor step by step and investigate issues with training.

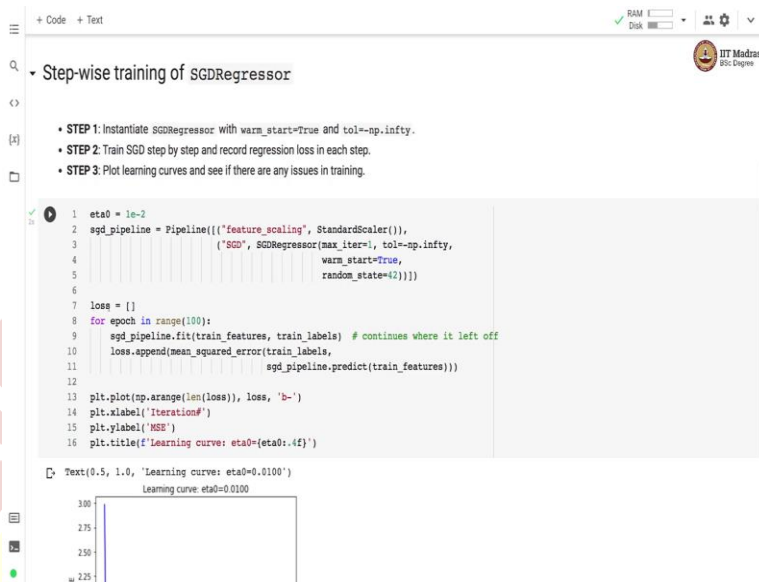
Step-wise training of SGDRegressor

STEP 1: Instantiate SGDRegressor with warm_start=True and tol=np.inf.
```

We studied that SGD is sensitive to feature scaling. So, what we will do is we will first add a feature scaling step and then check if we get a better mean absolute error. So, we add this feature scaling through the pipeline. So, now we have a pipeline that has got two stages. In the first stage, we perform the feature scaling with standard scalar and followed by SGDRegressor. Then now we call the fit on the pipeline object with the training features and labels.

Once the model is trained, we calculate the mean absolute error on the training and development set. So, now you can see that the error has reduced drastically. So, the error is now 0.51 And 0.63 on training and development sets respectively. However, the error is still high. And now what we will do is we will run the SGDRegressor step by step and investigate issues with training.

(Refer Slide Time: 5:09)



```
1 eta0 = 1e-2
2 sgd_pipeline = Pipeline([("feature_scaling", StandardScaler()),
3                           ("SGD", SGDRegressor(max_iter=1, tol=-np.inf,
4                                                warm_start=True,
5                                                random_state=42))])
6
7 loss = []
8 for epoch in range(100):
9     sgd_pipeline.fit(train_features, train_labels) # continues where it left off
10    loss.append(mean_squared_error(train_labels,
11                                  sgd_pipeline.predict(train_features)))
12
13 plt.plot(np.arange(len(loss)), loss, 'b-')
14 plt.xlabel('Iterations#')
15 plt.ylabel('MSE')
16 plt.title(f'Learning curve: eta0={eta0:.4f}')

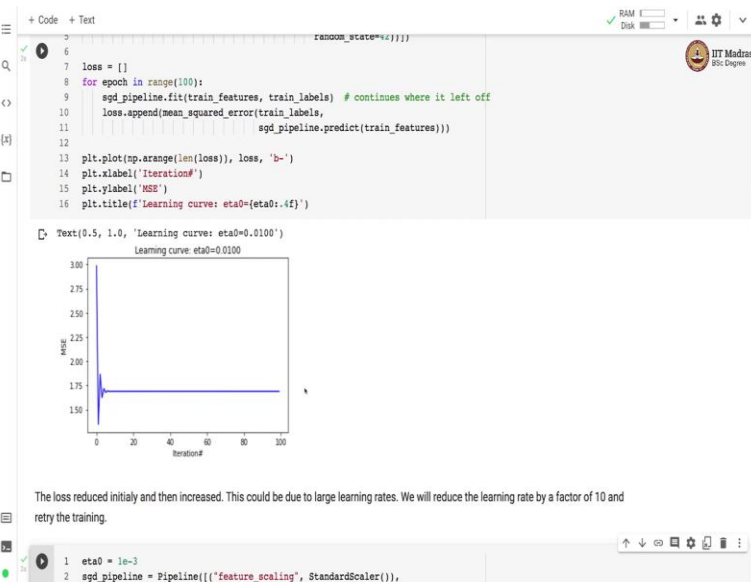
Text(0.5, 1.0, 'Learning curve: eta0=0.0100')
Learning curve: eta0=0.0100
```

So, we have discussed how to run SGDRegressor step by step in our slides. So, we instantiate SGDRegressor with warm start equal to true and tolerance equal to-infinity. So, again we are going to use a pipeline construct and the error rate is this particular learning rate which is eta 0 is set to instead 2.01 which is a default value. And we are actually instantiating this particular thing for visualization purposes.

We are not using this value inside SGDRegressor right now. So, hence SGDRegressor uses eta 0 which is the default value which also happens to be 0.01. Then what we do is, we perform the training in a loop. So, in every step, we call the fit on the pipeline object with the training feature matrix and labels. And since the warm start is equal to true what happens is it uses the parameter values that were learned in the last iteration.

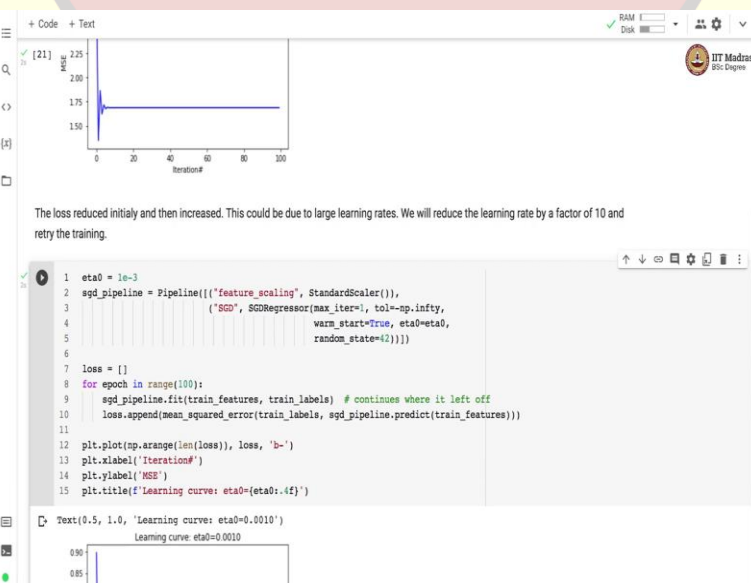
And then update those parameter values in the fit function. So, since we since now after this step we have fresh parameter values we calculate the fresh loss of 10 on this pipeline. So, we calculate the mean squared error here with the actual labels and the predicted labels. So, after we are done with 100 epochs or after running this particular process for 100 iterations we get loss values for each of the iteration.

(Refer Slide Time: 7:08)



We plot this loss value in what is called as a learning curve. And this learning curve is what we have used again and again in the machine learning techniques course. Then we have iterations on the x-axis and we have mean squared error on the y-axis. You can see that the mean squared error reduces initially, and then it goes up and settle somewhere over there. This could happen due to the large training rate. We will first reduce the training rate by a factor of 10 and retry the training.

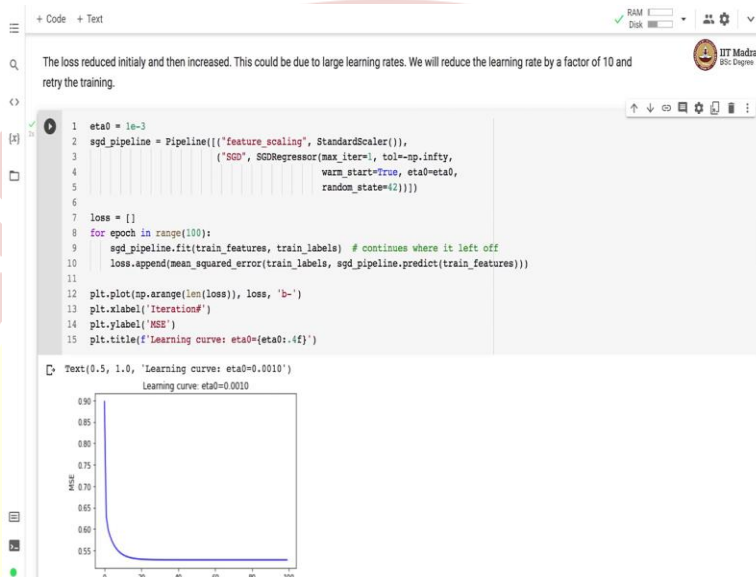
(Refer Slide Time: 7:42)



So, now, we are using eta 0 as 0.0001. And we are setting this eta 0 value in the SGDRegressor constructor. So, now everything from the previous step is exactly the same except that we are

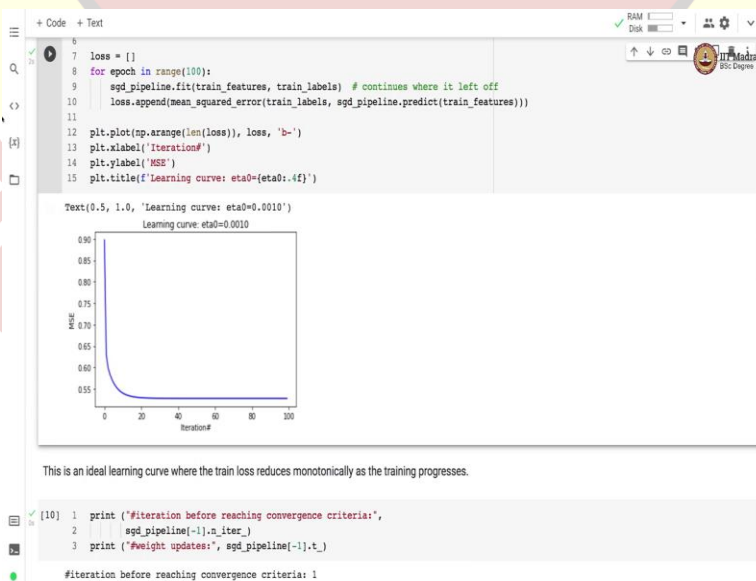
setting the value of eta 0 which is the initial learning rate. And just to remind you that the learning that will be used here learning style will be inverse scaling. And with inverse scaling, the learning rate is reduced in every step. So, here we are setting the initial learning rate.

(Refer Slide Time: 8:25)



We again repeat the same process where we train a SGDRegressor step by step and we collect the mean square error in every step and then plot it on the learning curve.

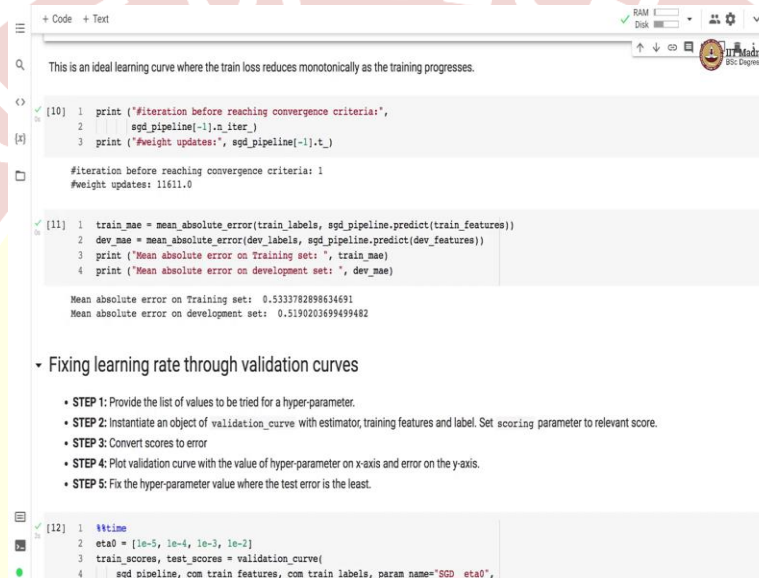
(Refer Slide Time: 8:39)



Now we have a learning curve which is looking something like this. So, what we can see here is as we do more and more iteration the training loss goes down gradually. And this is kind of a good

sign and this is an ideal learning curve where training loss reduces monotonically as the training progresses. Sure this loss is not reaching the 0, the 0 value which is the ideal value for the loss which is being squared error. And this could be due to underfitting, but that is a different issue. Here we seem to have found a learning rate that helps us to reduce the mean square error monotonically as the iterations increase.

(Refer Slide Time: 9:31)



```
+ Code + Text
This is an ideal learning curve where the train loss reduces monotonically as the training progresses.

[10] 1 print ('#iteration before reaching convergence criteria:',
2         sgdpipeline[-1].n_iter_)
3 print ('#weight updates:', sgdpipeline[-1].t_)

#iteration before reaching convergence criteria: 1
#weight updates: 11611.0

[11] 1 train_mae = mean_absolute_error(train_labels, sgdpipeline.predict(train_features))
2 dev_mae = mean_absolute_error(dev_labels, sgdpipeline.predict(dev_features))
3 print ('Mean absolute error on training set: ', train_mae)
4 print ('Mean absolute error on development set: ', dev_mae)

Mean absolute error on Training set: 0.5333782898634691
Mean absolute error on development set: 0.5190203699499482

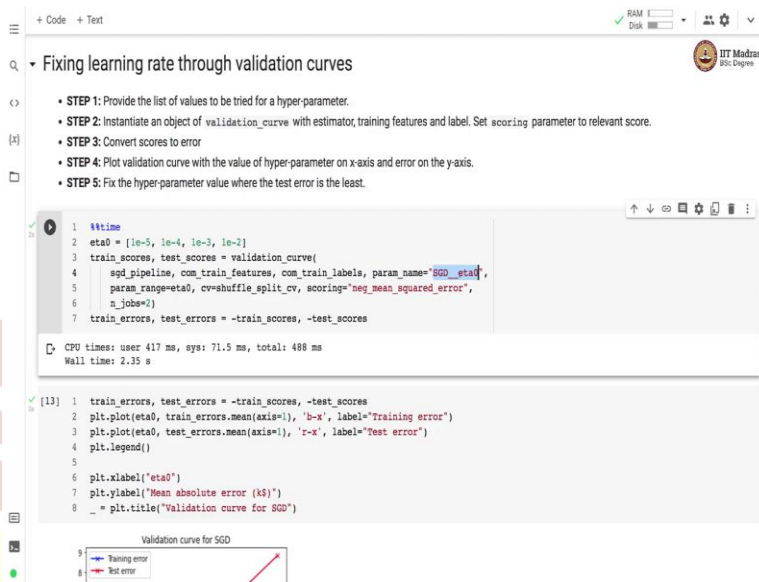
- Fixing learning rate through validation curves

• STEP 1: Provide the list of values to be tried for a hyper-parameter.
• STEP 2: Instantiate an object of validation_curve with estimator, training features and label. Set scoring parameter to relevant score.
• STEP 3: Convert scores to error
• STEP 4: Plot validation curve with the value of hyper-parameter on x-axis and error on the y-axis.
• STEP 5: Fix the hyper-parameter value where the test error is the least.

[12] 1 %%time
2 eta0 = [1e-5, 1e-4, 1e-3, 1e-2]
3 train_scores, test_scores = validation_curve(
4     sgdpipeline, com_train_features, com_train_labels, param_name='SGD_eta0',
```

So, we can also find out how many iterations SGD took before reaching the convergence criteria and how many weight updates were performed. So here, the SGD pipeline has taken one iteration and it has performed 11,611 weight updates. You can see that the mean absolute error on the training set is 0.53 whereas on the development set it is 0.51.

(Refer Slide Time: 10:19)



```
+ Code + Text
Fixing learning rate through validation curves

• STEP 1: Provide the list of values to be tried for a hyper-parameter.
• STEP 2: Instantiate an object of validation_curve with estimator, training features and label. Set scoring parameter to relevant score.
• STEP 3: Convert scores to error
• STEP 4: Plot validation curve with the value of hyper-parameter on x-axis and error on the y-axis.
• STEP 5: Fix the hyper-parameter value where the test error is the least.

1 #time
2 eta0 = [1e-5, 1e-4, 1e-3, 1e-2]
3 train_scores, test_scores = validation_curve(
4     sgd_pipeline, com_train_features, com_train_labels, param_name="SGD_eta0",
5     param_range=eta0, cv=shuffle_split_cv, scoring="neg_mean_squared_error",
6     n_jobs=2)
7 train_errors, test_errors = -train_scores, -test_scores

CPU times: user 417 ms, sys: 71.5 ms, total: 488 ms
Wall time: 2.35 s

[13]: 1 train_errors, test_errors = -train_scores, -test_scores
2     plt.plot(eta0, train_errors.mean(axis=1), 'b-x', label="Training error")
3     plt.plot(eta0, test_errors.mean(axis=1), 'r-x', label="Test error")
4     plt.legend()
5
6     plt.xlabel('eta0')
7     plt.ylabel('Mean absolute error (k$)')
8     plt.title("Validation curve for SGD")

Validation curve for SGD
Training error
Test error
```

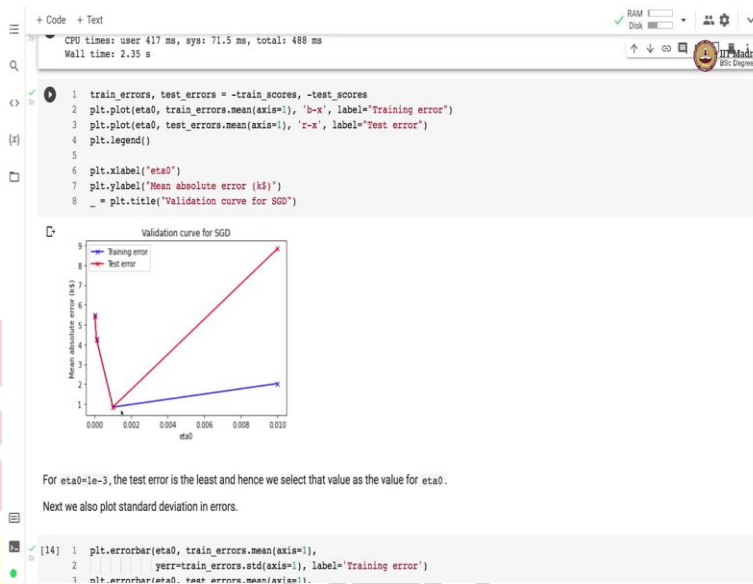
So, there is an alternate way to fix the learning rate that is to validate curves, let us study how to do that. Here, we have to provide the list of values of hyperparameter to be tried here the hyperparameter is the initial learning rate which is eta 0 and we are specifying the values that we want to try eta 0 in ascending order then we instantiate an object of validation curve with the estimator the training feature matrix training label matrix.

And we are setting the parameter for which you want to calculate this validation curve. And since we are using a pipeline construct the parameter name is the name of the step double_ the name of the parameter. So, we can see that in our pipeline object SGD step was denoted by identifier SGD capitalized, and hence, we refer to this parameter as capital SGD double_eta 0.

So, this is the parameter on which we want to calculate the validation curve. We want to vary this parameter and for each parameter value, we want to calculate the training loss and test loss. And we can also specify what kind of scoring mechanism we want to use. So, here we have specified negative mean squared error as a scoring mechanism, you also specify the cross-validation that we want to use and we are going to use shuffle split cross-validation.

We also specify the parameter range to be tried and this parameter range is what we have specified initially. So, this list also goes as an input to the validation curve. So, where the addition curve returns training score and test scores so we convert these scores into error by multiplying them by -1.

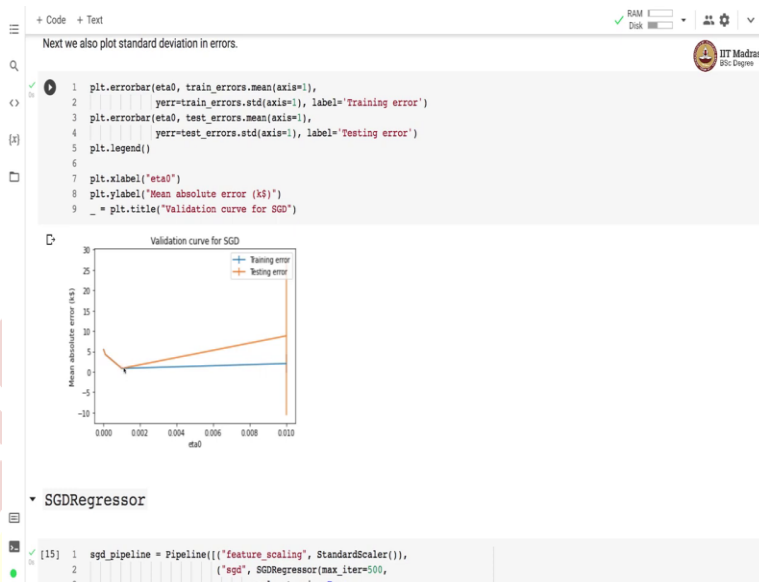
(Refer Slide Time: 12:26)



And we plot these errors on this particular plot whose x-axis is the value of the η_0 whose x-axis has different parameter values that we tried and η_0 was the parameter of our interest and we tried 5 different 4 different values of that so this is the first value, second value, third value, and fourth value and for each of those parameter values, we got training error and test error which are plotted over here.

And on the y-axis, you have the mean absolute error you can see that the training and test error reduced initially as we increase η_0 but beyond the point, it just shot up. So, this is the point where the test error is the least and hence we select this particular value of η_0 as the value that we want to use as the initial learning rate. We can also plot the standard deviation in the error.

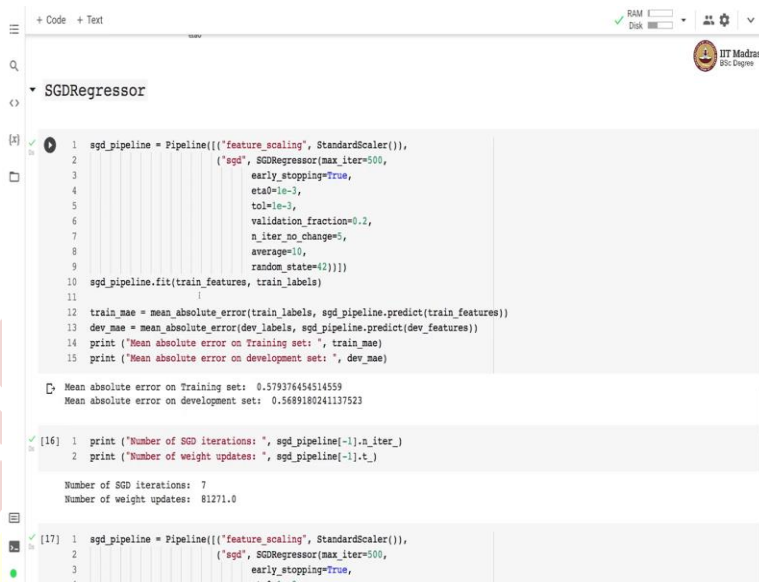
(Refer Slide Time: 13:43)



And this is slightly complicated because you can see that at the end the standard deviation is too high. And that results into some kind of unstable estimation for the test error. So, as we select this eta 0 point where the standard deviation is also not that high. And we have the lowest training and test error without much variation. So, this is how you can use the validation curve to set any hyperparameter to decide the right value for the hyperparameter of your interest.

So again, it is a very simple process, we have to provide the list of values to be tried. And then we simply instantiate the validation curve object and gather the training and test errors and plot them in on the validation plot. And then fix the value of the hyperparameter where the test error is the lowest. So, we can use this process in order to set any other hyperparameters.

(Refer Slide Time: 14:59)



```
1 sgd_pipeline = Pipeline([("feature_scaling", StandardScaler()),
2                           ("sgd", SGDRegressor(max_iter=500,
3                                                 early_stopping=True,
4                                                 eta=1e-3,
5                                                 tol=1e-3,
6                                                 validation_fraction=0.2,
7                                                 n_iter_no_change=5,
8                                                 average=10,
9                                                 random_state=2))])
10 sgd_pipeline.fit(train_features, train_labels)
11
12 train_mae = mean_absolute_error(train_labels, sgd_pipeline.predict(train_features))
13 dev_mae = mean_absolute_error(dev_labels, sgd_pipeline.predict(dev_features))
14 print('Mean absolute error on Training set: ', train_mae)
15 print('Mean absolute error on development set: ', dev_mae)

Mean absolute error on Training set: 0.579376454514559
Mean absolute error on development set: 0.5689180241137523

[16]: 1 print('Number of SGD iterations: ', sgd_pipeline[-1].n_iter_)
      2 print('Number of weight updates: ', sgd_pipeline[-1].t_)

Number of SGD iterations: 7
Number of weight updates: 81271.0

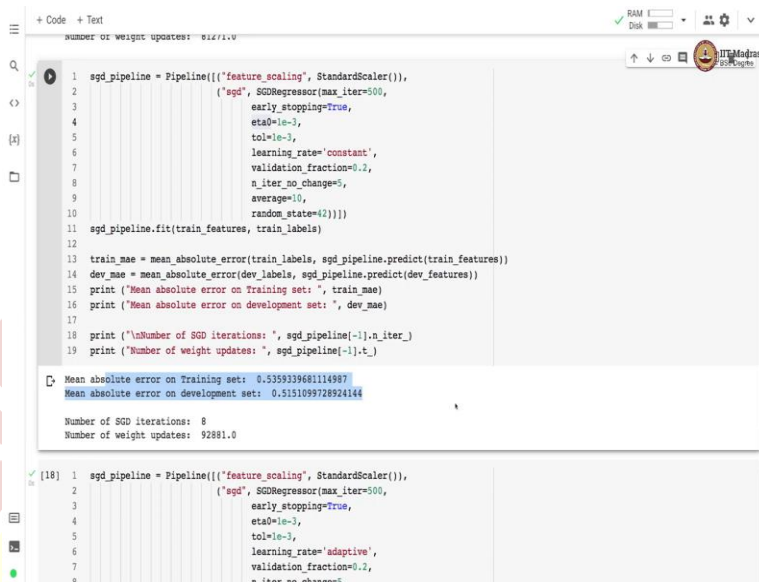
[17]: 1 sgd_pipeline = Pipeline([("feature_scaling", StandardScaler()),
      2                           ("sgd", SGDRegressor(max_iter=500,
      3                                                 early_stopping=True,
```

Not only in case SGDRegressor but any other hyperparameters that we may encounter while training different machine learning algorithms. Now that we have fixed the value of the initial learning rate let us train the SGDRegressor. So, now we are going to train SGDRegressor with maximum iteration to be 500 early stoppings equal to true. So, here we are going to stop SGDRegressor you know with early stopping criteria.

And in order to set early stopping criteria, we also need to provide the validation fraction. So, we are setting aside 20 percent examples as validation examples. Then n_iter_no_change which is the number of iteration for which we do not see much change you know we do not see change greater than the value of tolerance that is set to 5 and we also have set the tolerance which is again 0.0001.

So, this is the machinery for early stopping or these parameters are kind of set for early stopping. Then we also set an average equal to 10 in order to use averaged SGD. So, this will essentially start averaging SGD after it sees 10 samples. Now, with this setup we retrain the much we retrained the regression model and calculate the mean absolute error on training and development sets.

(Refer Slide Time: 16:43)



```
+ Code + Text
Number of weight updates: 92881.0

1 sgd_pipeline = Pipeline([('feature_scaling', StandardScaler()),
2                           ('sgd', SGDRegressor(max_iter=500,
3                                                  early_stopping=True,
4                                                  eta0=1e-3,
5                                                  tol=1e-3,
6                                                  learning_rate='constant',
7                                                  validation_fraction=0.2,
8                                                  n_iter_no_change=5,
9                                                  average=10,
10                                                 random_state=42))])
11 sgd_pipeline.fit(train_features, train_labels)
12
13 train_mae = mean_absolute_error(train_labels, sgd_pipeline.predict(train_features))
14 dev_mae = mean_absolute_error(dev_labels, sgd_pipeline.predict(dev_features))
15 print('Mean absolute error on Training set: ', train_mae)
16 print('Mean absolute error on development set: ', dev_mae)
17
18 print('\nNumber of SGD iterations: ', sgd_pipeline[-1].n_iter_)
19 print('Number of weight updates: ', sgd_pipeline[-1].t_)

Mean absolute error on Training set: 0.5359339481114987
Mean absolute error on development set: 0.51099728924144

Number of SGD iterations: 8
Number of weight updates: 92881.0

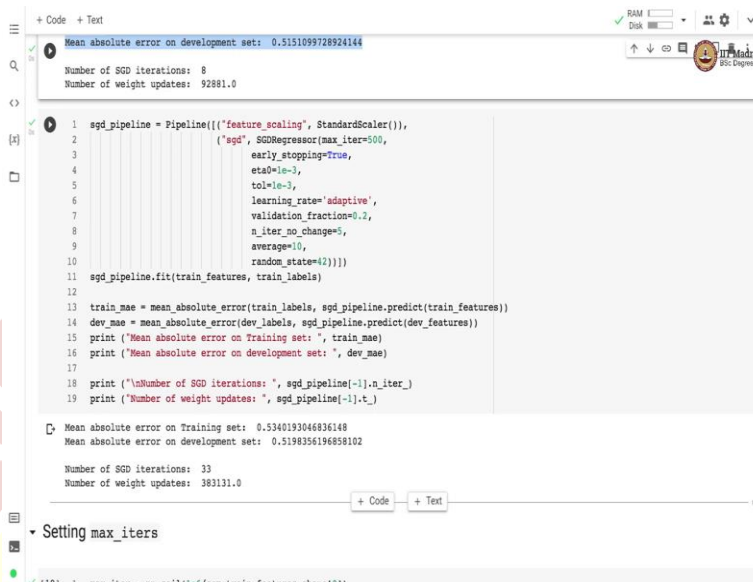
[18]: 1 sgd_pipeline = Pipeline([('feature_scaling', StandardScaler()),
2                                ('sgd', SGDRegressor(max_iter=500,
3                                                       early_stopping=True,
4                                                       eta0=1e-3,
5                                                       tol=1e-3,
6                                                       learning_rate='adaptive',
7                                                       validation_fraction=0.2,
8                                                       n_iter_no_change=5,
9                                                       random_state=42))])
sgd_pipeline.fit(train_features, train_labels)
```

So, we got a mean absolute error of 0.57 and 0.56 on the training and development sets respectively. SGD took several iterations and it performed 81271 weight updates. So, the number of weight updates is equal to iterations into the number of examples in the training set. And that will come out if you do the remarks it will come out to the number of weight updates. So, by default SGD uses inverse scaling as learning rate.

Now, what we will do is we will change the learning rate and here we have changed the learning rate to constant. And when we use a constant learning rate the learning rate specified in eta 0 is used across all the iterations. So, you can see that when we use constant learning rate it took us about 8 iterations and 92,881 weight updates. We get slightly better performance on this and you can observe by finding the reduced absolute error in comparison to the earlier model.

Here the error was 0.57 and 0.56 on training and development sets respectively that is reduced to 0.53 and 0.51.

(Refer Slide Time: 18:23)



```
+ Code + Text
Mean absolute error on development set: 0.5151099728924144
Number of SGD iterations: 8
Number of weight updates: 92881.0

1 sgd_pipeline = Pipeline([('feature_scaling', StandardScaler()),
2                           ('sgd', SGDRegressor(max_iter=500,
3                                                 early_stopping=True,
4                                                 eta0=1e-1,
5                                                 tol=1e-3,
6                                                 learning_rate='adaptive',
7                                                 validation_fraction=0.2,
8                                                 n_iter_no_change=5,
9                                                 average=10,
10                                                random_state=2))])
11 sgd_pipeline.fit(train_features, train_labels)
12
13 train_mae = mean_absolute_error(train_labels, sgd_pipeline.predict(train_features))
14 dev_mae = mean_absolute_error(dev_labels, sgd_pipeline.predict(dev_features))
15 print ('Mean absolute error on Training set: ', train_mae)
16 print ('Mean absolute error on development set: ', dev_mae)
17
18 print ('\nNumber of SGD iterations: ', sgd_pipeline[-1].n_iter_)
19 print ('Number of weight updates: ', sgd_pipeline[-1].t_)

Mean absolute error on Training set: 0.5340193046836148
Mean absolute error on development set: 0.5198356196858102

Number of SGD iterations: 33
Number of weight updates: 383131.0

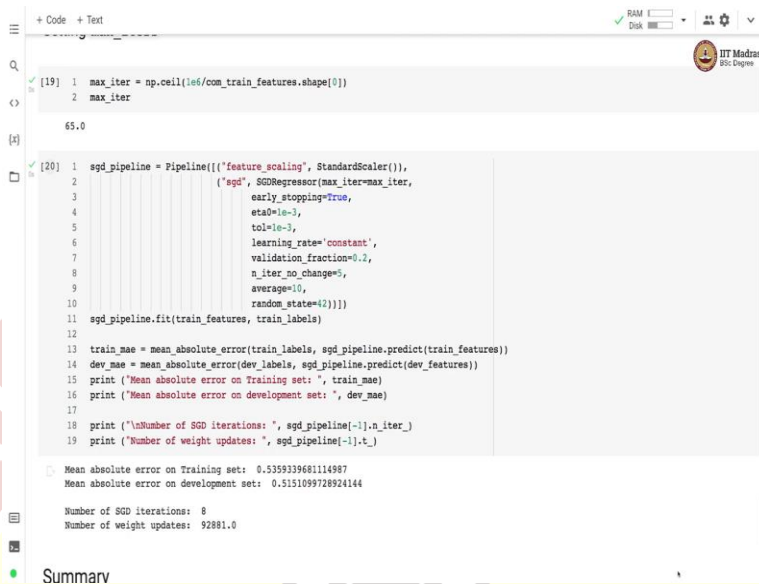
+ Code + Text

Setting max_iters
```

There is one more learning rate which is the adaptive learning rate. In adaptive learning what happens is it uses the initial learning rate as long as we reach the convergence criteria when the convergence criteria are reached the learning rate is reduced by dividing the learning rate by 5 and training continues as long as the convergence criteria are met again. And again at that, the training rate will be reduced further and this will continue unless the training rate drops below 10^{-6} .

So, you can see that here we obtained the mean absolute error on training to be 0.53 and on the development set, it is 0.51. It took 33 iterations and it performed 383,131 weight updates. So, this adaptive learning rate has taken longer iterations for SGD to converge because of the style of learning. It reduces the learning rate once it reaches the convergence criteria and is continuous unless the learning rate drops below 10^{-6} .

(Refer Slide Time: 19:45)



```
[19] 1 max_iter = np.ceil(1e6/com_train_features.shape[0])
      2 max_iter
      65.0

[20] 1 sgd_pipeline = Pipeline([("feature_scaling", StandardScaler()),
      2                               ("sgd", SGDRegressor(max_iter=max_iter,
      3                               early_stopping=True,
      4                               eta0=1e-3,
      5                               tol=1e-3,
      6                               learning_rate='constant',
      7                               validation_fraction=0.2,
      8                               n_iter_no_change=5,
      9                               average=10,
      10                              random_state=42))])
      11 sgd_pipeline.fit(train_features, train_labels)
      12
      13 train_mae = mean_absolute_error(train_labels, sgd_pipeline.predict(train_features))
      14 dev_mae = mean_absolute_error(dev_labels, sgd_pipeline.predict(dev_features))
      15 print('Mean absolute error on Training set: ', train_mae)
      16 print('Mean absolute error on development set: ', dev_mae)
      17
      18 print('\nNumber of SGD iterations: ', sgd_pipeline[-1].n_iter_)
      19 print('Number of weight updates: ', sgd_pipeline[-1].k_)
```

Mean absolute error on Training set: 0.5359339681114987
Mean absolute error on development set: 0.5151099728924144

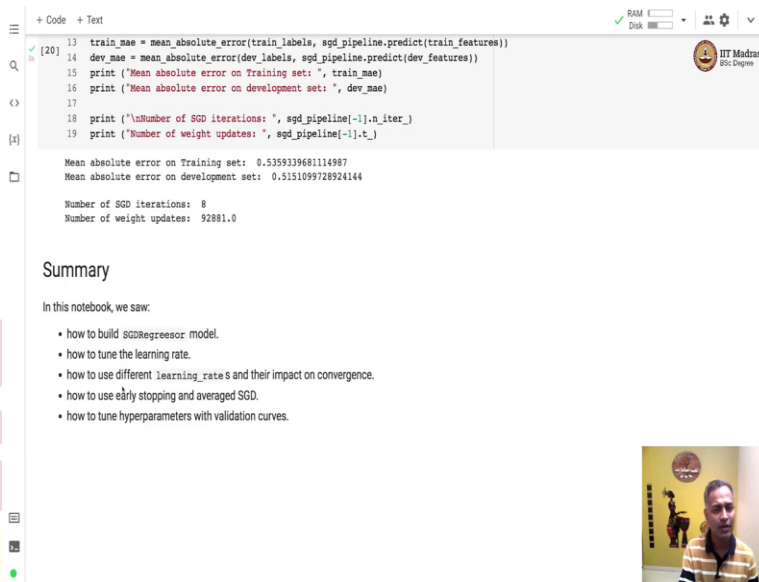
Number of SGD iterations: 8
Number of weight updates: 92881.0

Summary

So, we have seen in the slide is that empirically SGD converges after seeing about 1 million examples. So here, we set max iter to ceiling of 1 million divided by the total number of training examples in the training set. And in our case, it comes out to be 65 the maximum number of iteration comes down comes to 65. We said this max iter as a parameter in SGDRegressor constructor.

And we trained the SGDRegressor model with the training feature matrix and labels. So, even though we have said this max iter to 65 our SGD converges in 8 iterations and it has performed 92,881 updates in these 8 iterations. We obtain a mean absolute error of 0.53 on the training set and 0.51 on the development set.

(Refer Slide Time: 20:52)



The screenshot shows a Jupyter Notebook interface. The top part displays a code cell with the following Python code:

```
13 train_mae = mean_absolute_error(train_labels, sgd_pipeline.predict(train_features))
14 dev_mae = mean_absolute_error(dev_labels, sgd_pipeline.predict(dev_features))
15 print('Mean absolute error on Training set: ', train_mae)
16 print('Mean absolute error on development set: ', dev_mae)
17
18 print('\nNumber of SGD iterations: ', sgd_pipeline[-1].n_iter_)
19 print('Number of weight updates: ', sgd_pipeline[-1].t_)
```

Below the code, the output is shown:

```
Mean absolute error on Training set: 0.5359339681114987
Mean absolute error on development set: 0.5151099728924144

Number of SGD iterations: 8
Number of weight updates: 92881.0
```

The bottom part of the notebook shows a 'Summary' section with the text: 'In this notebook, we saw:' followed by a bulleted list:

- how to build `SGDRegressor` model.
- how to tune the learning rate.
- how to use different `learning_rates` and their impact on convergence.
- how to use early stopping and averaged SGD.
- how to tune hyperparameters with validation curves.

A small video thumbnail of a person is visible in the bottom right corner of the notebook interface.

So that brings us to the end of this particular demonstration. In this demonstration, we studied how to build the `SGDRegressor` model, how to tune the learning rate, how to use different learning rates and that impact on convergence, how to use early stopping and average SGD. We also learned how to tune hyperparameters with validation curves. We also tried different learnings (run) learning rates and found out the impact on the training different learning rates took a different number of steps.