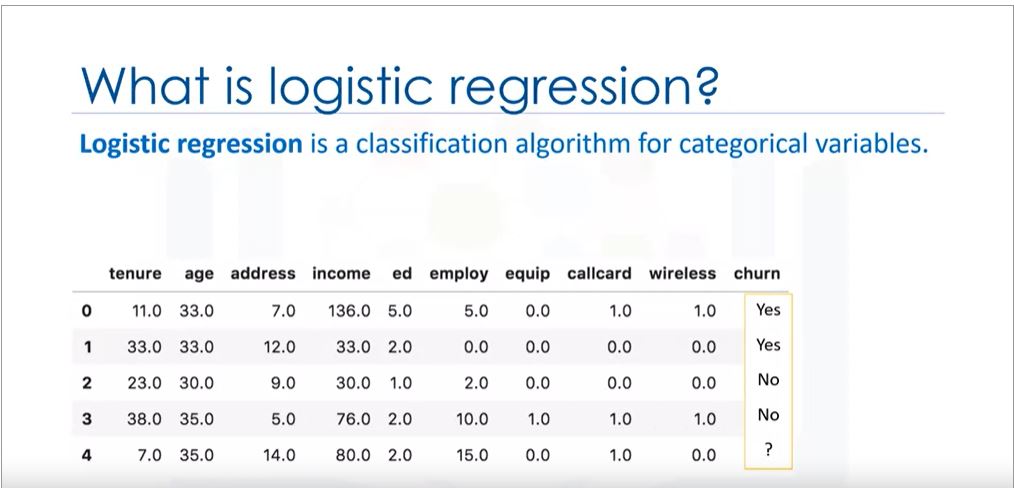
**Logistic Regression:**

Logistic regression is a statistical and machine learning technique for classifying the records of the dataset based on the input fields. It’s a classification algorithm for categorical variables.

Suppose you have the dataset of the customers of the telecom company. Now the target variable is churn i.e. whether the customer is continuing the recharge or leaving the product of the company. By analysing you can find the reasons why customers leave and predict whether the future customers will leave or not.



We can use the logistic regression model to predict the dependent variable churn with the help of independent variables tenure , age ,address , income , employ , etc. Linear regression is analogous to the logistic regression. In linear regression we use to predict the outcome as the continuous variable such as price of the house etc. But in logistic regression we predict he output as 1 or 0(Yes or No).

Logistic regression can be used for both binary and multi-class classification.

Logistic regression application :

-Predict probability of person having heart attack

-predict the mortality in injured patients

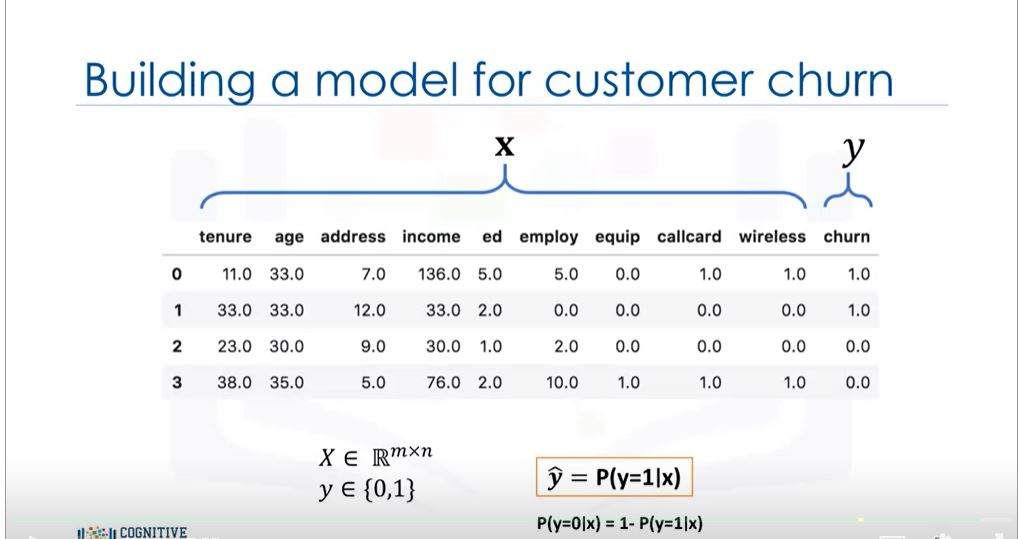
-predict whether customer will buy product or not

-predict whether product will fail or get success

-etc.

When should we use logistic regression?

1. If your data is binary-0/1,Yes/No, True/false
2. If you need probabilistic results
3. When you need a linear decision boundary
4. if you need to understand the impact of a feature



**Logistic regression VS Linear regression :**

So again we take the above example. Now the main aim of logistic regression is to obtain the class of the churn i.e. YES or NO and also to calculate the probabilities of the class type. Now we map all variables into continuous ones. Now can we use linear regression to get the prediction?

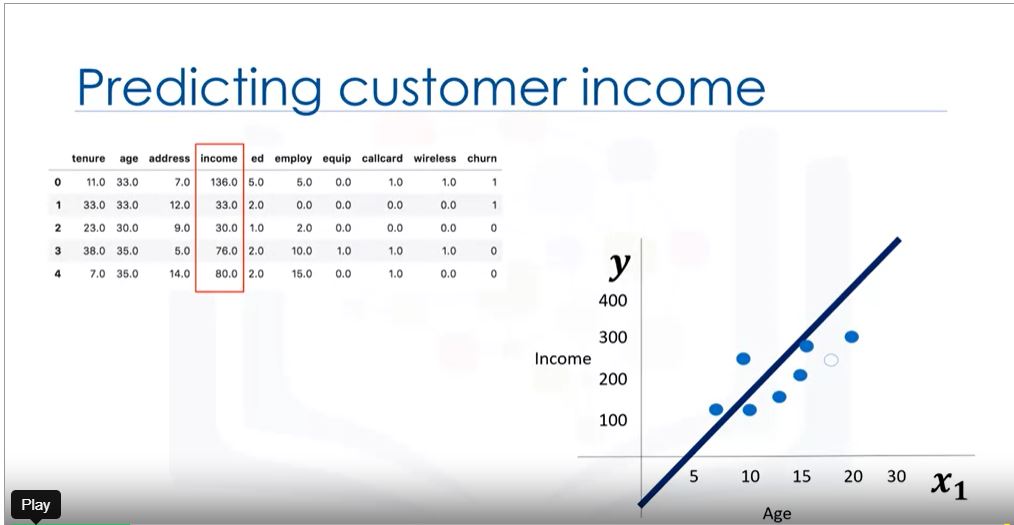
Now for a moment forget the prediction of the churn. Here we want to predict the income of the customer i.e. the continuous value rather than calculating the categorical value.

SO we can take a dependent variable income and the independent variable age for sake of simplicity.

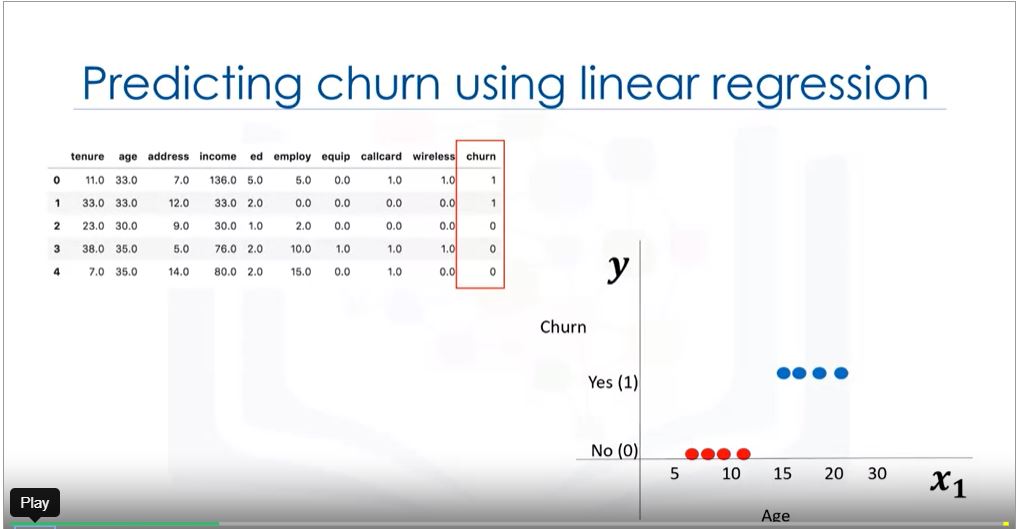
We can plot age vs income.

Now with the linear regression we can form a line or linear polynomial through the data by training the data. Suppose eqn of line is a+bx1.

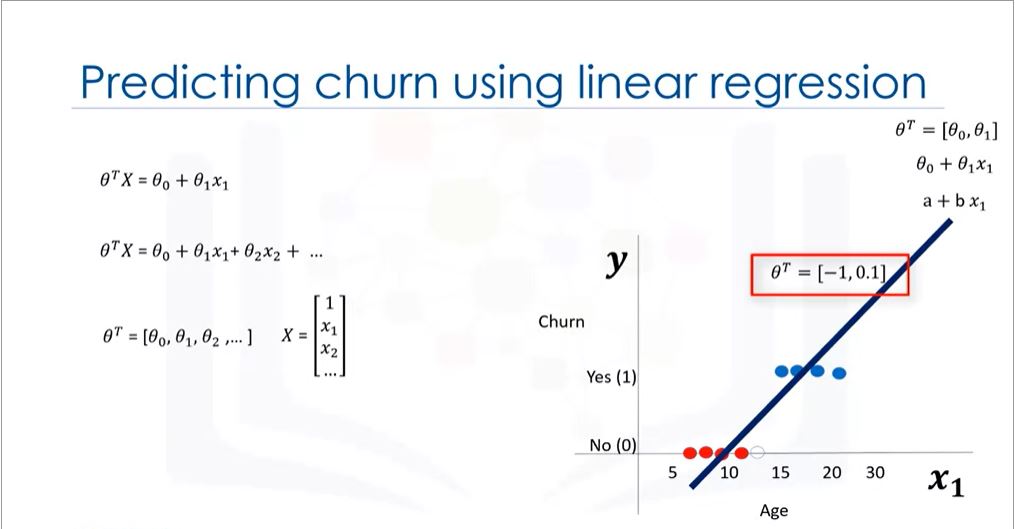
Here x1 is age and hence we can use the eqn of line to predict the value of y i.e. income.



But now if we want to predict the categorical value i.e. churn. Can we use the same technique?

If we try to do so we take age as the independent variable. And plot age vs churn graph.Graphically the plot is scattered and value of y has only two possibilities i.e. YES or NO. 

If we use linear regreesion,then line is plotted in the graph. The line has eqn a+bx1, or all the other formulas given in the image.



Theta is the weighted confidence and X is set values of customer.

if a=-1 and b=0.1

then eqn=-1 + 0.1 x1

if age is p1[13]

p1=-1 + 0.1\*13=0.3

So we got a number , hence we defined a threshold in the model that if the value is grater than 0.5 then churn value will be 1 or else it will be 0.

So here p1 belongs to class 0.

But this will not return the probability of p1 in class 0.

Hence in linear regression the problem is we get output as continuous values i.e 3 or -2. So now we have to decide a threshold and then we have to assign the class. So it produces a step function. Suppose 1 is the threshold then the output 1 and output with value 100 will come in same class. Hence it would be better if we have smooth line to predict and can return the probability of falling class as well .

So solution is we use the sigmoid function which not directly return the value of Y but return the probability of falling the case between 0 and 1.

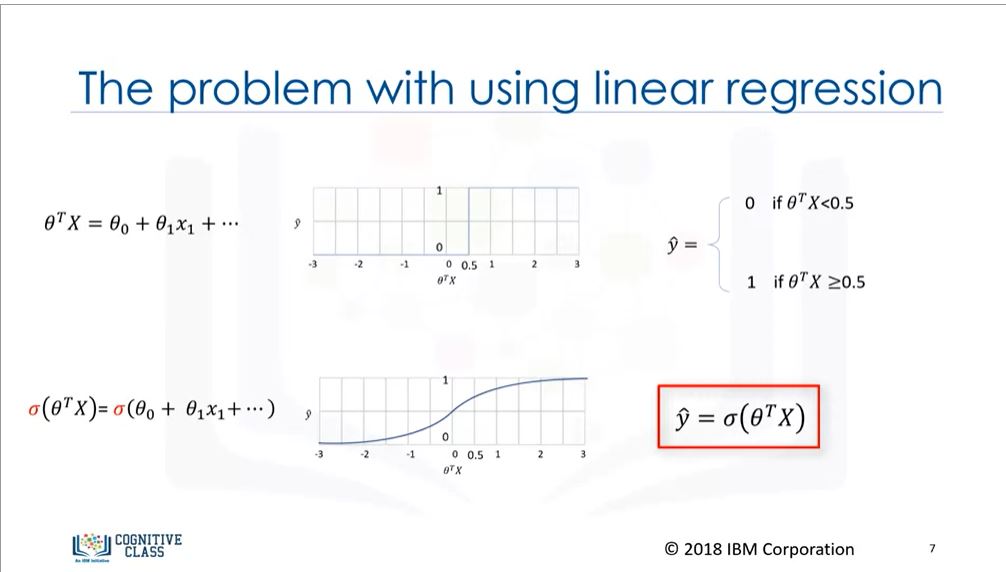
Now what is sigmoid function.

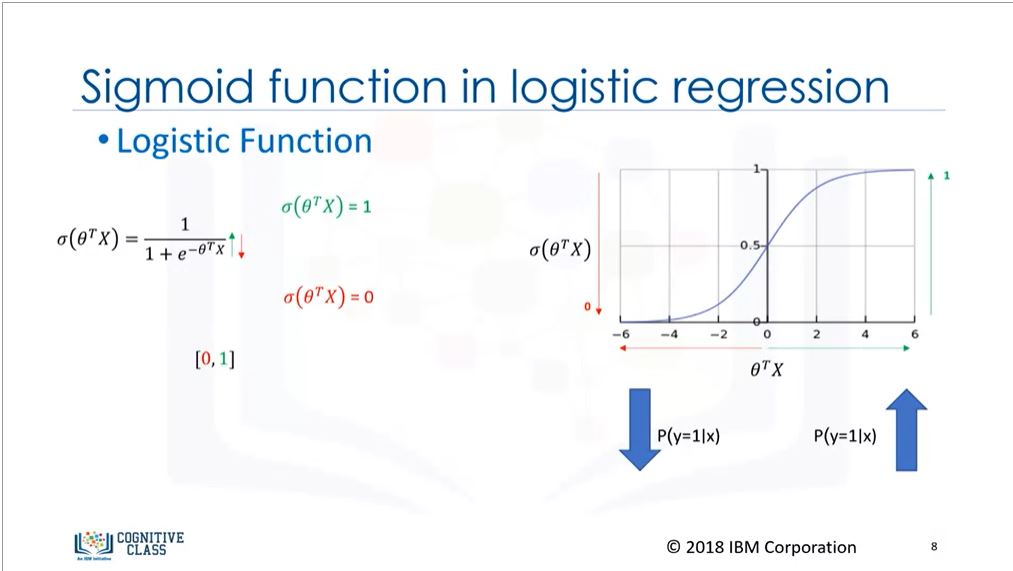
-Its an logistic function

Now here in image red line shows \theta^t \* X is low value , hence sigmoid will near to 0.

Now here in image green line shows \theta^t \* X is high value , hence sigmoid will near to 1.

So its value will be between 0 and 1.



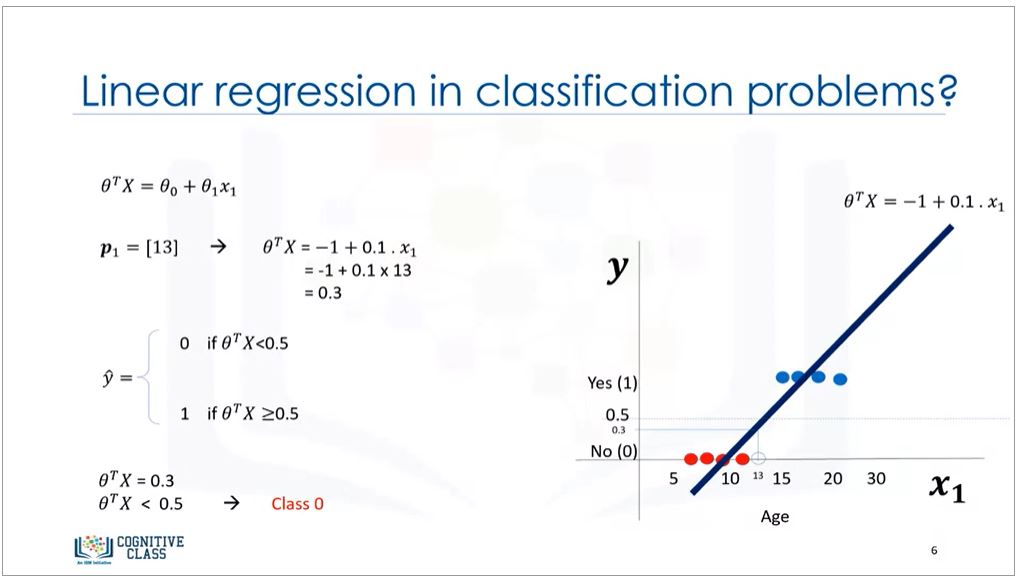


So now how to decide the output of our model by getting the value of sigmoid function?

* Found P(Y=1|X)
* We can find P(Y=0|X) = 1- P(Y=1|X)

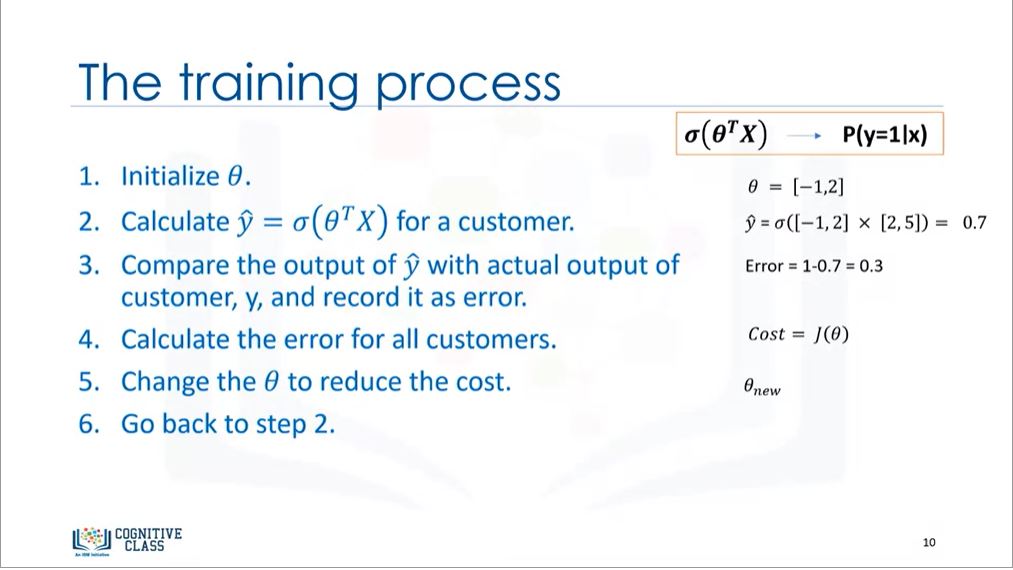
Suppose P(Y=1|X)=0.8

So P(Y=0|X)=1-0.8=0.2



Training process.

1. Initialize \theta
2. Calculate yhat
3. Compare yhat with actual label of customer and record the difference
4. calculate error of all customer.
5. Change \theta to reduce cost
6. Go to step 2



**Logistic regression training:**

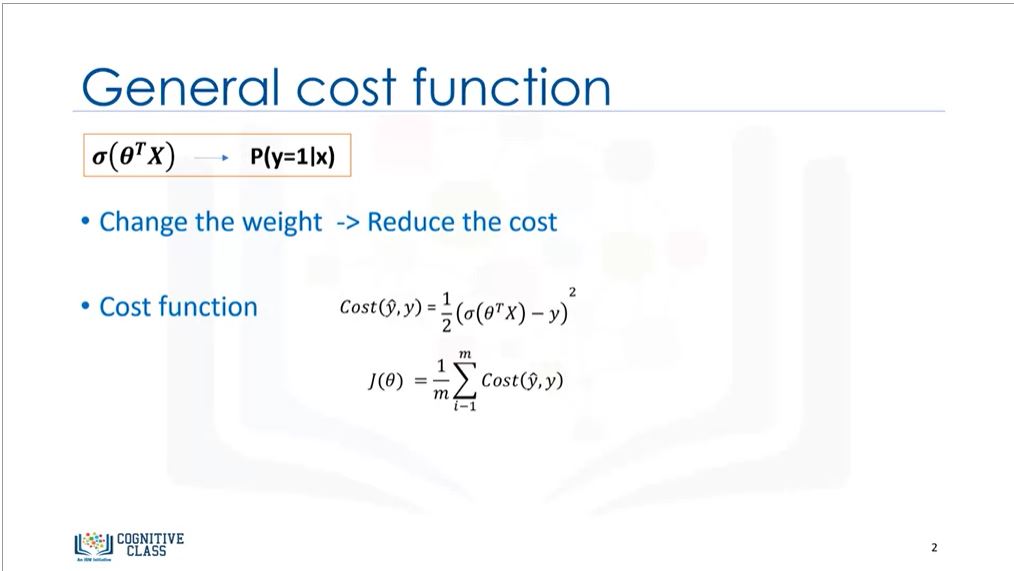
Now we have to change the theta to reduce the cost.

To calculate cost function we can use direct libraries in python.

But formula is Cost(yhat,y)= siqmoid(theta^t X) – y

But as there is negative values we square it and then half it.

Then add all the cost to find the total cost.

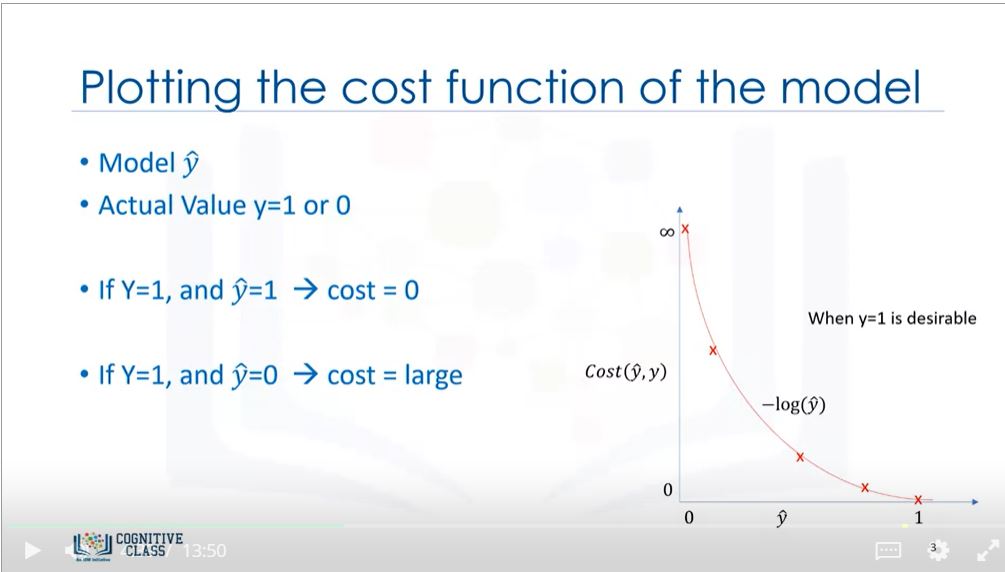


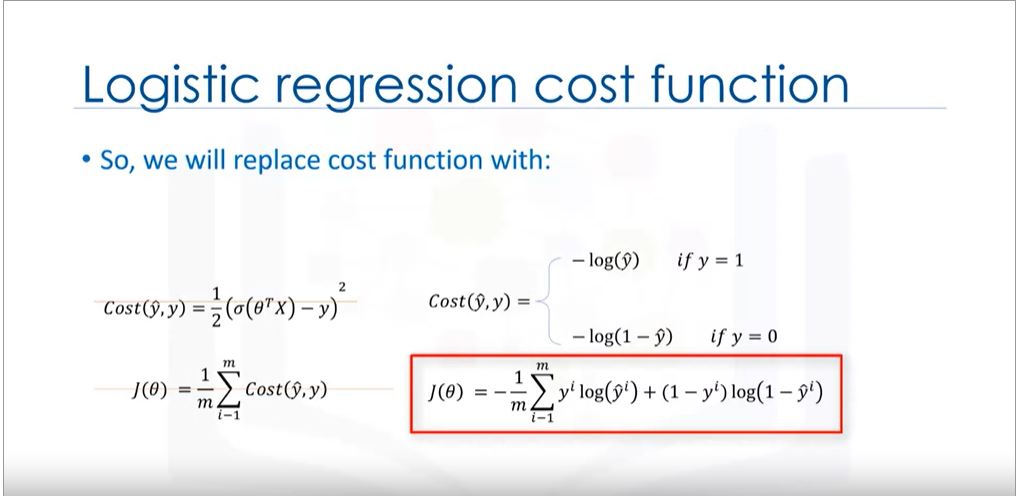
Now how to find the weight with small cost.

But with this formula its difficult to find the global minimum to get the thetha. Hence we change the formula of cost function.

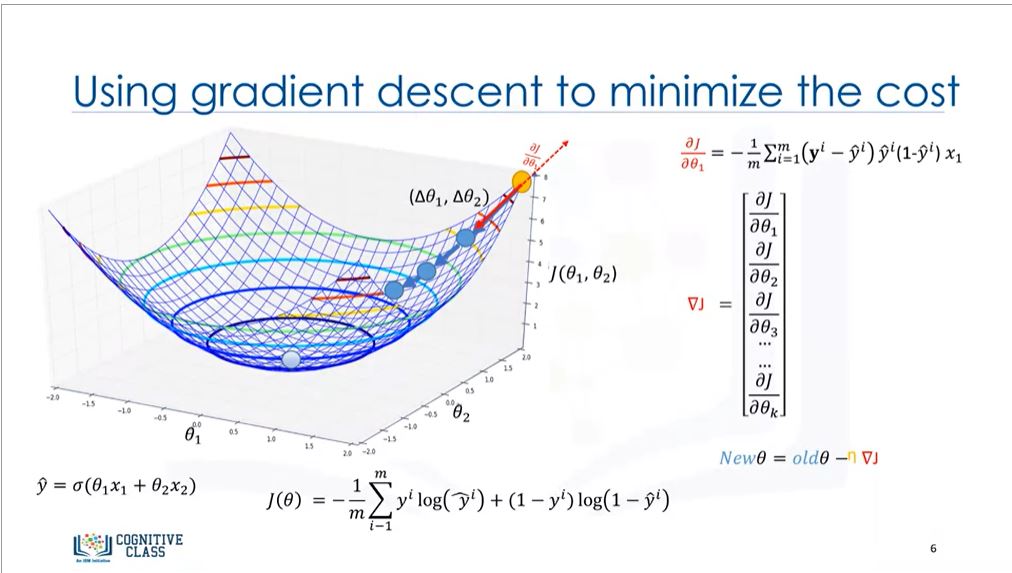
So we will plot cost vs yhat graph.

So it plots graph as cost=-log(yhat)



**Minimizing the cost function:**

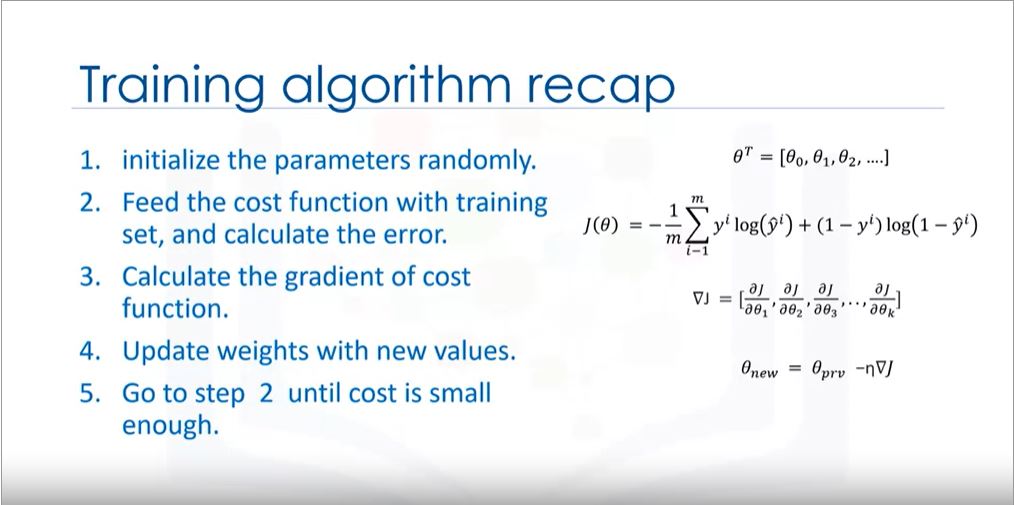
We can use gradient descent to reduce the cost function. A gradient descent is a techniwue to use the derivative of a cost function to change the parameter values , in order to minimize the cost.



Gradient Descent:

Now we plot the graph of theta1 vs theta 2 and add parameter J(theta1,theta2) and get a error ball.Now which point in the graph gives minimum cost ? So you have to select a random point in the ball . Lets say yellow ball is the starting point. Now we change it to blue ball by changing the distances by delta theta1 and delta theta2.No we can go further. The steeper the slope the further we can get. As we approach the lowest point slope diminishes and when we get to the minimum point of the curve . But we don’t know how to take this steps and in which direction. So here gradient comes in light . Gradient is slope which gives measures of step. And direction of gradient says in which direction we have to proceed.

More the slope, higher the gradient, higher the step. Lower the slope, lower the step.



**CODING:**

Now here we take the same example as above. We have a telecommunication company and we are predicting the churn of customers for future with given dataset.

#Importing all the libraries

import numpy as np

import pandas as pd

import scipy.optimize as opt

from sklearn import preprocessing

%matplotlib inline

import matplotlib.pyplot as plt

#About the dataset

The dataset includes information about:

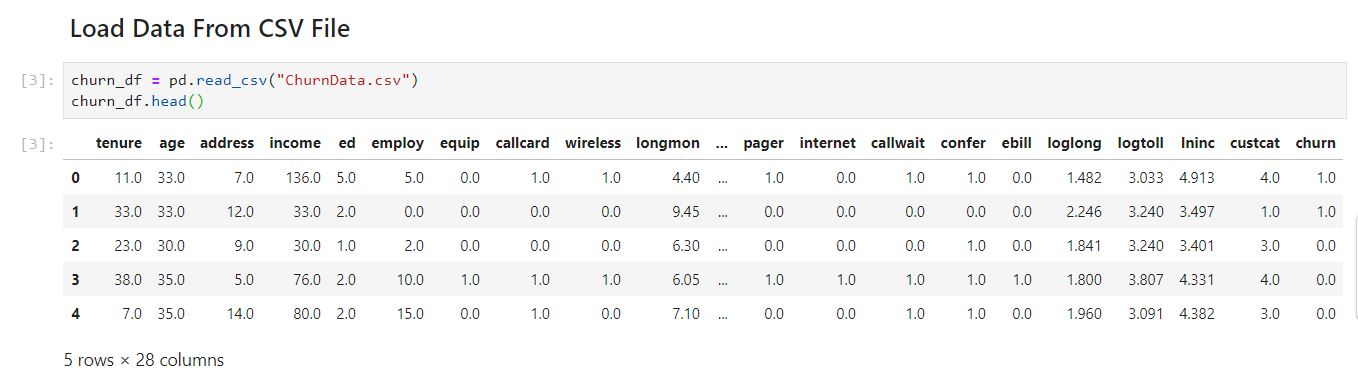
* Customers who left within the last month – the column is called Churn
* Services that each customer has signed up for – phone, multiple lines, internet, online security, online backup, device protection, tech support, and streaming TV and movies
* Customer account information – how long they had been a customer, contract, payment method, paperless billing, monthly charges, and total charges
* Demographic info about customers – gender, age range, and if they have partners and dependents

!wget -O ChurnData.csv <https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/ChurnData.csv>

#load the csv file

churn\_df=pd.read\_csv(‘ChurnData.csv’)

churn\_df.head()



#Now converting float value of churn to int using preprocessing library

churn\_df = churn\_df[['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equip', 'callcard', 'wireless','churn']]

churn\_df[‘churn’]= churn\_df[‘churn’].astype(int)

churn\_df.head()

#Now we define X and Y from the dataset

X=np.asarray(churn\_df[['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equip']])

Y=npasarray(churn\_df[[‘churn’]])

from sklearn import preprocessing

X = preprocessing.StandardScaler().fit(X).transform(X)

X[0:5]

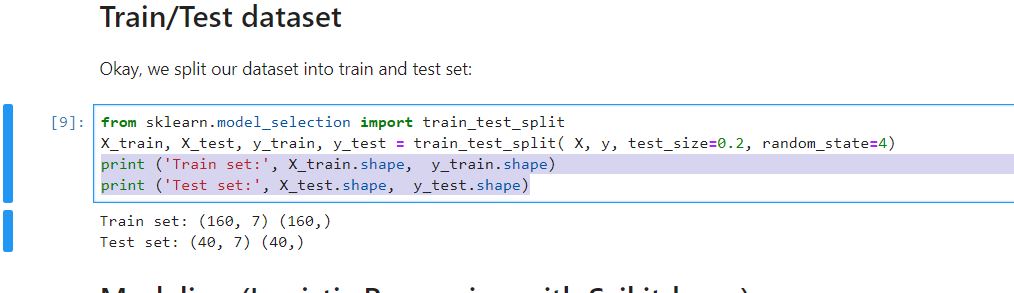
#Train-test split

from sklearn.nodel\_selection import train\_test\_split

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.4,random\_state=4)

print ('Train set:', X\_train.shape, y\_train.shape)

print ('Test set:', X\_test.shape, y\_test.shape)



#Now we train the model. For that we also have to see the regularization which is commonly used in ml to solve the problem of overfitting. C is the inverse of regularization strength.Smaller the C higher the regularization strength.

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix

lr=LogisticRegression(C=0.01,solver=’liblinear’).fit(X\_train,Y\_train)

yhat=lr.predict(X\_test)

yhat

yhat\_Prob=lr.predict\_proba(X\_test)

yhat\_Prob

**Evaluation**

**jaccard index**

Lets try jaccard index for accuracy evaluation. we can define jaccard as the size of the intersection divided by the size of the union of two label sets. If the entire set of predicted labels for a sample strictly match with the true set of labels, then the subset accuracy is 1.0; otherwise it is 0.0.

from sklearn.metrics import jaccard\_similarity\_score

jaccard\_similarity\_score(y\_test, yhat)

**confusion matrix**

Another way of looking at accuracy of classifier is to look at **confusion matrix**.

from sklearn.metrics import classification\_report, confusion\_matrix

import itertools

def plot\_confusion\_matrix(cm, classes,

normalize=False,

title='Confusion matrix',

cmap=plt.cm.Blues):

"""

This function prints and plots the confusion matrix.

Normalization can be applied by setting `normalize=True`.

"""

if normalize:

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

print("Normalized confusion matrix")

else:

print('Confusion matrix, without normalization')

print(cm)

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

fmt = '.2f' if normalize else 'd'

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, format(cm[i, j], fmt),

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label')

print(confusion\_matrix(y\_test, yhat, labels=[1,0]))

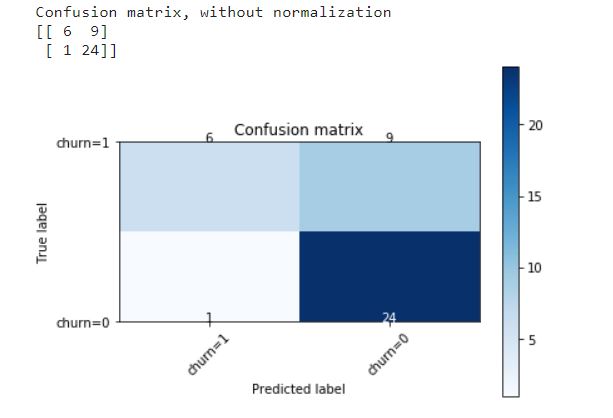
# Compute confusion matrix

cnf\_matrix = confusion\_matrix(y\_test, yhat, labels=[1,0])

np.set\_printoptions(precision=2)

# Plot non-normalized confusion matrix

plt.figure()

plot\_confusion\_matrix(cnf\_matrix, classes=['churn=1','churn=0'],normalize= False, title='Confusion matrix')

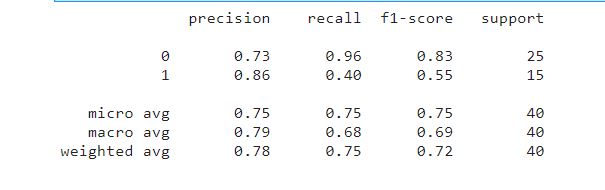
Look at first row. The first row is for customers whose actual churn value in test set is 1. As you can calculate, out of 40 customers, the churn value of 15 of them is 1. And out of these 15, the classifier correctly predicted 6 of them as 1, and 9 of them as 0.

It means, for 6 customers, the actual churn value were 1 in test set, and classifier also correctly predicted those as 1. However, while the actual label of 9 customers were 1, the classifier predicted those as 0, which is not very good. We can consider it as error of the model for first row.

What about the customers with churn value 0? Lets look at the second row. It looks like there were 25 customers whom their churn value were 0.

The classifier correctly predicted 24 of them as 0, and one of them wrongly as 1. So, it has done a good job in predicting the customers with churn value 0. A good thing about confusion matrix is that shows the model’s ability to correctly predict or separate the classes. In specific case of binary classifier, such as this example, we can interpret these numbers as the count of true positives, false positives, true negatives, and false negatives.

print (classification\_report(y\_test, yhat))



Based on the count of each section, we can calculate precision and recall of each label:

* **Precision** is a measure of the accuracy provided that a class label has been predicted. It is defined by: precision = TP / (TP + FP)
* **Recall** is true positive rate. It is defined as: Recall =  TP / (TP + FN)

So, we can calculate precision and recall of each class.

**F1 score:** Now we are in the position to calculate the F1 scores for each label based on the precision and recall of that label.

The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0. It is a good way to show that a classifer has a good value for both recall and precision.

And finally, we can tell the average accuracy for this classifier is the average of the F1-score for both labels, which is 0.72 in our case.

### log loss

Now, lets try **log loss** for evaluation. In logistic regression, the output can be the probability of customer churn is yes (or equals to 1). This probability is a value between 0 and 1. Log loss( Logarithmic loss) measures the performance of a classifier where the predicted output is a probability value between 0 and 1

from sklearn.metrics import log\_loss

log\_loss(y\_test, yhat\_prob)

0.6017092478101185