Introduction to Machine Learning
Prof. Mr. Anirban Santara
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

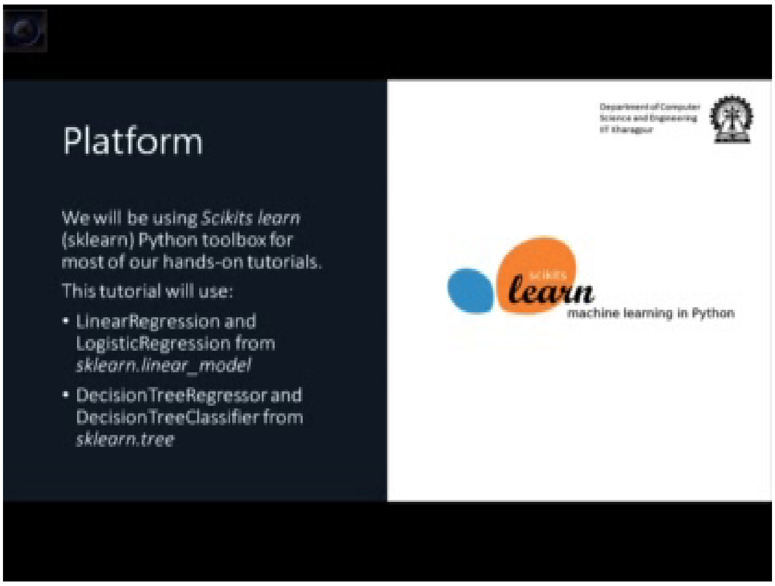
Lecture – 09
Python Exercise on Decision Tree and Linear Regression

Hello everyone, welcome to the first tutorial class of this course. I am Anirban Santara I am a PhD student in the Department of Computer Science and Engineering of IIT, Kharagpur and a TA for this course.

So, these tutorial classes I am going to take you through step by step hands on problem solving in the area of Machine Learning. In the first tutorial class, we are going to concentrate on Linear regression, Logistic regression and Decision trees, and we will use decision trees for regression and classification and these are going to be the first steps that will give you picture or give you idea of how problem solving really occurs in real life machine learning applications.

The platform which we are going to use is python based tool box called Scikits learn.

(Refer Slide Time: 01:18)



Platform

We will be using *Scikits learn* (sklearn) Python toolbox for most of our hands-on tutorials.

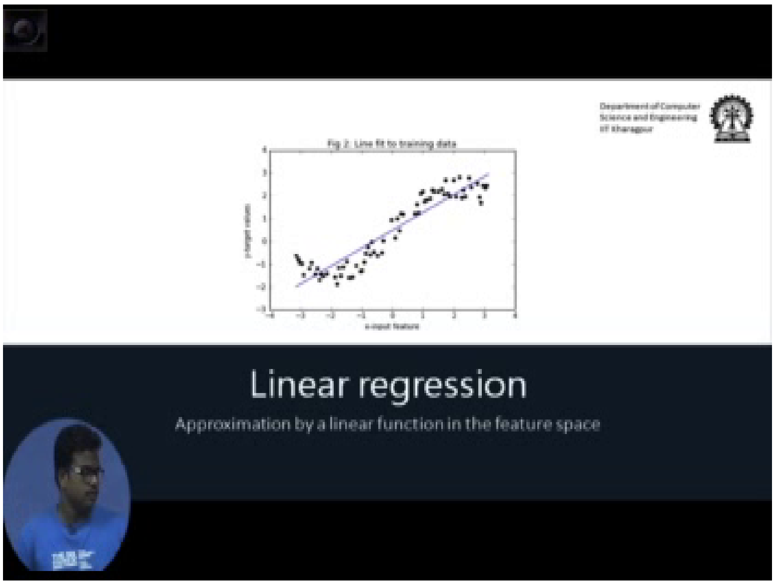
This tutorial will use:

- LinearRegression and LogisticRegression from *sklearn.linear_model*
- DecisionTreeRegressor and DecisionTreeClassifier from *sklearn.tree*

Department of Computer Science and Engineering IIT Kharagpur
scikits
learn
machine learning in Python

This particular tool box is very popular in the industry as well as in the academia and has implementations of huge number of machine learning algorithms, and we are going to use this particular library in most parts of our course. So, in this particular tutorial we are going to use two things from the linear model part of the Scikit learn tool box. The first thing will be linear regression, second will be logistic regression. And then, from the tree module of this particular tool box we are going to use decision tree regressor and decision tree classifier.

(Refer Slide Time: 01:53)

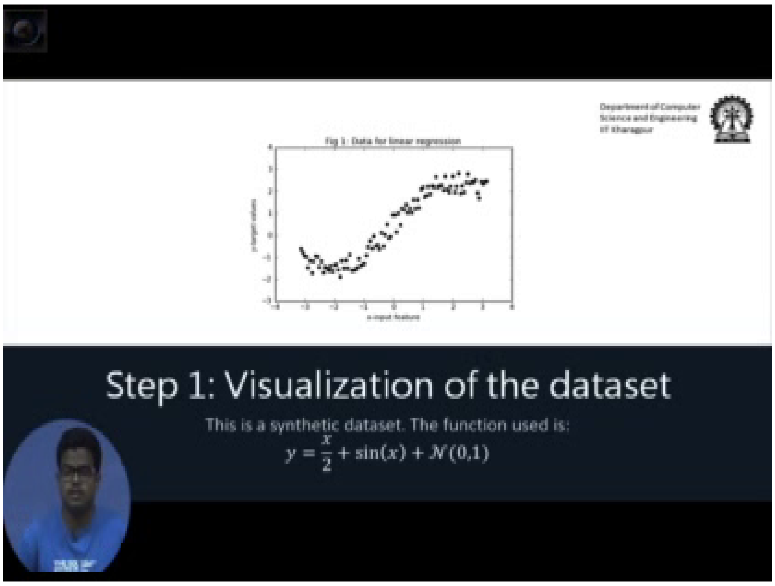


Linear regression

Approximation by a linear function in the feature space

Let us come to Linear regression first. Linear regression is all about approximating linear function on your data. So, given a data you are going to fit model and this model is limited by the family of linear functions of the input features.

(Refer Slide Time: 02:15)



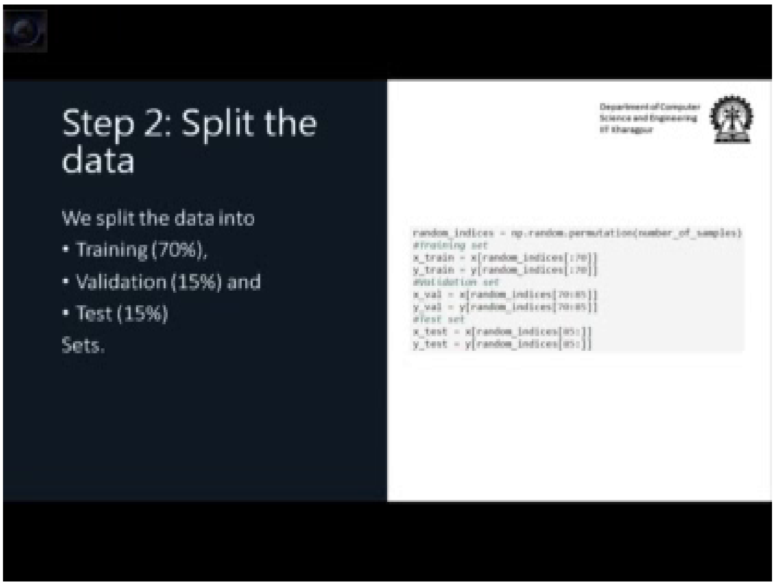
Step 1: Visualization of the dataset

This is a synthetic dataset. The function used is:

$$y = \frac{x}{2} + \sin(x) + \mathcal{N}(0,1)$$

The first thing which we will do in this tutorial is which create a synthetic dataset. So, why do we use this synthetic dataset? Just to keep things simple. So, our synthetic dataset has been made using the function which you can see on the slide. So, it is a very easy function. And the third part of the function is actually Gaussian noise which gives some randomness to the data and a feel of what real life data sets actually look like. The first step in any machine learning problem solving is to visualize the dataset and this actually gives nice picture of how complex the problem actually is and like how to exactly go about solving the problem. So, visualization is important.

(Refer Slide Time: 03:03)



Step 2: Split the data

We split the data into
- Training (70%),
- Validation (15%) and
- Test (15%)

Sets.

Department of Computer Science and Engineering IIT Kharagpur

```
random_indices = np.random.permutation(number_of_samples)
#Training set
x_train = x[random_indices[:70]]
y_train = y[random_indices[:70]]
#Validation set
x_val = x[random_indices[70:85]]
y_val = y[random_indices[70:85]]
#Test set
x_test = x[random_indices[85:]]
y_test = y[random_indices[85:]]
```

The second step in any machine learning application is to split the dataset which is available to us into 3 fragments - the Training data set, the Validation set and the Test set.

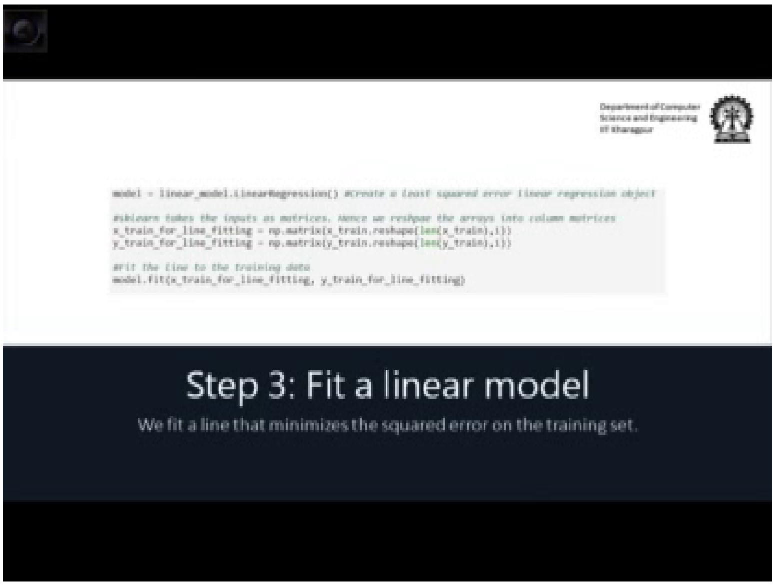
So, the protocol is to train the model on the training set and after every episode of training in order to evaluate how good the algorithm would actually generalize in the real world we are going to evaluate the model on the validation set, and after every episode of learning we go about doing one phase of evaluation on the validation set and we come back and tune the hyper parameters of the system. Like, we can change the kind of features we are using, we may like to change certain model architectures and say number of nodes in odd number of trees in an ensemble.

So, all number of nodes in a neural network and these things are the hyper parameters which are not optimized directly by the learning algorithm, but those hyper parameters have to be observed based on the models performance on the validation data or how good it actually generalizes to real world. So, that is why we need the validation set.

And finally, after all the training has been done, all the tuning of the architecture has been done, we take the test set and we find the performance of the model on the test. And this particular performance which we get, it is actually supposed to be reported as a measure of actual generalization of the model because this particular test set was not show to the model at any phase of its training - neither during its parameter learning nor during its hyper parameter tuning. So, this actually gives a feel of what set of unseen examples or unseen cases may look like.

So, that is why we need to this kind of a split of the data that is available to us into training, validation and test sets. And, it is general practice to have 70 percent of the data aside as training set, and the rest 30 percent is divided into validation and test sets. So, that is what we do here.

(Refer Slide Time: 05:26)



Department of Computer
Science and Engineering
IIT Kharagpur

```
model = linear_model.LinearRegression() #Create a least squared error linear regression object

#sklearn takes the inputs as matrices. Hence we reshape the arrays into column matrices
x_train_for_line_fitting = np.matrix(x_train.reshape(len(x_train),1))
y_train_for_line_fitting = np.matrix(y_train.reshape(len(y_train),1))

#Fit the line to the training data
model.fit(x_train_for_line_fitting, y_train_for_line_fitting)
```

Step 3: Fit a linear model

We fit a line that minimizes the squared error on the training set.

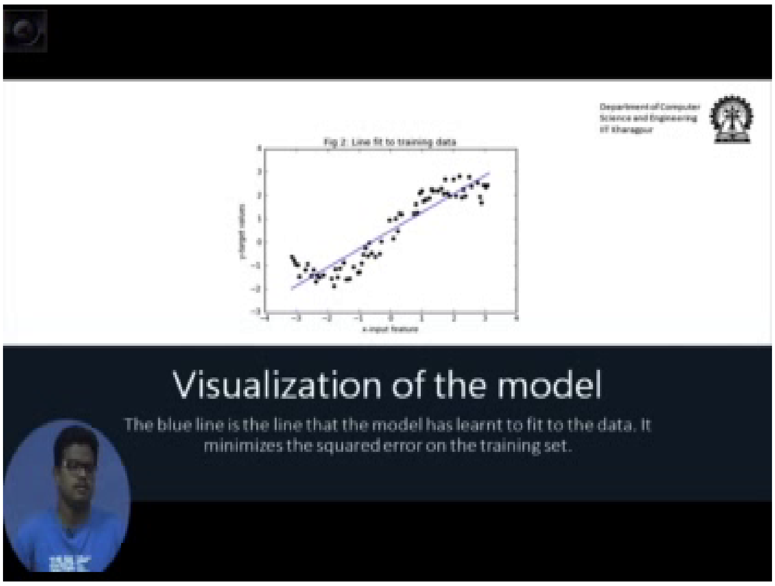
The next step is to Fit a linear model on to the training set. And over here we first invoke the linear regression, the class and we create an object of this particular class and this particular class has all necessary functions that are required for working with linear regression models; that is it.

And we first like create copies of our training set - the x train and y train are the training features, the inputs and the labels; and we reshape them into the particular convention which is necessary for the linear regression object and we create copies of these data sets because it tends to over write the examples if we do not do it, so that is why. So, first we do this and first the data is ready, so that is what we have in the second line. So, x train for line fitting and y train for line fitting, they are the input features and targets that is they (Refer Time: 06:48) outputs of course, respectively.

And next we go for the training part and in this particular episode what do we do, this is particular step we fit which we invoke model dot fit this function. So, this particular function does a least squared error minimization on the training set. So, whatever it does do for every single training example it calculates the prediction for the model and then it finds out what is the amount of deviation of this predicted value from the decided value which is also given to the model.

And now this deviation or the error is first squared, so that it becomes insensitive to the sign and then it is summed over the entire training set. And this particular error is the squared error that we are talking about, and this error is minimized through gradient descent learning. Finally, we get a model which minimizes the squared error.

(Refer Slide Time: 07:58)

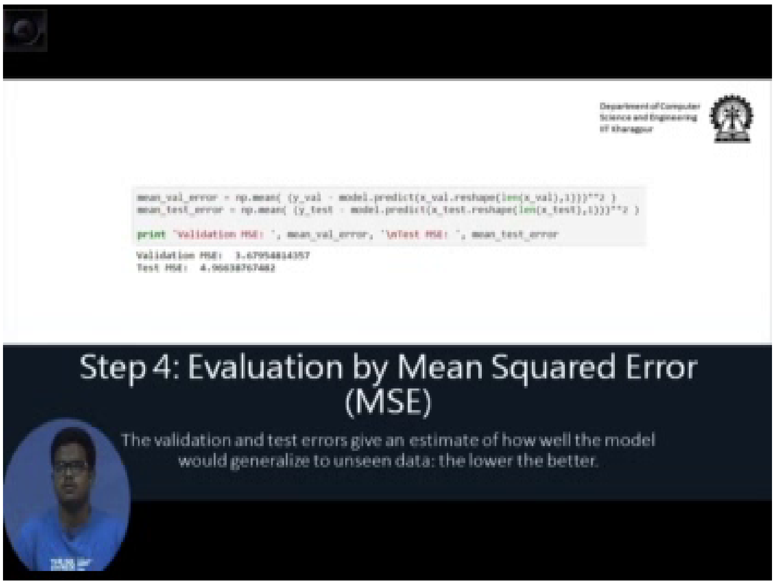


Visualization of the model

The blue line is the line that the model has learnt to fit to the data. It minimizes the squared error on the training set.

So, here we see that the model has learnt to a fit line to the data and this line minimizes a squared error on the training set, and you can see that it passes nicely through the distribution of samples.

(Refer Slide Time: 08:15)



```
mean_val_error = np.mean( (y_val - model.predict(x_val.reshape(len(x_val),1)))**2 )
mean_test_error = np.mean( (y_test - model.predict(x_test.reshape(len(x_test),1)))**2 )

print 'Validation MSE: ', mean_val_error, '\nTest MSE: ', mean_test_error

Validation MSE:  3.67954814357
Test MSE:  4.96638767482
```

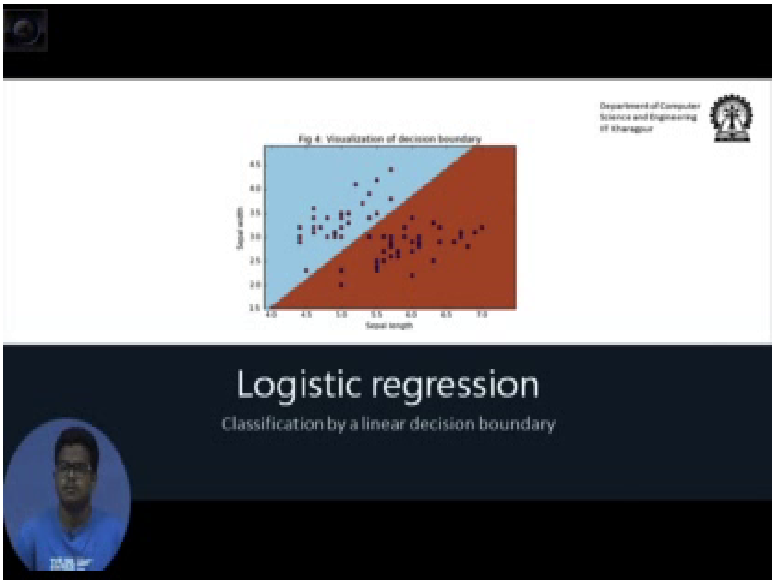
Step 4: Evaluation by Mean Squared Error (MSE)

The validation and test errors give an estimate of how well the model would generalize to unseen data: the lower the better.

And the fourth step is Evaluation of the model. So, in this particular step what do we do we find out how good the trained model generalizes. So, because it all matters to us, what all matters to us, is how good the model would perform on some unknown task. It does not matter to us how good the model would perform on some seen examples or the examples from the training as which we already know, we already know the shared outputs, the correct outputs for all of those examples they do not matter to us anymore, what matter to us is how good the model would perform when it is sent out to the world, to the unknown world and asked to perform.

So, that is what is meant by the generalization performance, and this generalization performance is to be measured next. So, what do we do? We find out the mean squared errors for the validation set and the test set. And those numbers can be calculated in using the code that is given.

(Refer Slide Time: 09:20)



Logistic regression

Classification by a linear decision boundary

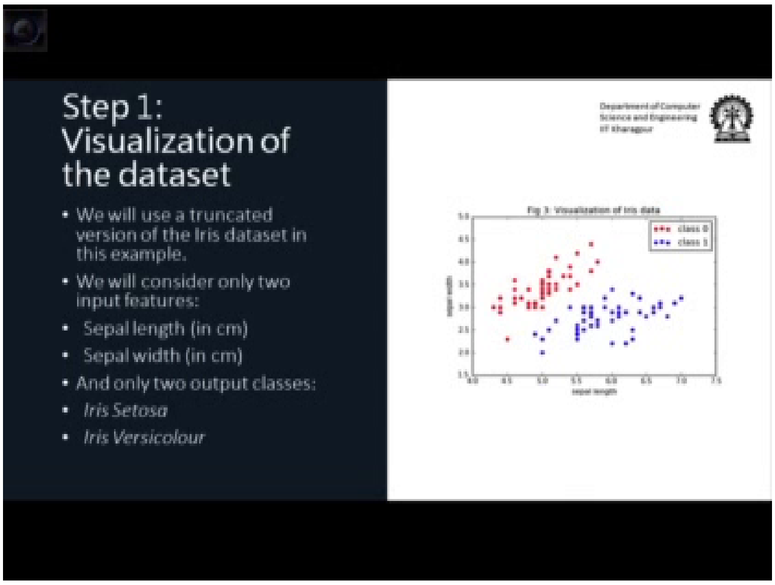
That was the end of the linear regression part. And all the code and instructions and explanations will be provided to you through an IPython notebook, which will be provided to you via the portal for this particular course, on NPTEL. Currently they are also on GitHub and I think that they will preserve these IPython notebooks on our GitHub page as well and those links will be provided in the course website, all right.

So, you would be able to directly use that particular notebook and you execute the cells one by one. So, each cell describes one particular step in the learning procedure, in the entire procedure. The entire procedure is broken up into steps and each step is within each cell and you execute a cell and it will execute the particular action that is required, all right. So, that is going to be really convenient for you people to practice and understand how a thing is done.

Next, we come to Logistic Regression. Logistic Regression is all about classification using a linear decision boundary. Now, you can look up on the web and find out how exactly this classification via linear decision boundary happens using logistic regression. Rather, I can just say in a few words that logistic regression model would try to, it is inherently a binary classifier, so it is a two class classifier and it can be trained to output a probability for the two classes.

Given a particular input example, it can out put the probability of it belonging to either class 0 or class 1 say, so like either true or false. Now you can like threshold the output into at like say 0.5 when the output which you are logistic regression unit would give is a between 0 and, so if you threshold that value at 0.5 will get classification output all right. So, this is how logistic regression works. So, then you can also look up on the web. So, logistic regression performs a classification via a linear decision boundary.

(Refer Slide Time: 11:38)



Step 1: Visualization of the dataset

- We will use a truncated version of the Iris dataset in this example.
- We will consider only two input features:
- Sepal length (in cm)
- Sepal width (in cm)
- And only two output classes:
- *Iris Setosa*
- *Iris Versicolour*

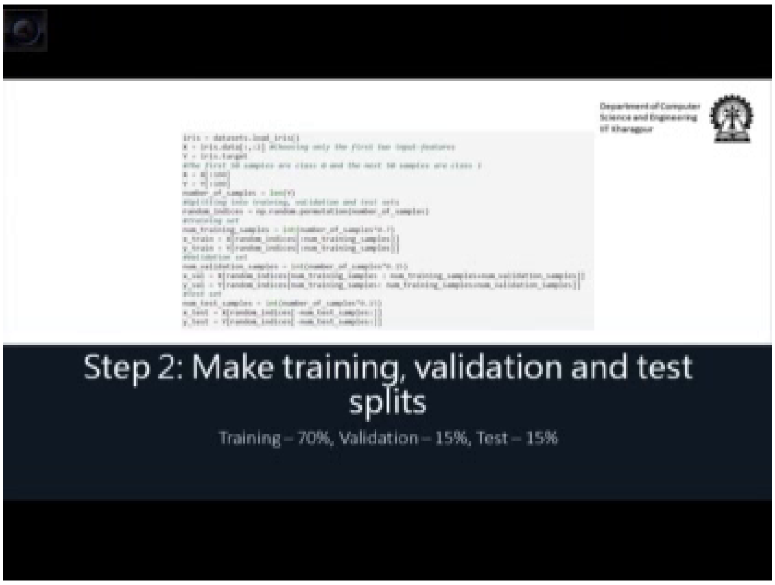
Department of Computer Science and Engineering
IIT Kharagpur

Fig 3: Visualization of Iris data

And we are going to use a very popular dataset it is called the Iris dataset, as in this particular example. So, this dataset is actually a dataset of Iris flowers. So, these plants come in 3 different species, 3 different kinds of flowers. So, the task is to classify these flowers on the basis of just 4 of their features viz, the sepal length, sepal width, petal length and petal width. Based on these 4 features you have to classify the flowers into one of the three different species.

So, this is the task all right. But to keep it simple and easy to visualize and appreciate I have decided to reduce the complexity of the dataset update. So, what we are going to do? We are just going to take up two features of the data viz, the sepal length and sepal width only and just consider two classes out of 3 possible classes of flower we are going to take two classes of the flowers, so the simplification is this. So, we use only two input features, the first two and the first two classes, all right.

(Refer Slide Time: 13:05)



Department of Computer Science and Engineering IIT Kharagpur

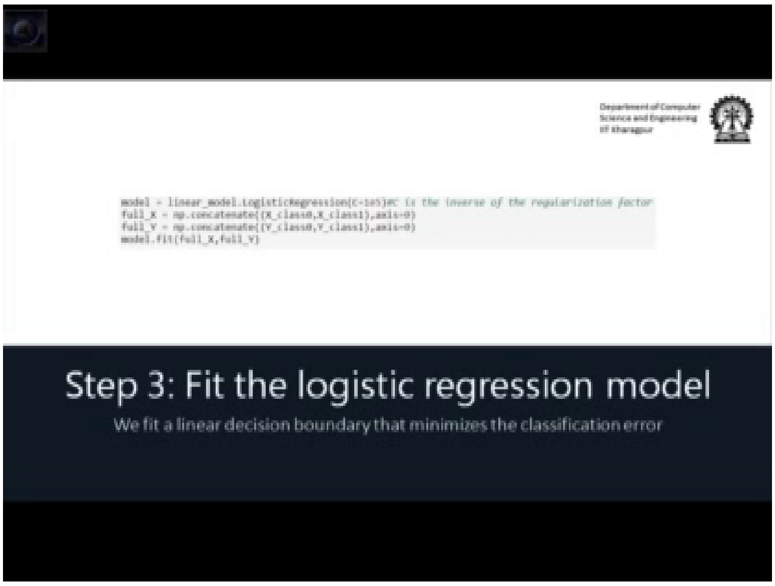
```
iris = datasets.load_iris()
X = iris.data[:,:2] #Choosing only the first two input features
Y = iris.target
#The first 50 samples are class 0 and the next 50 samples are class 1
X = X[:100]
Y = Y[:100]
number_of_samples = len(Y)
#Splitting into training, validation and test sets
random_indices = np.random.permutation(number_of_samples)
#Training set
num_training_samples = int(number_of_samples*0.7)
x_train = X[random_indices[:num_training_samples]]
y_train = Y[random_indices[:num_training_samples]]
#Validation set
num_validation_samples = int(number_of_samples*0.15)
x_val = X[random_indices[num_training_samples : num_training_samples+num_validation_samples]]
y_val = Y[random_indices[num_training_samples: num_training_samples+num_validation_samples]]
#Test set
num_test_samples = int(number_of_samples*0.15)
x_test = X[random_indices[-num_test_samples:]]
y_test = Y[random_indices[-num_test_samples:]]
```

Step 2: Make training, validation and test splits

Training – 70%, Validation – 15%, Test – 15%

So, we do this same exercise as before we split the data into training, validation and test sets as it is necessary, and then we fit the logistic regression model.

(Refer Slide Time: 13:14)



Department of Computer Science and Engineering IIT Kharagpur

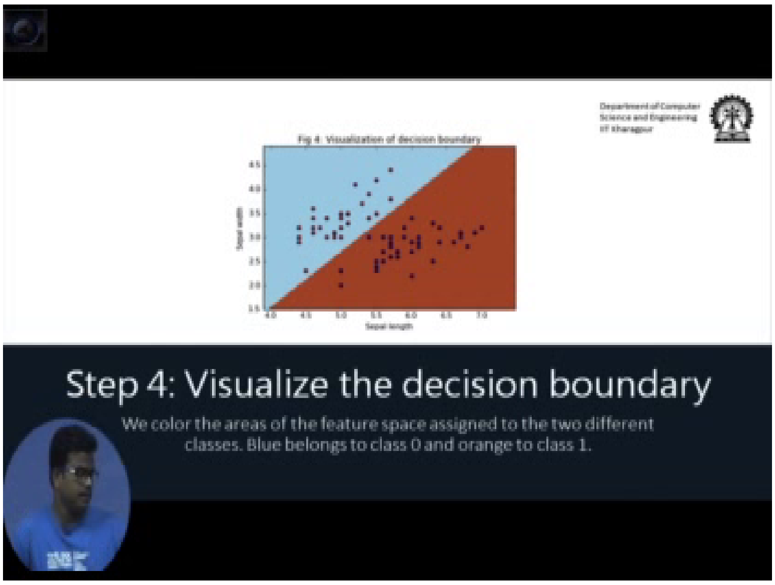
```
model = linear_model.LogisticRegression(C=1e5)#C is the inverse of the regularization factor
full_X = np.concatenate((X_class0,X_class1),axis=0)
full_Y = np.concatenate((Y_class0,Y_class1),axis=0)
model.fit(full_X,full_Y)
```

Step 3: Fit the logistic regression model

We fit a linear decision boundary that minimizes the classification error

So, we invoke the logistic regression class and create an object and then we do a model dot fit. So, this does the minimum error approximation. This particular algorithm will minimize the classification error and you can find the full documentation on official website of Scikit learn.

(Refer Slide Time: 13:47)

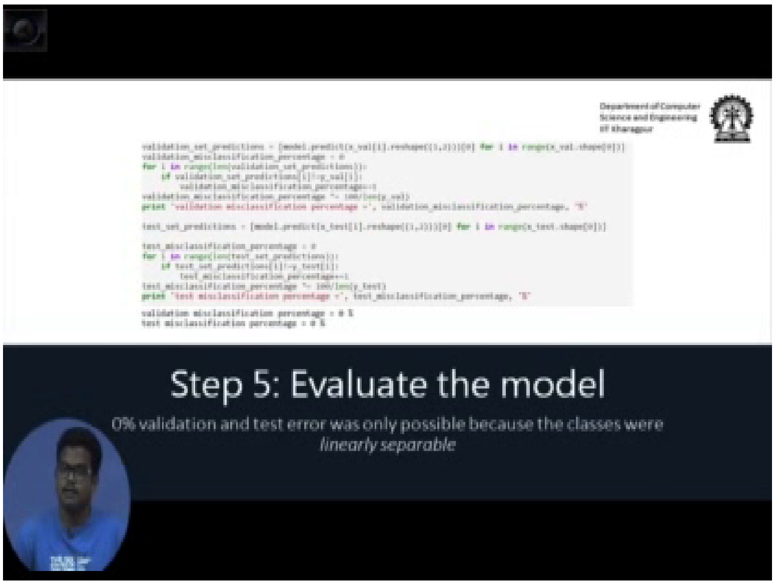


Step 4: Visualize the decision boundary

We color the areas of the feature space assigned to the two different classes. Blue belongs to class 0 and orange to class 1.

Once this model is trained you can find that this model creates a linear decision boundary, as I said before. So, two different classes are easily linearly separable in this case and that is why it is possible to draw such a beautiful decision boundary which correctly divides the entire features space into two fragments right. So, as you can see that the blue region belongs to class 0 according to the model, and the orange one belong to class 1.

(Refer Slide Time: 14:23)


Department of Computer Science and Engineering IIT Kharagpur

```
validation_set_predictions = [model.predict(x_val[i].reshape((1,2)))[0] for i in range(x_val.shape[0])]
validation_misclassification_percentage = 0
for i in range(len(validation_set_predictions)):
    if validation_set_predictions[i]!=y_val[i]:
        validation_misclassification_percentage+=1
validation_misclassification_percentage *= 100/len(y_val)
print 'validation misclassification percentage =', validation_misclassification_percentage, '%'

test_set_predictions = [model.predict(x_test[i].reshape((1,2)))[0] for i in range(x_test.shape[0])]

test_misclassification_percentage = 0
for i in range(len(test_set_predictions)):
    if test_set_predictions[i]!=y_test[i]:
        test_misclassification_percentage+=1
test_misclassification_percentage *= 100/len(y_test)
print 'test misclassification percentage =', test_misclassification_percentage, '%'
```

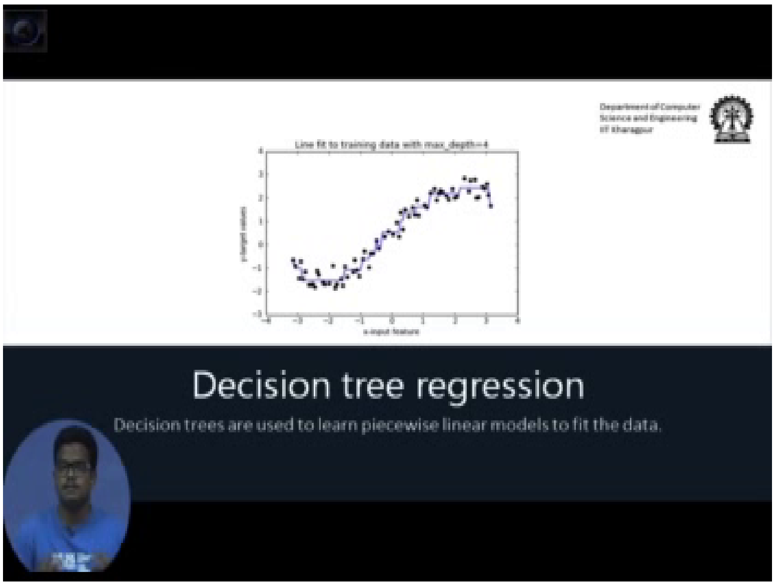
```
validation misclassification percentage = 0 %
test misclassification percentage = 0 %
```

Step 5: Evaluate the model

0% validation and test error was only possible because the classes were *linearly separable*

Finally, we evaluate the model and we find that we have 0 percent misclassification error, both on validation and test sets. So, it completely generalizes according to as far as we know according to the data that has been provided to us. So, it is just because of the case that we were fortunate to have the two classes that we picked to be linearly separable in the particular features space that we choose and that is not the case in most of the real world situations, but just to keep things simple and just to give you people an idea of how logistic regression works, I choose this example, alright.

(Refer Slide Time: 15:10)

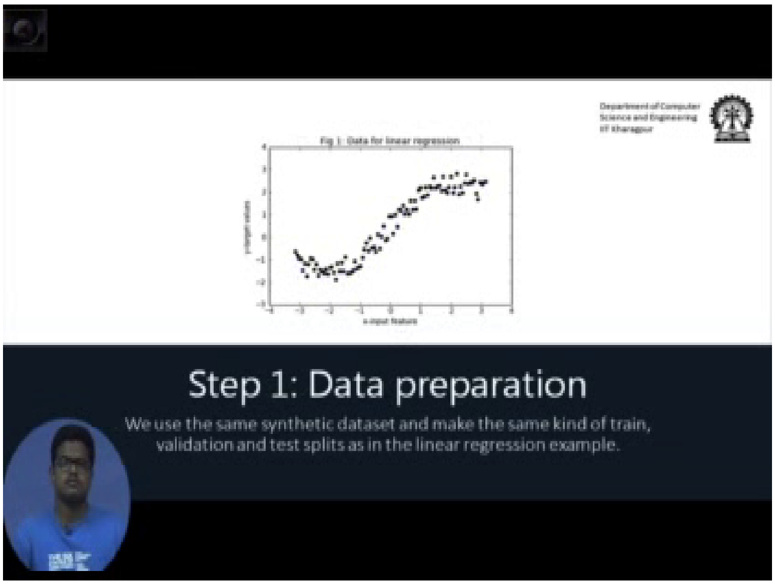


Decision tree regression

Decision trees are used to learn piecewise linear models to fit the data.

Next, we enter Decision Tree Regression. Decision trees are piecewise linear models, all right. The linear regression and logistic regression, they were like single line fitting models, at least the variant that I showed you the particular model that I demonstrated in this tutorial. But a decision tree tries to learn a set of if else conditions, if then else conditions. Like, if the input be in a particular range of the features space or in a particular place of features space then it is going to fit one particular line to the data. So, it is a piecewise linear model, right.

(Refer Slide Time: 15:55)



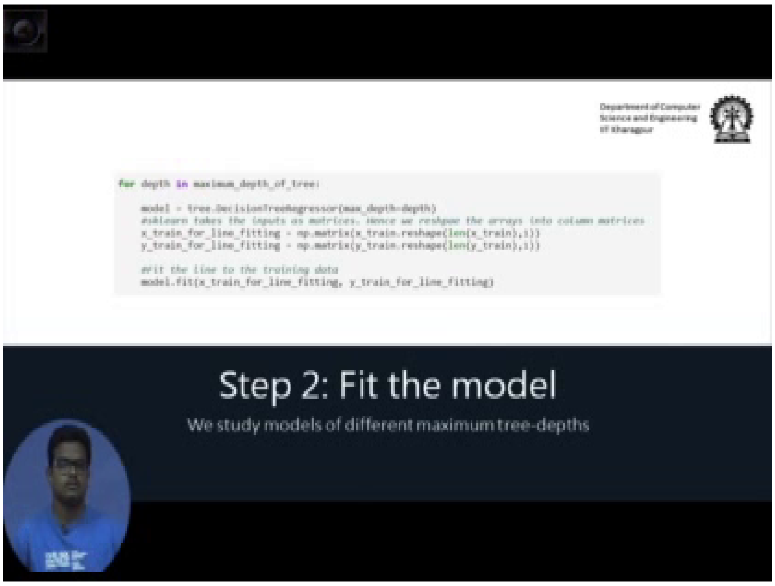
Department of Computer Science and Engineering IIT Kharagpur

Step 1: Data preparation

We use the same synthetic dataset and make the same kind of train, validation and test splits as in the linear regression example.

And we use the same dataset, same synthetic dataset that we used in the linear regression example.

(Refer Slide Time: 16:02)



Department of Computer Science and Engineering IIT Kharagpur

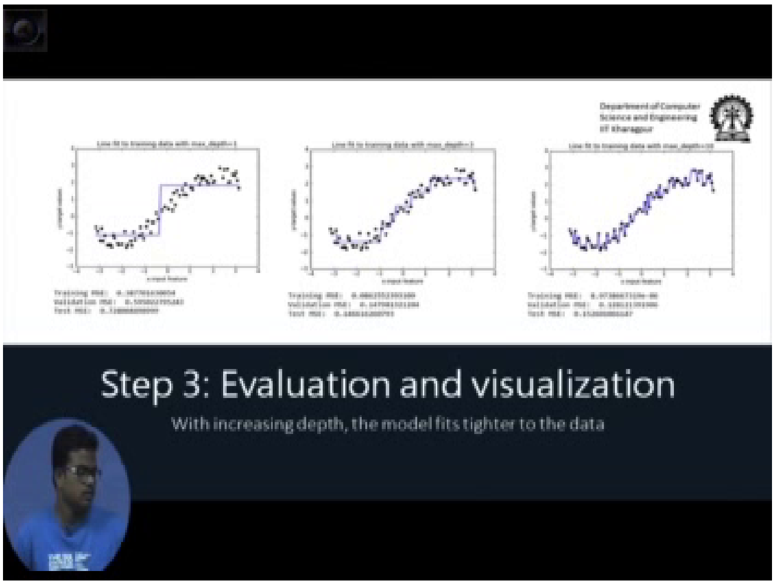
```
for depth in maximum_depth_of_tree:

    model = tree.DecisionTreeRegressor(max_depth=depth)
    #sklearn takes the inputs as matrices. Hence we reshpae the arrays into column matrices
    x_train_for_line_fitting = np.matrix(x_train.reshape(len(x_train),1))
    y_train_for_line_fitting = np.matrix(y_train.reshape(len(y_train),1))

    #fit the line to the training data
    model.fit(x_train_for_line_fitting, y_train_for_line_fitting)
```

Step 2: Fit the model

We study models of different maximum tree-depths

And finally, we Fit the model. So, we invoke the decision tree regressor class, we create an object of that class and we fit the model by minimizing again the amount of deviation which we have from the true values. Now, what we are going to do in this particular section of the tutorial is understand how we can engineer the maximum depth parameter of a decision tree to control the amount of bias and variance that is the end the model.

(Refer Slide Time: 16:40)



Line fit to training data with max_depth=1
Line fit to training data with max_depth=3
Line fit to training data with max_depth=10
x input feature
y target values

Training MSE: [illegible]
Validation MSE: [illegible]
Test MSE: [illegible]

Training MSE: [illegible]
Validation MSE: [illegible]
Test MSE: [illegible]

Training MSE: [illegible]
Validation MSE: [illegible]
Test MSE: [illegible]

Department of Computer Science and Engineering IIT Kharagpur

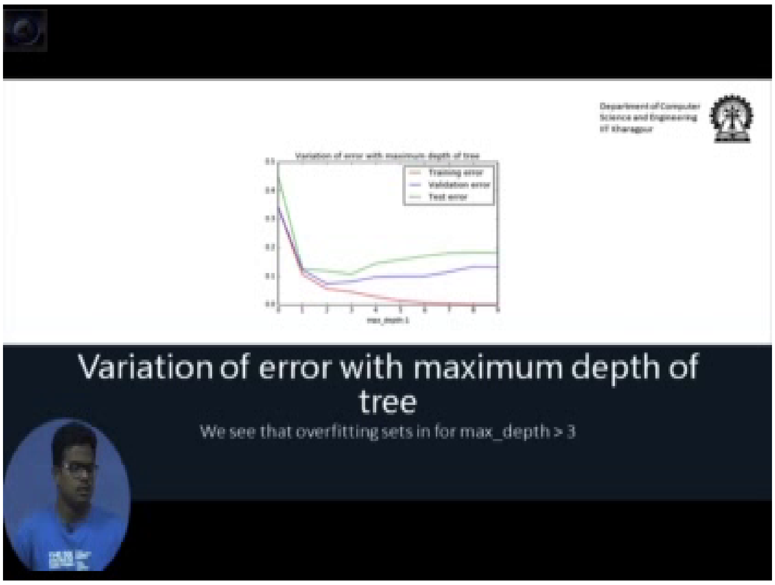
Step 3: Evaluation and visualization

With increasing depth, the model fits tighter to the data

So, we see that the left most example that you see that has a maximum depth of 1, so it is just single if else condition. So, if the data be lesser than 1 then, it is using one particular line and if it is greater than 1 then it is using another line and a very simple model that is (Refer Time: 17:05) is possible. So, maximum depth equal to 1. We can see that in both the training error and the generalization error which is given by the training and the (Refer Time: 17:15) validation and the test errors are high. And while, if you go to maximum depth equal to 3 then we can see that we have a piecewise linear model which fits really closely to the data, but may be it fits just well which we will come to the next slide and we will discuss about this. So, which one fits the best.

Whereas, if we increase the maximum depth to 10, then you can see that this tries to like fit the data really, really closely and this is not at all sustainable or at all generalizable because if you can see that it is trying to like hug on to every single point of the example set, of the trying set and most of these points are noisy because we explicitly super imposed some Gaussian noise on the data. So, this is not at all encouragable. So, that is why, that is why we should, like this particular module has very high variance and we should be able to identify this over fitting happening and choose a maximum depth which is lesser than this value.

(Refer Slide Time: 18:28)



Department of Computer Science and Engineering IIT Kharagpur

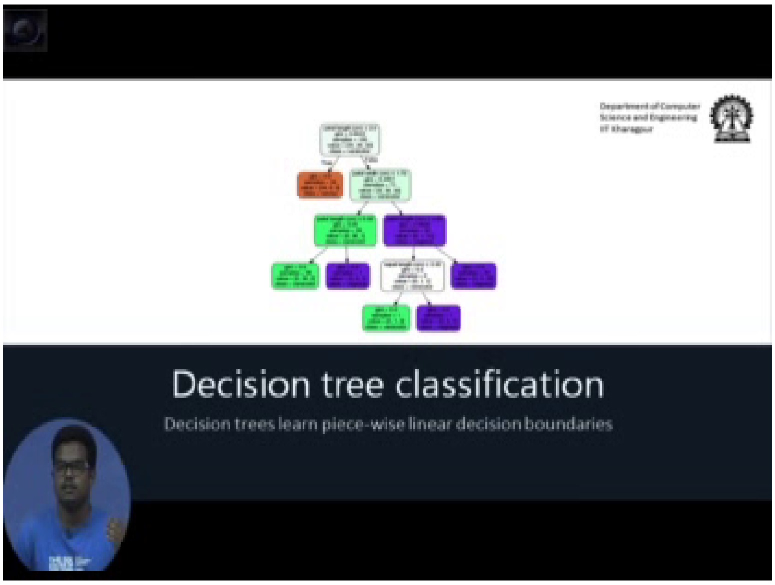
Variation of error with maximum depth of tree

We see that overfitting sets in for max_depth > 3

The next slide, this curve shows the variation of training validation and test errors with maximum depth, all right. So, as you can see that for maximum depth equal to 3 up to maximum depth equal to 3 rather training, validation and test errors, all of them keep decreasing or at least they are non-increasing. Whereas, as soon as we increase the maximum depth parameter beyond 3 this model starts over fitting and that is clear because the training error decreases, whereas the validation and test error keep tend to increase, all right.

So, this is what is done in the practical machine learning. So, we would rather choose like when for making the model which is going to be deployed we are going to choose maximum depth of 2 or 3. Rather, 2 is a far safer option because we do not really want the model to fit (Refer Time: 19:39) on the data because we want to keep it loose so that like it, kind of fits well on the data all right. So, it kind of fits well on the training data as well as it generalizes well in the real world. So, our preferred choice of max depth would be either 2 or 3 in this case.

(Refer Slide Time: 20:05)



Decision tree classification

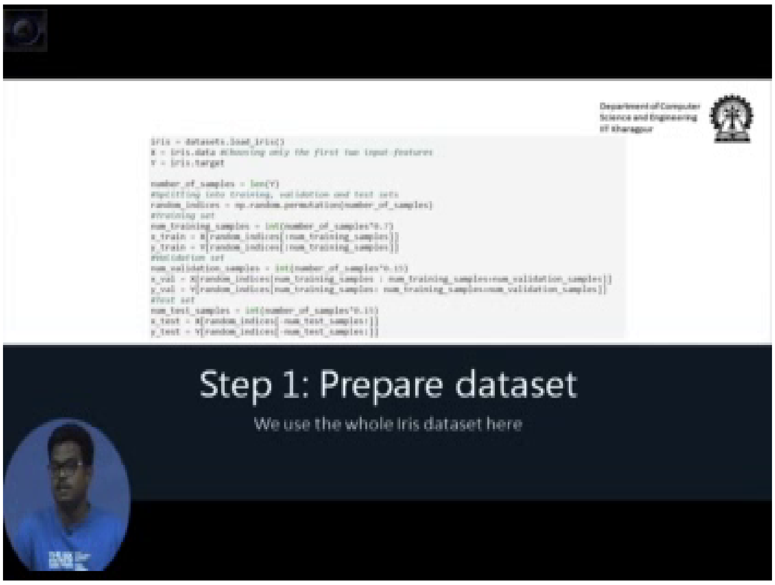
Decision trees learn piece-wise linear decision boundaries

Department of Computer Science and Engineering IIT Kharagpur

Next, let us look at the application of Decision trees in classification. So, decision trees learn piecewise linear decision boundaries. A single logistic regression classifier would learn a single line as a decision boundary whereas, decision trees can like learn piecewise linear model.

So, what do we mean by that? We mean by that the feature space is going to be divided into small volumes and within each single volume the decision tree is going learn one particular hyper plane, which is the generalization of a line and that is general linear function. So, this particular kind of behavior happens when decision trees used in classification and that is what we are going to do next.

(Refer Slide Time: 20:54)



Department of Computer Science and Engineering IIT Kharagpur

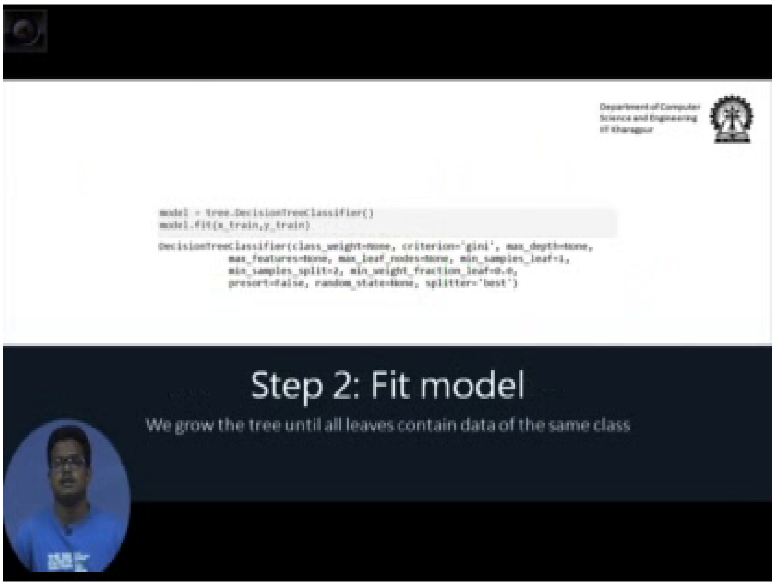
```
iris = datasets.load_iris()
X = iris.data #Choosing only the first two input features
Y = iris.target

number_of_samples = len(Y)
#Splitting into training, validation and test sets
random_indices = np.random.permutation(number_of_samples)
#Training set
num_training_samples = int(number_of_samples*0.7)
x_train = X[random_indices[:num_training_samples]]
y_train = Y[random_indices[:num_training_samples]]
#Validation set
num_validation_samples = int(number_of_samples*0.15)
x_val = X[random_indices[num_training_samples : num_training_samples+num_validation_samples]]
y_val = Y[random_indices[num_training_samples: num_training_samples+num_validation_samples]]
#Test set
num_test_samples = int(number_of_samples*0.15)
x_test = X[random_indices[-num_test_samples:]]
y_test = Y[random_indices[-num_test_samples:]]
```

Step 1: Prepare dataset

We use the whole Iris dataset here

So, we are going use the entire of Iris dataset here. So, Iris dataset has 4 input features and 3 output classes. And, do the same kind of training validation and test splitting.

(Refer Slide Time: 21:08)



Department of Computer Science and Engineering IIT Kharagpur

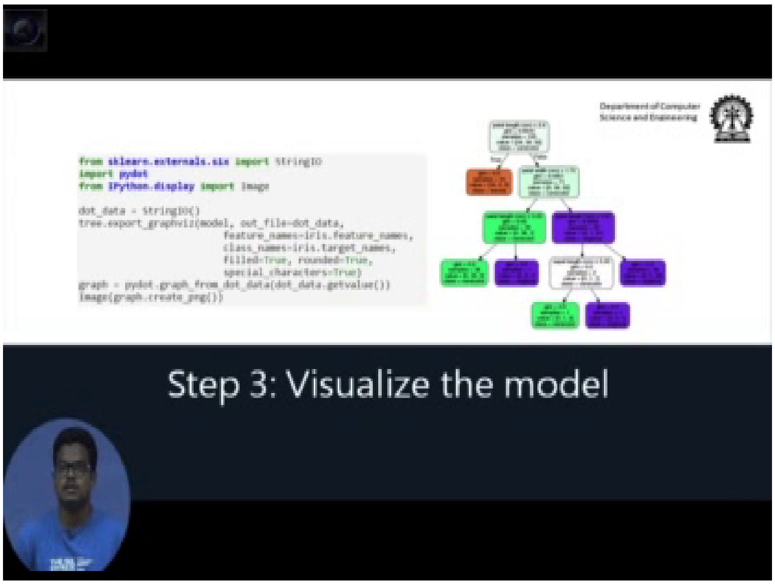
```
model = tree.DecisionTreeClassifier()
model.fit(x_train,y_train)

DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best')
```

Step 2: Fit model

We grow the tree until all leaves contain data of the same class

And next, we create an object of the decision tree classifier class and we fit the model, all right. We fit the model I am sorry, yes, and this also minimizes that classification error.

(Refer Slide Time: 21:28)

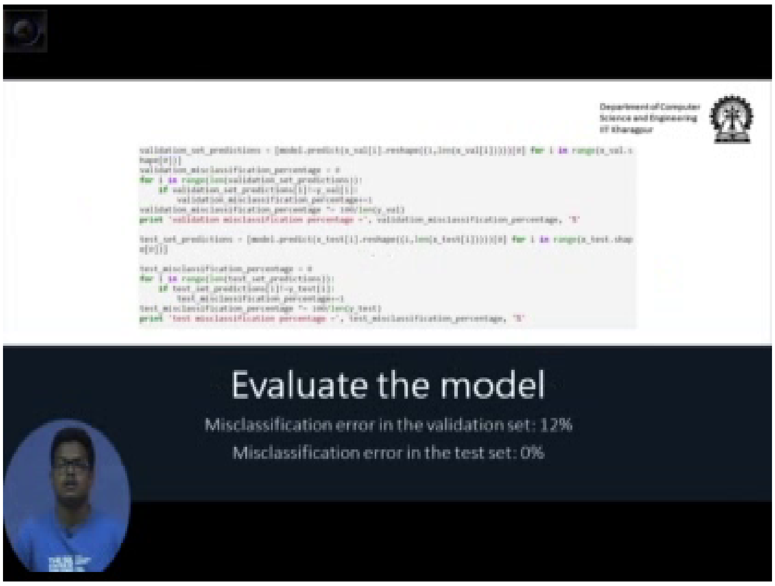


```
from sklearn.externals.six import StringIO
import pydot
from IPython.display import Image

dot_data = StringIO()
tree.export_graphviz(model, out_file=dot_data,
                     feature_names=iris.feature_names,
                     class_names=iris.target_names,
                     filled=True, rounded=True,
                     special_characters=True)
graph = pydot.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Step 3: Visualize the model

So, this is one piece of code and the source of this code is also given in the IPython notebook which will be also presented on the course website, you can find it out there. So, this is the nice way of visualizing the decision tree at what particular decision node, what kind of decision is getting learned and as you can see that finally, the model learns to you know the each of these leaves has a Gini index of 0.

(Refer Slide Time: 21:55)



```
validation_set_predictions = [model.predict(x_val[i].reshape((1,len(x_val[i]))))[0] for i in range(x_val.s
hape[0])]
validation_misclassification_percentage = 0
for i in range(len(validation_set_predictions)):
    if validation_set_predictions[i]!=y_val[i]:
        validation_misclassification_percentage+=1
validation_misclassification_percentage *= 100/len(y_val)
print 'validation misclassification percentage =', validation_misclassification_percentage, '%'

test_set_predictions = [model.predict(x_test[i].reshape((1,len(x_test[i]))))[0] for i in range(x_test.shap
e[0])]

test_misclassification_percentage = 0
for i in range(len(test_set_predictions)):
    if test_set_predictions[i]!=y_test[i]:
        test_misclassification_percentage+=1
test_misclassification_percentage *= 100/len(y_test)
print 'test misclassification percentage =', test_misclassification_percentage, '%'
```

Evaluate the model

Misclassification error in the validation set: 12%

Misclassification error in the test set: 0%

The Gini index, a Gini coefficient is a measure of how homogenous a cluster is. So, gini index or gini coefficient of 0.0 means that the cluster of inputs or the set of inputs that are coming to that particular leaf node are all belonging to one single class. So, they are completely homogenous. So, when you like push in one example an unknown example through the decision tree and it lands up in one particular leaf you are 100 percent sure that what particular class it is going to be because each leaf corresponds to one class.

This is like one kind of decision tree making procedure, making philosophy there can be other variance as well. So, we will discuss it in the next classes or we will supply links to you web links which give better description of how complicated decision trees and decision forests are learnt to classify the data well.

Next is evaluation of the model. And we do it in the same way as we had done in the last three examples and we get a validation set error of 12 percent because see, the previous one which we did it was a simplified version of the iris dataset and the beauty of the iris dataset is that the first and the second classes they are linearly separable, but they are not linearly separable with the third class. So, as soon as the third class comes it is difficult to fit a linear decision boundary and that is why get some non zero validation and validation error.

In this case, the test error is luckily 0 just because the third class at come in and sets where not any more linearly separable and there is a nice way of visualizing decision boundaries learnt at (Refer Time: 24:15) single of these nodes. So, every single node or rather decision node learns a particular decision rule. So, it learns to choose certain features and splits on those features on that particular chosen feature access which increases the homogeneity in the clusters going into the children nodes of that particular decision node and so, we have 12 percent validation error in this example.

So, you can try out these models using other input data and we can discuss the performances on the course forum.

Thank you so much. Thank you so much for attending my first ever video recording. Thank you so much, see you guys in the next video and we will be discussing principle component analysis and features selection which also form very crucial aspects of practical machine learning.

Bye bye, see you next time.