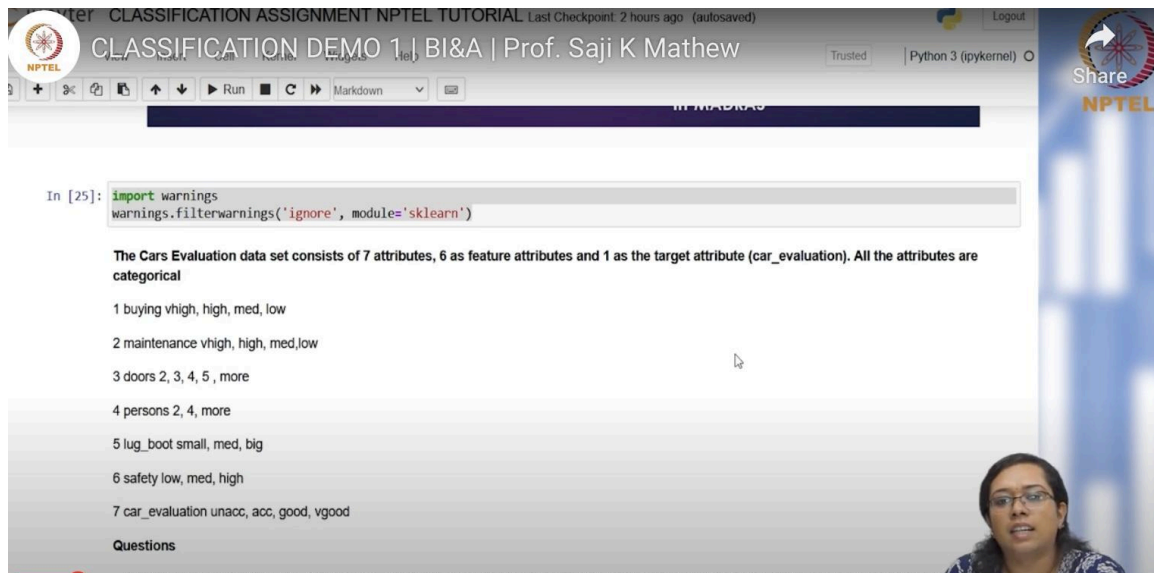


Course Name: Business Intelligence and Analytics
Professor Name: Prof. Saji.K.Mathew
Department Name: Department of Management Studies
Institute Name: Indian Institute of Technology Madras
Week: 08
Lecture: 30

CLASSIFICATION DEMO 1

Good morning all. Welcome to today's Business Intelligence and Analytics lecture. Today we are going to see a classification assignment tutorial. So this is basically a supervised machine learning method, as you all know because we have studied what classification is, what clustering is. So this is a supervised machine learning model as opposed to clustering which is a unsupervised ML model. So today we are going to see a car based data set in which we have to classify the cars in, into various classes.

So we already have a data set with us and we are using it to see how we can classify using different decision tree, that is CART algorithm and random forest algorithm that we have already studied previously in the class. So we will move into that. We have taken Jupyter notebook and we are going to work in Jupyter notebook for working in Python language. So this is how it can be seen in Python notebook.



The screenshot shows a Jupyter Notebook interface with the following content:

```
In [25]: import warnings
warnings.filterwarnings('ignore', module='sklearn')
```

The Cars Evaluation data set consists of 7 attributes, 6 as feature attributes and 1 as the target attribute (car_evaluation). All the attributes are categorical

- 1 buying vhigh, high, med, low
- 2 maintenance vhigh, high, med, low
- 3 doors 2, 3, 4, 5, more
- 4 persons 2, 4, more
- 5 lug_boot small, med, big
- 6 safety low, med, high
- 7 car_evaluation unacc, acc, good, vgood

Questions

1. Build decision trees from the data using algorithms discussed in the class. Report model performance based on accuracy, error, sensitivity (recall), specificity (precision).

So first what you have to do is you have to import the CSV file or, you know give the path name so that you can import the data set and then we will be working with the data

set. So I already have coded the entire thing and we will be just running and seeing the output and I will keep explaining what is going to happen and we have the questions and answers with us. So initially we will be mostly using the scikit-learn library. So you all might be knowing what is scikit-learn library and in that, I am just you know inserting import warning in which it will just filter out the warnings because you know, there will be so many warnings as we proceed. So I am just filtering out the warning that is the first code in this.

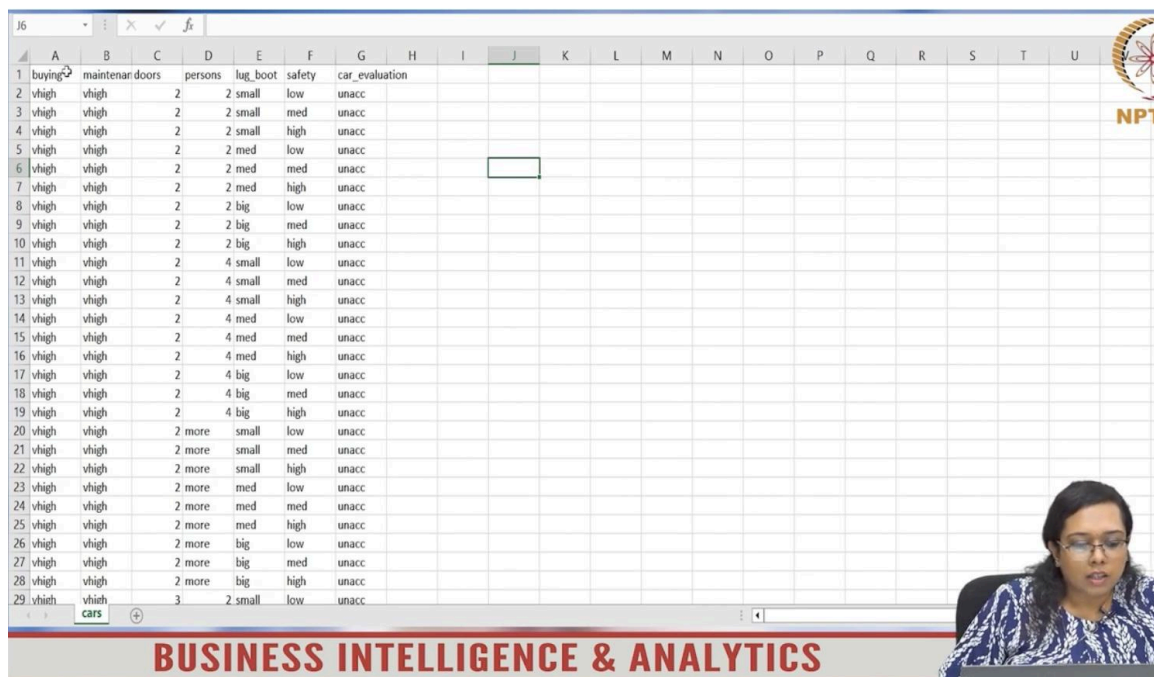
And so basically we are going to use decision tree algorithm to build the classification models and we have a car data set with us and as I already told decision trees are a supervised learning algorithm. It can be used for both classification as well as regression task. So we will just see the metadata of the cars data set that we are having. So it is having 7 attributes, totally 7 attributes which are buying, maintenance, doors, persons, luggage boot, safety and the final or the target variable is car evaluation. So the input features are 1 to 6 which is buying to safety and as you can see from the screen and the seventh one that is the target variable.

CLASSIFICATION DEMO 1



Video

Lec.no: 30

Next Lecture



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	buying	maintenan	doors	persons	lug_boot	safety	car_evaluation														
2	vhigh	vhigh	2	2	small	low	unacc														
3	vhigh	vhigh	2	2	small	med	unacc														
4	vhigh	vhigh	2	2	small	high	unacc														
5	vhigh	vhigh	2	2	med	low	unacc														
6	vhigh	vhigh	2	2	med	med	unacc														
7	vhigh	vhigh	2	2	med	high	unacc														
8	vhigh	vhigh	2	2	big	low	unacc														
9	vhigh	vhigh	2	2	big	med	unacc														
10	vhigh	vhigh	2	2	big	high	unacc														
11	vhigh	vhigh	2	4	small	low	unacc														
12	vhigh	vhigh	2	4	small	med	unacc														
13	vhigh	vhigh	2	4	small	high	unacc														
14	vhigh	vhigh	2	4	med	low	unacc														
15	vhigh	vhigh	2	4	med	med	unacc														
16	vhigh	vhigh	2	4	med	high	unacc														
17	vhigh	vhigh	2	4	big	low	unacc														
18	vhigh	vhigh	2	4	big	med	unacc														
19	vhigh	vhigh	2	4	big	high	unacc														
20	vhigh	vhigh	2	more	small	low	unacc														
21	vhigh	vhigh	2	more	small	med	unacc														
22	vhigh	vhigh	2	more	small	high	unacc														
23	vhigh	vhigh	2	more	med	low	unacc														
24	vhigh	vhigh	2	more	med	med	unacc														
25	vhigh	vhigh	2	more	med	high	unacc														
26	vhigh	vhigh	2	more	big	low	unacc														
27	vhigh	vhigh	2	more	big	med	unacc														
28	vhigh	vhigh	2	more	big	high	unacc														
29	vhigh	vhigh	3	2	small	low	unacc														
		cars																			



BUSINESS INTELLIGENCE & ANALYTICS

So target variable, we are having how many classes? Four classes. First class is unacceptable, acceptable, good and very good. So basically we are having four classes as target variables into which, at least one of the classes we have to classify a car into. So what we will be doing in normal classification algorithms is, firstly we will split it into training set and test set. So training set we will use for model building and then we will test it out in the test set.

That is pretty much what we are going to do here also. So every car in the test set has to be in one of the target classes that is unacceptable, acceptable, good or very good car. So how will we train this data? We have 6 input features and I will just show the data set which is in excel file, so that you will get a better idea. So this is the excel file of cars data set which is in excel format and as you can see the columns A, B, C, D, E and F, they are the input features and car evaluation which is in G column, that is the output target variable to which we have to classify the cars into. So there are, this is actually a big data set with thousands of entries, so as you can see from the screen and can you see what classes are there in each of the input variables? So the values under each input variable, let us take buying.

So it has values like, you know very high. What is actually buying? This buying attribute actually describes the buying price of the car. So basically it has 4 categorical values. These are not numerical values, what we have got in the data set is not a categorical value. It is a categorical value and not a numerical value because it is in alphabetical form, very high, high and all those.

So we will just scroll and see what all values are there. So it is very high, high, then there is medium and then low. So there are 4 categorical values which is very high, high, medium and low. And the second input feature is maintenance. So basically this represents the maintenance price of the car and has 4 categorical values which is, it is similar to buying.

So very high, high, medium and low. So this first input feature and the second input feature basically has the same categorical values. Then the third one, third input feature is doors. So doors has, as you can see from the name itself, it represents how many doors are there in the car. So it has values starting from 2 which is the least one and then 3, 4, 3 is there, 4 is there, then 5 and more is there.

So basically cars will have doors starting from the range of 2 and then increases. So as we can see, this has numerical input and then the fourth input feature is the persons. That is the number of people who can be seated in the car or rather we can call it as the seating capacity of the car. So it has, persons has 3 values, that is 2, 4 and more. So it

has 3 values.

Then the fifth input feature is the luggage, luggage boot. So this attribute indicates the size of the luggage boot or the trunk of the car. So it has 3 values which is small, medium and big. As you can see, it has 3 input features which are small, medium and big. Then the final input feature is called the safety and safety obviously, it represents the safety rating of the car.

So it also has 3 values which is low, medium and high safety. And based on this input features we have the data set according to which the cars have been classified into either of the 4 classes. So we will go back into the code to see how this can be worked out. As we saw in the excel file, there were 6 input features and 1 target variable to which the car had to be classified into. So these are the questions which we have to do in today's class which is, first question is, build decision trees from the data using algorithms discussed in class.

The Cars Evaluation data set consists of 7 attributes, 6 as feature attributes and 1 as the target attribute (car_evaluation). All the attributes are categorical

- 1 buying vhigh, high, med, low
- 2 maintenance vhigh, high, med, low
- 3 doors 2, 3, 4, 5, more
- 4 persons 2, 4, more
- 5 lug_boot small, med, big
- 6 safety low, med, high
- 7 car_evaluation unacc, acc, good, vgood

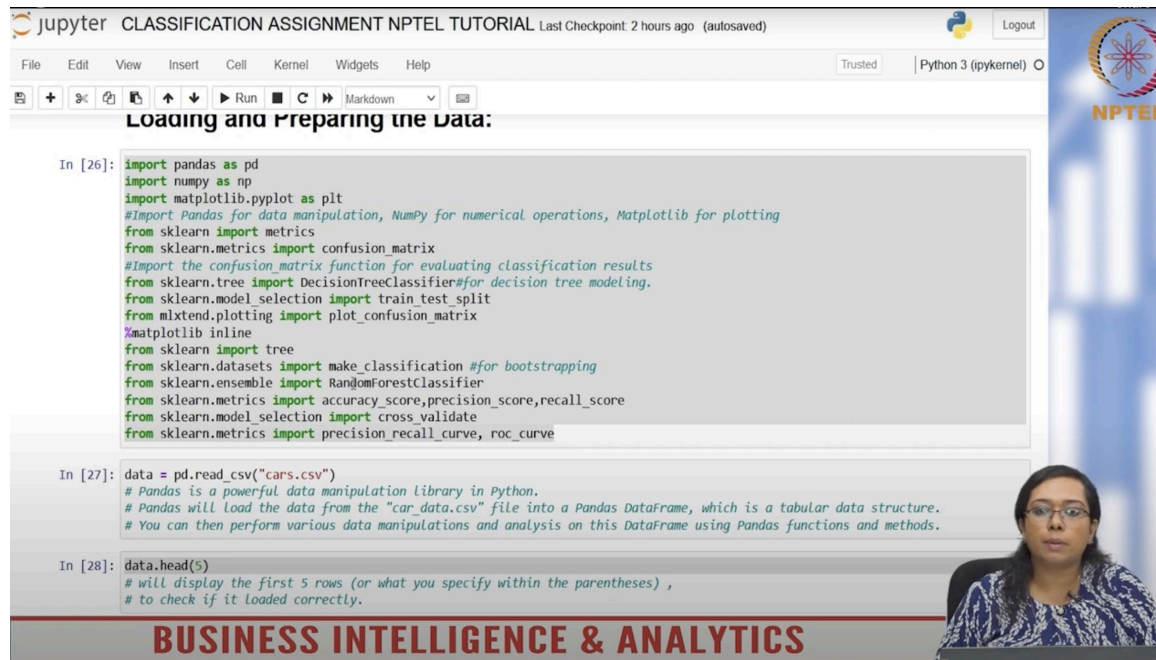
Questions

1. Build decision trees from the data using algorithms discussed in the class. Report model performance based on accuracy, error, sensitivity (recall), specificity, and precision using k-fold cross validation.
2. Compare the models using ROC curves and Precision-Recall graphs
3. Compute precision and recall using micro and macro averaging approaches. Interpret the results.

BUSINESS INTELLIGENCE & ANALYTICS

So we have discussed pretty much algorithms which will belong to the category of classification like CART algorithm, random forests and so on. And we have to report the model performance based on various matrices like accuracy, error, sensitivity or recall and specificity and precision. And we have to use the k fold cross validation approach also which was also discussed in the class. Then the second question is we have to compare the models using ROC curve. So ROC curve also, you must be familiar with and also the precision recall graphs which is a plot where precision is plotted against the recall for various models.

And the third question is, we have to compute the precision and recall using micro and macro averaging approaches. So you can read about micro and macro averaging approaches but we will also discuss that in the class. And finally we have to interpret their cells. So that is what is the agenda of today. So we will move into the first part which is loading and preparing the data.



The screenshot shows a Jupyter Notebook interface with the following content:

CLASSIFICATION ASSIGNMENT NPTEL TUTORIAL Last Checkpoint: 2 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Loading and Preparing the Data:

```
In [26]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
#Import Pandas for data manipulation, NumPy for numerical operations, Matplotlib for plotting
from sklearn import metrics
from sklearn.metrics import confusion_matrix
#Import the confusion_matrix function for evaluating classification results
from sklearn.tree import DecisionTreeClassifier#for decision tree modeling.
from sklearn.model_selection import train_test_split
from mlxtend.plotting import plot_confusion_matrix
%matplotlib inline
from sklearn import tree
from sklearn.datasets import make_classification #for bootstrapping
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,precision_score,recall_score
from sklearn.model_selection import cross_validate
from sklearn.metrics import precision_recall_curve, roc_curve
```

```
In [27]: data = pd.read_csv("cars.csv")
# Pandas is a powerful data manipulation library in Python.
# Pandas will load the data from the "car_data.csv" file into a Pandas DataFrame, which is a tabular data structure.
# You can then perform various data manipulations and analysis on this DataFrame using Pandas functions and methods.
```

```
In [28]: data.head(5)
# will display the first 5 rows (or what you specify within the parentheses) ,
# to check if it loaded correctly.
```

BUSINESS INTELLIGENCE & ANALYTICS

So in this step, we will be pretty much loading the data and preparing for getting into the task of actual classifications. So the initializing functions will be dealt with in the first step. So first line of the code is import pandas as pd. So pandas is a library as you all know. It is used for data manipulation mostly.

So what is pd? So pd is an alias or alias for pandas library that we are using in this code. So the second line is import numpy as np. So it is same. Numpy is used for numerical operations.

It is also a library. All these library are usually used in whenever you do coding in python. So numpy is mainly used for numerical operations. And the third one, the third library is matplotlib and, usually for plotting graphs, bar charts, pie charts, etc. Basically it does the matlab functions of plotting and here we are giving the alias as plt. So that is basically the first three lines of the code.

Now from sklearn, sklearn is the name for the scikit-learn library. So what we are doing is we are importing metrics. So the metrics module from the scikit-learn library, it

basically provides you with various metrics for evaluating these ML models that we are developing like accuracy, precision, recall and so on, f1 score and so on. So basically we are importing the metrics module from the scikit-learn library and from the metrics module we are importing the confusion matrix because we will be using that to assess the true positives, true negatives and all those. Here we are not engaging ourselves with a binary classification model wherein there are only two classes, yes or no and then you know you have to plot a 2x2 matrix.

We are not dealing with that. We have multiple classes. We actually have four classes with us, unacceptable, acceptable, good and very good. So there are four classes to which we have to classify our cars into. So it is basically a multi-class kind of problem that we are dealing with. Then the next line is from scikit-learn, we are importing the decision tree classifier.

So firstly we will be modeling our data using decision tree. So that is why we are importing the decision tree classifier class from the scikit-learn and next one is the train test split. So this function of train test split which belongs to the model selection module from the scikit-learn library, what it does is, it splits the data into test set and training set. So the percentage in which you have to split that is up to you. So you can give it as 70-30 split or 80-20 split.

So that is up to you how you divide the data into. So once that is done from mlexend library, we are also importing the plot confusion matrix. This is for getting us the plot of the confusion matrix. Then we are importing the tree. This is basically for functionality for decision tree modeling as well as visualizing the tree.

That is why we are importing the tree from the scikit-learn library. Then we are importing some other functions from the scikit-learn library, like make_classification which is for bootstrapping. This is majorly done before the random forest generation and all. So next is random forest classifier which is used for random forest and then I am just importing again the individual matrices like accuracy score, precision score, recall score etc. and also the cross_validate function because we have to do the k-fold cross validation.

So that is pretty much it about loading and preparing the data. Actually we have not yet loaded the data. So that is the next step. So what we are doing in this second line of code, that is data = pd.read_csv. So in that what we are doing is, data that is the name that we have given for the data frame and pd.read_csv, pd is the alias for pandas library. So what we are doing is, when we execute this line of code, it reads the data from car.csv and then it stores it in the data variable, which is the pandas data frame.

So data is the name for pandas data frame that we have given here and then after we have done this, we can perform many functions like data manipulation and various kinds of analysis that we are going to perform. So the next code in this is data.head and in brackets I have given 5. So I guess you know what it is. So it will just show you the first 5 rows of the data that we have.

The screenshot shows a Jupyter Notebook window titled "CLASSIFICATION ASSIGNMENT NPTEL TUTORIAL". The code cell contains the following text:

```
In [47]: data
# will display the first 5 rows (or what you specify within the parentheses) ,
# to check if it loaded correctly.
```

The output cell shows the following DataFrame:

	buying	maintenance	doors	persons	lug_boot	safety	car_evaluation
0	4	4	2	2	0	0	unacc
1	4	4	2	2	0	1	unacc
2	4	4	2	2	0	2	unacc
3	4	4	2	2	1	0	unacc
4	4	4	2	2	1	1	unacc
...
1723	1	1	3	2	1	1	good
1724	1	1	3	2	1	2	vgood
1725	1	1	3	2	2	0	unacc
1726	1	1	3	2	2	1	good
1727	1	1	3	2	2	2	vgood

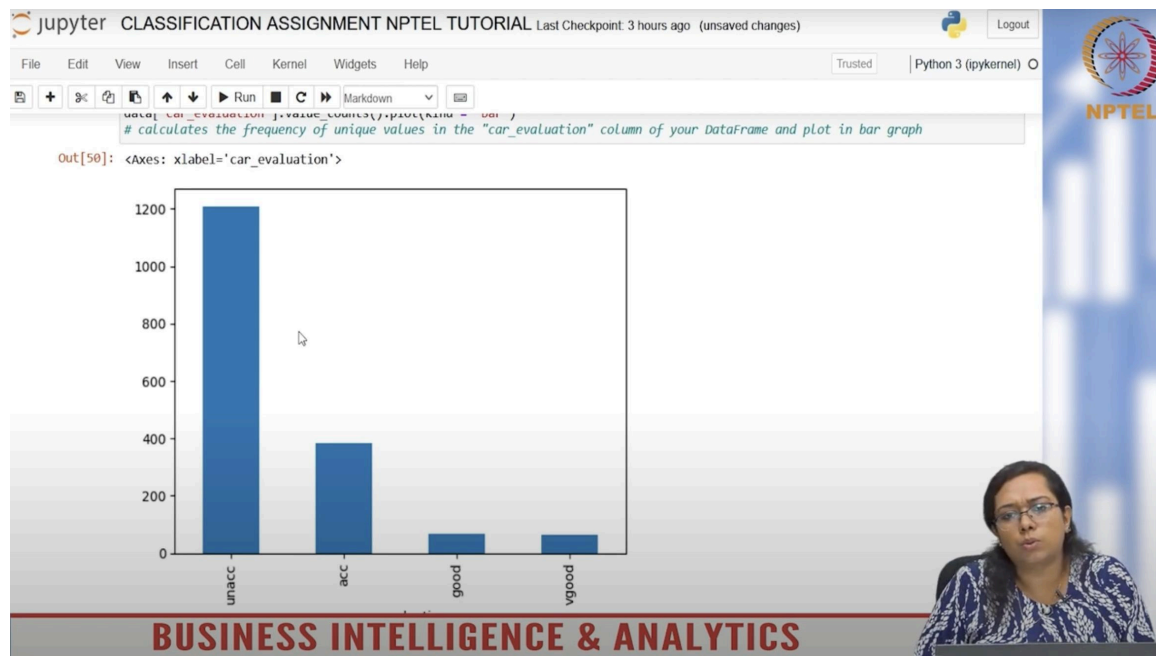
Below the DataFrame, it says "1728 rows x 7 columns". At the bottom of the notebook, the next code cell is visible:

```
In [29]: data["car_evaluation"].value_counts()
data["car_evaluation"].value_counts().plot(kind = "bar")
```

So if you are not giving anything then you can just print data as well. So what it will do is, it is running. So it will show how many rows are there, 1728 rows and 7 columns. That is the total number of columns and rows that we have in our cars.data file. So if you do not want to print the entire thing you can just give as data.head and within 10 brackets say 10. So if you run that you will get it. That is the first 10 rows that will be printed. So the next one, data within quotes car evaluation.

What is car evaluation? It is the target variable that we have. So we just want to know how imbalanced or balanced the data set or you know the target variable is. That is why we are printing this value counts. It is nothing but it will just tell you how many counts are there. So if I just comment this second line that is, and then run it, it will just show how many counts are there, the exact counts.

So unacceptable class is the biggest class. Most of the cars are unacceptable. So 1210 cars are in unacceptable category. Acceptable category we have 384 cars, good 69 and very good 65. So if you want to see this in a bar graph, then you can give this.



That is data of car evaluation.value counts and plot equal to bar. It can be any kind of plot. So here I have given a bar chart so that we can just visualize how imbalanced this data set is. So we are having a highly imbalanced data set over here. So we have pretty many things that we can do to imbalanced data, set so that we can balance it before training our model.

So if we work with imbalanced data set, with that when we test it then we will be having a highly biased model which will be highly biased towards the majority class. What is the majority class here? It is unacceptable class. So our model will tend to get biased in that direction, if we are just going towards modeling without balancing the data set. So we will be seeing how to balance the data set.

We have inbuilt functions for that. The next step that we are going to do is, once we have loaded the data and we have initialized certain functions and called the libraries, imported the libraries, what we are going to do is the data preprocessing stage, in which we have to check how in what kind of data type we have got the data set. That is, you know I will just show you the screen in which we have the excel file of the cars data set. So you can see which form of data set are we having here. So if we see, we have categorical values under the input features. That is, it is not numerical but the problem is that when we run the CART algorithm and so on, the algorithms tends to accept features or values which are numerical in nature and it gets biased when we give categorical inputs and sometimes it even rejects the categorical values that we have.

So what we have to do is, we have to do encoding or we have to change these categorical values into numerical values. So you must be knowing the types of data. There are categorical data, numerical data, ordinal data, interval ratio data. So you will be knowing I guess you will be knowing the difference of all those. So what we are having here is does it have a particular order, if you see, if we take this buying input feature, input label buying then we have four classes that is very high, high I guess it is low and yeah very high, high, medium and low.

So we have four class labels which are, which does it have a particular order, if you think like buying price if it is very high, very high beyond your capacity then you will not be buying that car. So it has a particular order for all these categorical inputs. So what we will do is, we will replace this according to their intensities. We will replace this into numerical values, that is what we are going to do in the next preprocessing of data step.

So we will go into that now. So what is called as value encoding is when we convert the categorical values into numerical values. In our current problem we can do multiple kinds of encoding. So there are multiple kinds of ways in which you can change the categorical into numerical data. So one thing is you can do it individually, like you go and map very high as 4, high as 3 and medium as 2 and then low as 1. So you can do the individual mapping and that is what is called as the ordinal mapping, in which you replace these with integers that will be taken as, you know so the problem is that, you know the ordinal encoding assumes that the difference between each value are the same like $2 - 1$ is 1, $1 - 0$ is 1.

So you know the difference is same, that is what it assumes. So it can lead to some kind of bias but we are going forward with this because when we see our data set, even though we have categorical values, all those categorical values are internally ordinal in nature because they have a particular order. Say safety, if it is very high we have a input feature as safety. So if safety is very high, we tend to buy that car. If safety is a bit low then we will be in confusion. If safety is very very less then we are not going to buy those cars.

So internally even though we have categorical values in our data set, those are ordinal. They have a implicit order in them. So what we can do is we can go for this ordinal encoding. So that is what we are doing in our code now. But there are multiple other encoding methods also which is called as one-hot encoding, in which it can handle both nominal and ordinal variable and what it will do is it will consider say, we have four classes apples, bananas, grapes and oranges.

Jupyter CLASSIFICATION ASSIGNMENT NPTEL TUTORIAL Last Checkpoint: 3 hours ago (autosaved) Python 3 (ipykernel)

Data preprocessing -Perform data transformation by replacing categorical values with numerical values using dictionaries (mappers)

Value Encoding ¶

Since all of the variables in our dataset are ordinal variable, it is not acceptable by the decision tree of scikit-learn.

To include the ordinal variables in our analysis, there are various methods to do it:

Ordinal Encoding

Since ordinal variables contains values in order, such as "low", "medium", and "high", the ordinal encoding captures the order of the ordinal variables by assigning corresponding integer to each value. For example, assigning 0 to "low", 1 to "medium", and 2 to "high." However, since the ordinal encoding assumes the difference between each value are the same, it is not the case of ordinal variables, which might leads to some bias.

One-hot encoding

One-hot encoding could handle both nominal and ordinal variable but replace the values with a series of dummy variables. For example, if there are 3 values in the variable "fruit", which are "apple", "banana", and "papaya", then one-hot encoding will transform the variable to 2 dummy variables, while the first represents to "apple?" and the second are "banana?". Thus, if one value get both FALSE from "apple?" and "banana?", it is "papaya." Though one-hot encoding could be used to deal with categorical variables, it is reported that it might degrade the performance of decision tree. Changing to other packages

Except sci-kit learn, there are other packages could support categorical variables to the decision trees, such as CatBoost, H2O, or LightGBM.

BUSINESS INTELLIGENCE & ANALYTICS

Say, we have four classes. So first instance, if it is an apple then it will consider it as 1, 0, 0, 0. So our next class is banana. Say if the class was banana then it will take us 0, 1, 0, 0. So that is what happens in one-hot encoding wherein it will consider the true class as the true and all other classes as false or in a binary kind of classification.

Jupyter CLASSIFICATION ASSIGNMENT NPTEL TUTORIAL Last Checkpoint: 3 hours ago (autosaved) Python 3 (ipykernel)

assigning corresponding integer to each value. For example, assigning 0 to "low", 1 to "medium", and 2 to "high." However, since the ordinal encoding assumes the difference between each value are the same, it is not the case of ordinal variables, which might leads to some bias.

One-hot encoding

One-hot encoding could handle both nominal and ordinal variable but replace the values with a series of dummy variables. For example, if there are 3 values in the variable "fruit", which are "apple", "banana", and "papaya", then one-hot encoding will transform the variable to 2 dummy variables, while the first represents to "apple?" and the second are "banana?". Thus, if one value get both FALSE from "apple?" and "banana?", it is "papaya." Though one-hot encoding could be used to deal with categorical variables, it is reported that it might degrade the performance of decision tree. Changing to other packages

Except sci-kit learn, there are other packages could support categorical variables to the decision trees, such as CatBoost, H2O, or LightGBM.

```
In [30]: mapper1 = {"vhigh" : 4, "high" : 3, "med" : 2, "low" : 1}
data["buying"] = data["buying"].replace(mapper1)
data["maintenance"] = data["maintenance"].replace(mapper1)
doors_mapper = {2 : 0, 3 : 1, 4 : 2, "smore" : 3}
data["doors"] = data["doors"].replace(doors_mapper)
person_mapper = {2 : 0, 4 : 1, "more" : 2}
data["persons"] = data["persons"].replace(person_mapper)
lug_mapper = {"small" : 0, "med" : 1, "big" : 2}
data["lug_boot"] = data["lug_boot"].replace(lug_mapper)
safety_mapper = {"low" : 0, "med" : 1, "high" : 2}
data["safety"] = data["safety"].replace(safety_mapper)
data.head(10)
```

```
Out[30]:
```

	buying	maintenance	doors	persons	lug_boot	safety	car_evaluation
0	4	4	2	2	0	0	unacc

BUSINESS INTELLIGENCE & ANALYTICS

So that is what happens in that. So here, since our categorical values have an order in them, we are going for ordinal encoding. So we will see how we are doing that. We

have a mapper function in which we are taking, so this is mapper 1 in which we are assigning 4 for very high, 3 for high, 2 for medium and 1 for low. So as we saw in the first two input features which is buying as well as maintenance, both had four categorical values. So we are using a common mapper for both in which, so see the second line, data of buying.

So data is the data frame. So in the data frame of buying, we are replacing it with mapper 1. So what it will do is we will see how it has changed the data once we have done the encoding. So the second one is maintenance. So since both had the same values, that is very high, high, medium and low was common for both buying as well as maintenance input feature we are using the same mapper for the first two. The second mapper that we are using is the doors mapper because we are using it to map the doors.

For example, if it has two doors then it is not a good car. So we are giving the value of 0. Similarly for three doors we are giving 1, four doors we are giving 2 and five or more doors, obviously it is a high end car so we are giving 3. Then the next mapper is person mapper. So if only two people can sit then the car is not that good, so we are obviously assigning it to 0. Similarly, four we are assigning 1 and four more, more than four people if they can sit, then we are assigning 2.

Then the next mapper is luggage boot space mapper, that is lug mapper in which if it is small we are assigning 0, medium 1 and big 2. So we are replacing the data frame of luggage boot as we are replacing using the lug mapper. Then the next one is safety mapper. So safety if it is low, it is not a good thing. So we are mapping it by 0 then medium as 1 and high as 2. So we are also replacing the safety data using the mapper.

Then I am just you know visualizing the data, I am just you know printing the data the first ten lines or you can even you know print the entire thing to see how our data has changed. So see whatever very high, high everything was there, everything has got mapped into numerical values. So henceforth we will be working with our algorithm using these values only because the algorithm, that is the CART algorithm inputs, you know takes inputs as numerical values.

So that is what we have done here. So only the car evaluation is now having categorical values and rest of the input features, we have changed into numerical values. So the next process that we are going to do is, we know that the output, target variable is highly imbalanced. So if we just scroll back and see the unacceptable class has almost 1200 values. So total values are 1700 in that 1200 values belong to unacceptable class and the rest of the classes they have very, very small values except you know, especially the last

two classes which are good and very good. They have extremely least number of classes that are cars that are classified as good and very good.

CLASSIFICATION ASSIGNMENT NPTEL TUTORIAL Last Checkpoint: 3 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

out[51]:

	buying	maintenance	doors	persons	lug_boot	safety	car_evaluation
0	4	4	2	2	0	0	unacc
1	4	4	2	2	0	1	unacc
2	4	4	2	2	0	2	unacc
3	4	4	2	2	1	0	unacc
4	4	4	2	2	1	1	unacc
...
1723	0	0	0	0	0	0	good
1724	0	0	0	0	0	0	unood
1725	0	0	0	0	0	0	unacc
1726	0	0	0	0	0	0	good
1727	0	0	0	0	0	0	unood

1728 rows x 7 columns

This line of code imports the SMOTE (Synthetic Minority Over-sampling Technique) class from the imblearn.over_sampling module. SMOTE is a technique used in machine learning for addressing class imbalance problems in classification tasks. It generates synthetic samples for the minority class.

BUSINESS INTELLIGENCE & ANALYTICS

So what it tends to do is it creates a bias. So before going to that, we have to balance the imbalanced dataset and for that what we are going to do is something which is known as SMOTE. So what is SMOTE? SMOTE is Synthetic Minority Oversampling Technique. So there is a library called imblearn, imbalanced learn in that we are importing the oversampling module. SMOTE is basically a technique that is used in machine learning for addressing class imbalance problems in classification task. It generates synthetic samples from the minority class to balance the class distribution, thereby preventing the model from being biased towards the majority class.

So what is the majority class here? It is unacceptable and the rest of the classes are minority classes. So unacceptable class is the majority class here and the rest of the classes which are acceptable, good and very good. They are minority classes. So what we do is, we oversample the minority classes in a way that, you know it creates many synthetic samples from the minority class and it boosts the minority class, so that the class distribution between the minority and majority class is balanced.

This line of code imports the SMOTE (Synthetic Minority Over-sampling Technique) class from the imblearn.over_sampling module. SMOTE is a technique used in machine learning for addressing class imbalance problems in classification tasks. It generates synthetic samples for the minority class to balance the class distribution, thereby preventing the model from being biased towards the majority class.

```

In [31]: from imblearn.over_sampling import SMOTE

In [32]: y=data['car_evaluation']#Assigns target variable 'car_evaluation' to the variable y.
X_train, X_test, y_train, y_test = train_test_split(data.loc[:, "buying" : "safety"], data.loc[:, "car_evaluation"], test_size =
# Apply SMOTE to balance the dataset
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_train, y_train = smote.fit_resample(X_train, y_train)#oversample the minority class in the training set (X_train, y_train).
X_test, y_test = smote.fit_resample(X_test, y_test)#oversample the minority class in the testing set

#sampling_strategy='auto' means that SMOTE will automatically determine no of synthetic samples to generate to balance class dis
# random_state=42 sets a random seed for reproducibility.

In [33]: pd.DataFrame(y_train).value_counts()
#pd.DataFrame(y_train).value_counts().plot(kind="bar")

```

BUSINESS INTELLIGENCE & ANALYTICS

All have equal number of samples. So in order to do that, what we do is from imblearn library and from the oversampling module, we will import the SMOTE function and for that, and the next line what we are going to do is, we are assigning the target variable into y. So what is the target variable? It is the car evaluation. In car evaluation, we have four classes. So we are assigning the data frame of the car evaluation class into y.

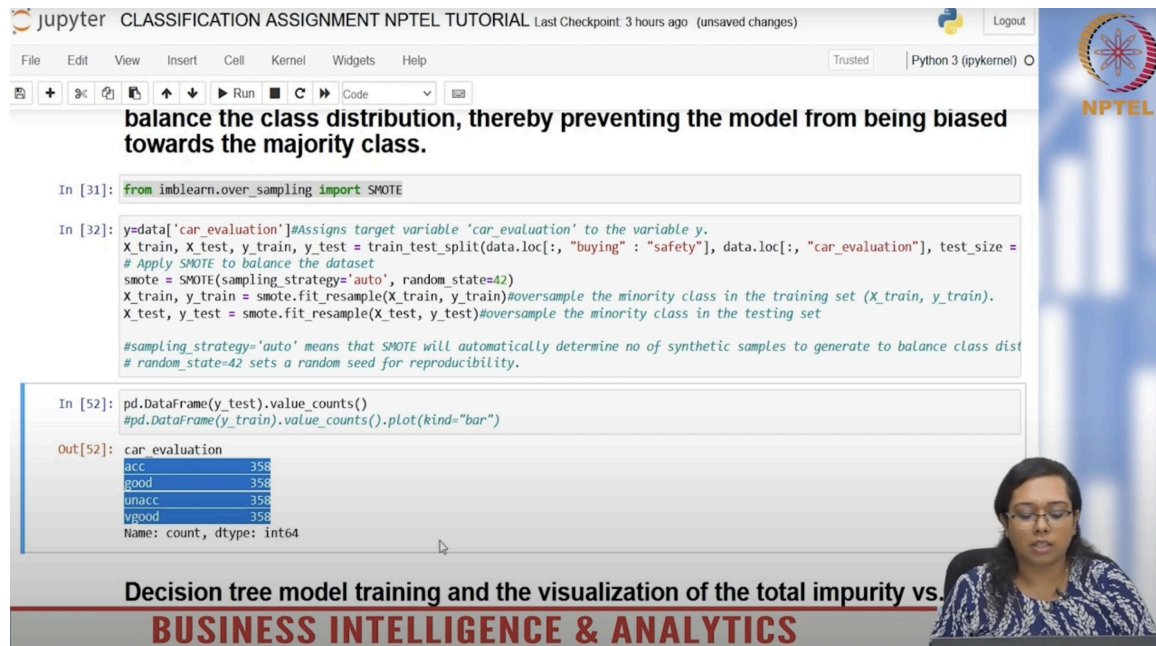
So the target variable is y and the input variable will be x. So that is basically how we are going to work, x, y split. So the next line that is x train, x test, y train and y test. This line is for test train split, you guessed it right. So how are we splitting the data? So basically, I have given data dot location.

So location wise, we are splitting. From buying to safety it is the input variable that is x and data dot location of car evaluation, that is the output variable that is y. So that is how we are splitting and test size what we are giving is 0.33, that means testing data will be 33 percent and the rest of will be used for training purpose and we are just assigning a random state, that is 42 which is usually used. We can go back and see why 42 is used often.

It has some meaning attached to it and now we are using the SMOTE. We are calling SMOTE and in sampling strategy, we are giving auto. It will just automatically assign and here also, we are giving a random state of 42 and, in x train, y train equal to SMOTE dot fit re-sample of x train, y train. So whatever x train and y train we have got after splitting the data set, we are calling the SMOTE function for that so that it will over sample the minority class in the training data set. And the second line, it can be avoided

as well.

You need not balance the test data set but I have just done it. So here we are over sampling the minority class from the test data set and then we have made it balanced. So after we have done this entire coding what it will do is, it will balance our data set. So this is just for a proof of, if we have actually balanced the data set or not. So what we are doing is, we are again printing the value counts for y train.



The screenshot shows a Jupyter Notebook window titled "CLASSIFICATION ASSIGNMENT NPTEL TUTORIAL". The code in the notebook is as follows:

```
In [31]: from imblearn.over_sampling import SMOTE

In [32]: y=data['car_evaluation']#Assigns target variable 'car_evaluation' to the variable y.
X_train, X_test, y_train, y_test = train_test_split(data.loc[:, "buying" : "safety"], data.loc[:, "car_evaluation"], test_size =
# Apply SMOTE to balance the dataset
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_train, y_train = smote.fit_resample(X_train, y_train)#oversample the minority class in the training set (X_train, y_train).
X_test, y_test = smote.fit_resample(X_test, y_test)#oversample the minority class in the testing set

#sampling_strategy='auto' means that SMOTE will automatically determine no of synthetic samples to generate to balance class dist
# random_state=42 sets a random seed for reproducibility.

In [52]: pd.DataFrame(y_test).value_counts()
#pd.DataFrame(y_train).value_counts().plot(kind="bar")

Out[52]: car_evaluation
acc      358
good     358
unacc    358
vgood    358
Name: count, dtype: int64
```

Below the code, there is a video feed of a woman speaking. At the bottom of the notebook interface, there is a red banner with the text "Decision tree model training and the visualization of the total impurity vs. BUSINESS INTELLIGENCE & ANALYTICS".

Similarly you can print for y test as well because we have done for test also. So if you run this, that is also coming as same. That is all the classes have equal values now. x test, x train, I mean y test and y train all have equal values. So we have addressed the problem of imbalance data and we are going to the next step. So the next part of the code is decision tree model training and visualization of the total impurity versus effective alpha for the training set.

Now we are going towards decision tree classification and for that, first line that is `clf = DecisionTreeClassifier(criterion='gini')` to decision tree classifier within brackets criterion equal to Gini. So we are using the Gini coefficient. You know what is the Gini coefficient and how to calculate for Gini of the split and the total Gini and how to calculate this and CCP alpha. Computing the CCP alpha can be an individual thing. You can take it as 0 or you can adjust the CCP alpha according to which the impurity of the final tree is becoming less.

So you have to adjust the alpha value accordingly and I have just given the minimum sample split size as 300. So basically what it does is, this line does is it uses the Gini

impurity as the criterion for splitting the nodes in our tree and we have, initially we have not pruned the tree because we do not know which alpha is suitable for pruning. So we have just set it to 0, so that initially no pruning is done. So in this model, what we are doing is cost complexity pruning which also you have studied in the class. So it is basically a post-pruning technique in which, you know you allow the tree to grow and then you go and cut the leaves.

So that is kind of a post-pruning. If you apply a particular kind of, you know rule to pre-prune the tree then those are pre-pruning techniques and we are not going to do that in our model today. So I have just, you know minimum number of samples required to split an internal node. I have just set it to 300. So the next one is, next line that you can see from the code is to calculate the cost complexity pruning path for the decision tree using our training data which is x train and y train. That is what we are assigning a cost complexity path, CCP path in this and we are giving the input as x train and y train for that.

The screenshot shows a Jupyter Notebook window titled "CLASSIFICATION ASSIGNMENT NPTEL TUTORIAL". The code in the cell is as follows:

```
In [53]: clf = DecisionTreeClassifier(criterion = "gini", ccp_alpha=0,min_samples_split=300)
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
fig, ax = plt.subplots()
ax.plot(ccp_alphas[:-1],impurities[:-1],marker='o', drawstyle="steps-post")
#blue lines drawn to highlight step changes; orange is the graph
ax.set_xlabel("Effective alpha")
ax.set_ylabel("Total impurity of leaves")
ax.set_title("Total Impurity vs effective alpha for training set")
plt.plot(ccp_alphas, impurities)
# trains a decision tree classifier, calculates the cost-complexity pruning path,
# and visualizes the relationship between the total impurity of tree leaves and
# the effective alpha (complexity parameter) for the training set.
```

The output shows a plot titled "Total Impurity vs effective alpha for training set". The plot displays a series of blue lines representing step changes in total impurity as the effective alpha increases. An orange line is also visible, representing the total impurity of the leaves. The y-axis is labeled "Total impurity of leaves" and has a tick mark at 0.7. The x-axis is labeled "Effective alpha".

BUSINESS INTELLIGENCE & ANALYTICS

Then the next couple of code are just for, you know having plot of total impurity of leaves versus effective alpha graph. So that is for that only. So the third line and that is CCP alphas, impurities equal to path dot CCP alphas, path impurities, that is it extracts the calculated alpha values or the cost complexity parameters and the corresponding impurity values from the cost complexity path that we have just initialized before this. So that line stands for that. Then we are creating a matplotlib figure and we are creating an axis for plotting, that is it plots the total number of impurities of leaves versus effective alpha values and it will, you know also highlight the step changes.

That is why the draw style has been set to steps post. So it will also show you how much change is there with every increasing step. So we have set the labels for the axis, X axis and Y axis and then we are going to plot it using the PLT dot plot within bracket CCP alphas versus impurities. And this is the graph that we have got and from this you will be able to see what kind of alpha you have to assign, so that you know if you assign somewhere around 0.1, then the see the increase in impurities it is a drastic increase in impurities.

So the alpha value has to be less. Even to get a better picture, we are going to print the alpha values and the increase in impurities as we, you know change the alpha value that is what we are going to do in the next piece of code. So as the effective alpha value increases, it corresponds to increasing complexity in the decision tree. So an increasing graph indicates a risk of overfitting. Overfitting occurs when a model captures noise.

The screenshot shows a Jupyter Notebook interface. At the top, the title bar reads "jupyter CLASSIFICATION ASSIGNMENT NPTEL TUTORIAL Last Checkpoint: 3 hours ago (unsaved changes)". The notebook contains a plot and a code cell. The plot shows "Effective alpha" on the x-axis (ranging from 0.00 to 0.10) and an unlabeled y-axis (ranging from 0.2 to 0.3). The data points are connected by a blue line with a step-like pattern, showing a sharp increase in the y-value as the x-value approaches 0.1. The code cell contains the following Python code:

```
In [35]: #As the effective alpha increases, it corresponds to increasing complexity in the decision tree.
#an increasing graph indicates a risk of overfitting. Overfitting occurs when a model captures noise
#or random fluctuations in the training data rather than the underlying patterns
#Since there is only n-1 elements in next, add another element "1" to get the same size as others

next = np.diff(impurities)
next = np.append(next, 1)
alpha_choice = pd.DataFrame(ccp_alphas, columns = ["ccp_alphas"])
alpha_choice["impurities"] = impurities
alpha_choice["Increase in impurity"] = next
alpha_choice
```

The output of the code cell is a DataFrame with the following data:

	ccp_alphas	impurities	Increase in impurity
0	0.000000	0.206728	0.000027
1	0.000027	0.206755	0.000083
2	0.000083	0.206838	0.000342
3	0.000342	0.207180	0.001133

At the bottom of the notebook, there is a red banner with the text "BUSINESS INTELLIGENCE & ANALYTICS". A small video inset of a woman is visible in the bottom right corner of the notebook area.

So you know, random noises are accommodated in our model, then it tends to overfit, and random fluctuations in the training data rather than the underlying patterns. So that explains it and that is our next code and since there is only n - 1 elements in the next, we add another element 1 to get the same size. So this is nothing but, you know we are just printing the increase in impurity using this. So in the first line, that is next = np.diff(impurities), what it does is it computes the difference between consecutive elements in the impurities array. So there is an array called impurities and it just computes the difference between the consecutive values and it just calculates and shows you how much there is an increase in impurity between each set of alpha values.

So the next line is, `next = np.append(next,1)`. So it basically appends a value of 1 to the end of the next array. So this is basically done to represent the impurity increase for the, you know largest alpha value that is 1. And next is `alpha_choice = pd.DataFrame (CCP_ alphas , columns = [CCP_ alphas])`. So in this what we do is, we are creating a pandas data frame `pd.DataFrame` called `alpha_choice` where we have a column named `CCP alphas` containing all the alpha values which we get from the cost complexity pruning path.

alpha_choice

```
Out[35]:
```

	ccp_alphas	impurities	Increase in impurity
0	0.000000	0.206728	0.000027
1	0.000027	0.206755	0.000083
2	0.000083	0.206838	0.000342
3	0.000342	0.207180	0.001133
4	0.000378	0.208314	0.002688
5	0.002688	0.210982	0.003088
6	0.003088	0.214069	0.004263
7	0.004263	0.218333	0.004748
8	0.004748	0.223081	0.006388
9	0.006388	0.229469	0.012769
10	0.012769	0.242238	0.027570
11	0.027570	0.269807	0.029136
12	0.029136	0.298943	0.029333
13	0.029333	0.328276	0.057746
14	0.057746	0.386022	0.262966
15	0.087655	0.648888	0.101012
16	0.101012	0.750000	1.000000

BUSINESS INTELLIGENCE & ANALYTICS

And the next we are just defining what are, you know our columns. So we are just adding the impurities column, we are adding the increase in impurities column and we are just printing everything together. So this is how it looks and we have the serial number over here, then the CCP alpha starting from 0 and it extends till 0.1010 and we have the impurity. So if you just scroll, you can see how the impurity increases initially, that is it starts with 0.206 and it stays like that, then you know around the sixth row we get 0.21, sorry fifth row and then it increases to 0.22. Then there is a sudden kind of increase when you come to this tenth, eleventh and twelfth line, there is it is becoming 0.24 then 0.26 then suddenly 0.29 and then immediately 0.32 then 0.38 and if you see around 14th line there is a sudden increase in the impurity from 0.38 to 0.64. So what it tells is that, you know you need not make your model complicated by taking an alpha as largest 0.05 or 0.08 because there is a huge increase in the impurities. So the increase in impurities, if you see the third column you can see that the impurity increases by 0.26 over here, rest everything was in 0.00 something but here it is 0.26 and then 0.1 and then

1 in the last one because it is the last alpha value that we are considering. So based on this, you can decide on what alpha value to give, it is a, you know you can just keep changing the alpha value to see how your tree grows and you know gets pruned.