**8.Machine Learning:**

Types of machine learning:

1. Supervised Learning
2. Unsupervised Learning
3. Semisupervised Learning

**1.Supervised Learning:**

* It will be traines with labelled data
* Takes a feature for every variable
* Feature 1 is for variable1 and so on
* Output is any one of the label

**Ex:**

Assume 3 coins with different 1 ruppe,euro,dirum

1st feature value is 1 rupee

2nd feature value is 1 euro

3rd feature value is 1 dirum

- weight==feature

- currency = label (3grams=1 ruppee)

when you provide a new coin to the machine it will look for the feature value which matches feature value it will give that label as output

**Supervised Learning Algorithms:**

1. linear regression
2. logistic regression
3. decision tree
4. Random forest etc..

**1.Linear Regression:**

* Linear regression is also a type of machine learning-algorithm
* Supervised machine-learning algorithm
* Learns from the labelled datasets and maps the data points to the most optimized linear function
* These points can be used for prediction on new datasets.

Linear Regression line is y=a+bx

**Dependent and independent Variable:**

* Independent: The independent variable is the cause.its value is independent of other variables in your study.
* Dependent: The independent variable is the effect. Its value depends on changes in the independent variable.

**Case Study:**

Consider measurements of a chemical reaction: The mass of the product increases with time.

The observations are:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Time (m) | 5 (f1) | 7 | 12 | 16 | 20 |
| Mass (gms) | 40 (l1) | 120 | 180 | 210 | 240 |

* Time- Independent
* Mass-Dependent

According to case study we have 5 values for features

5 labels [1 for each feature]

**Calculation:**

Find the mean of dependent and independent variable?

**Code:**

from sklearn.linear\_model import LinearRegression

LR=LinearRegression()

t=[[5],[7],[12],[16],[20]]   #time

m=[40,120,180,210,240]       #mass

LR.fit(t,m)

LR.predict([[25]])     #2D

**Output:**

array([316.7012987])

**2.Logistic Regression**:

* Used for binary classification
* Supervised learning algorithm

**Binary Classification:**

**Ex:**

**segregating food:**

Y=0, class A-healthy food

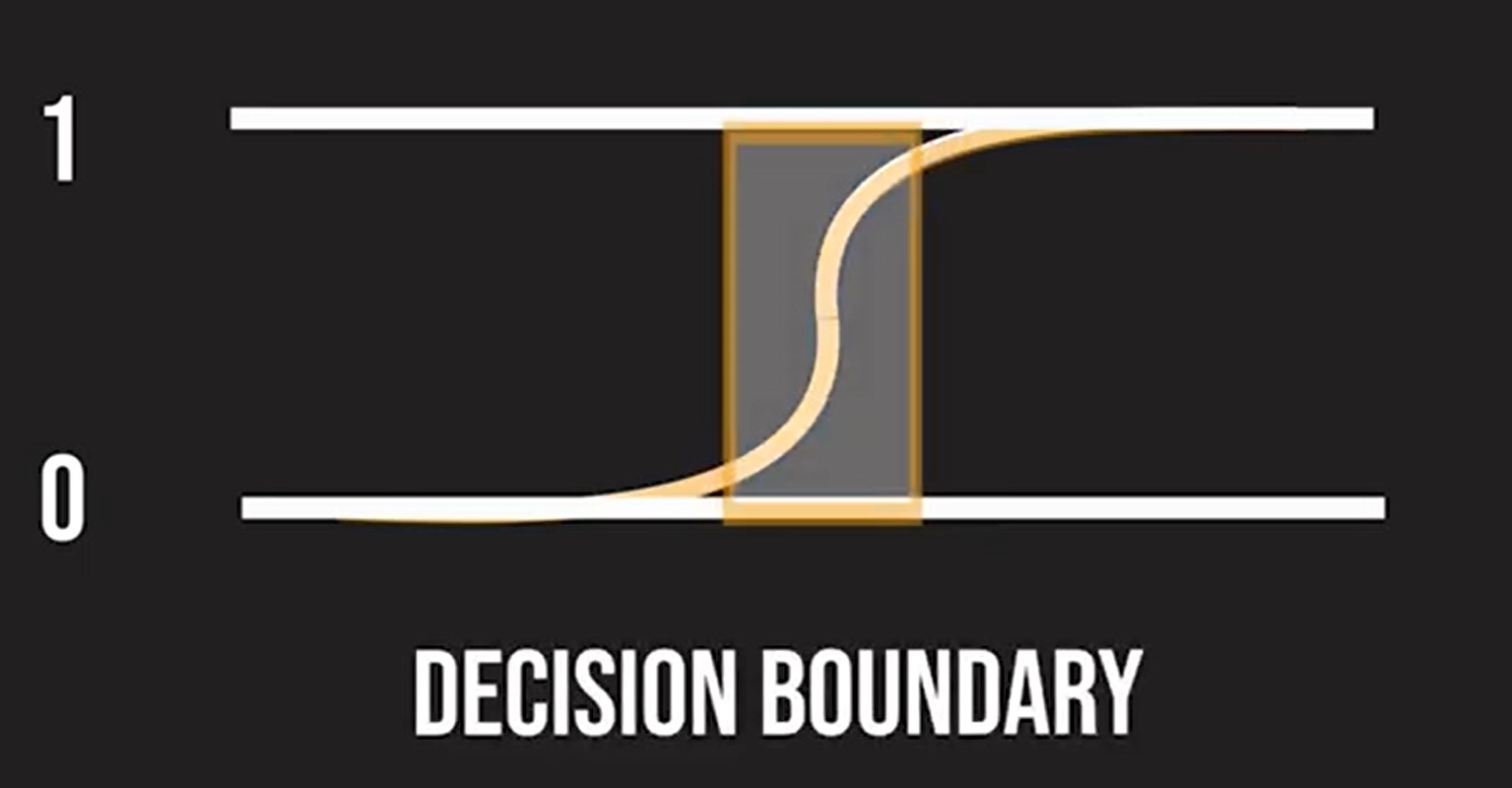
Y=1, classB-unhealthy food

**Case Study:**

* Letus take data from football matches:
* Based on the distance between player and goal post, we are going to predict whether it is a goal or not!
* Let us plot some trails now
* When distance=2m, goal is scored, Y=1
* When distance=4m, goal is scored, Y=1
* For 5,7,10,20,22 m, it is always a goal, Y=1
* When distance = 23m, for 15 trails,few are goals Y=1, few are failures Y=0

**Sigmoid Function:**

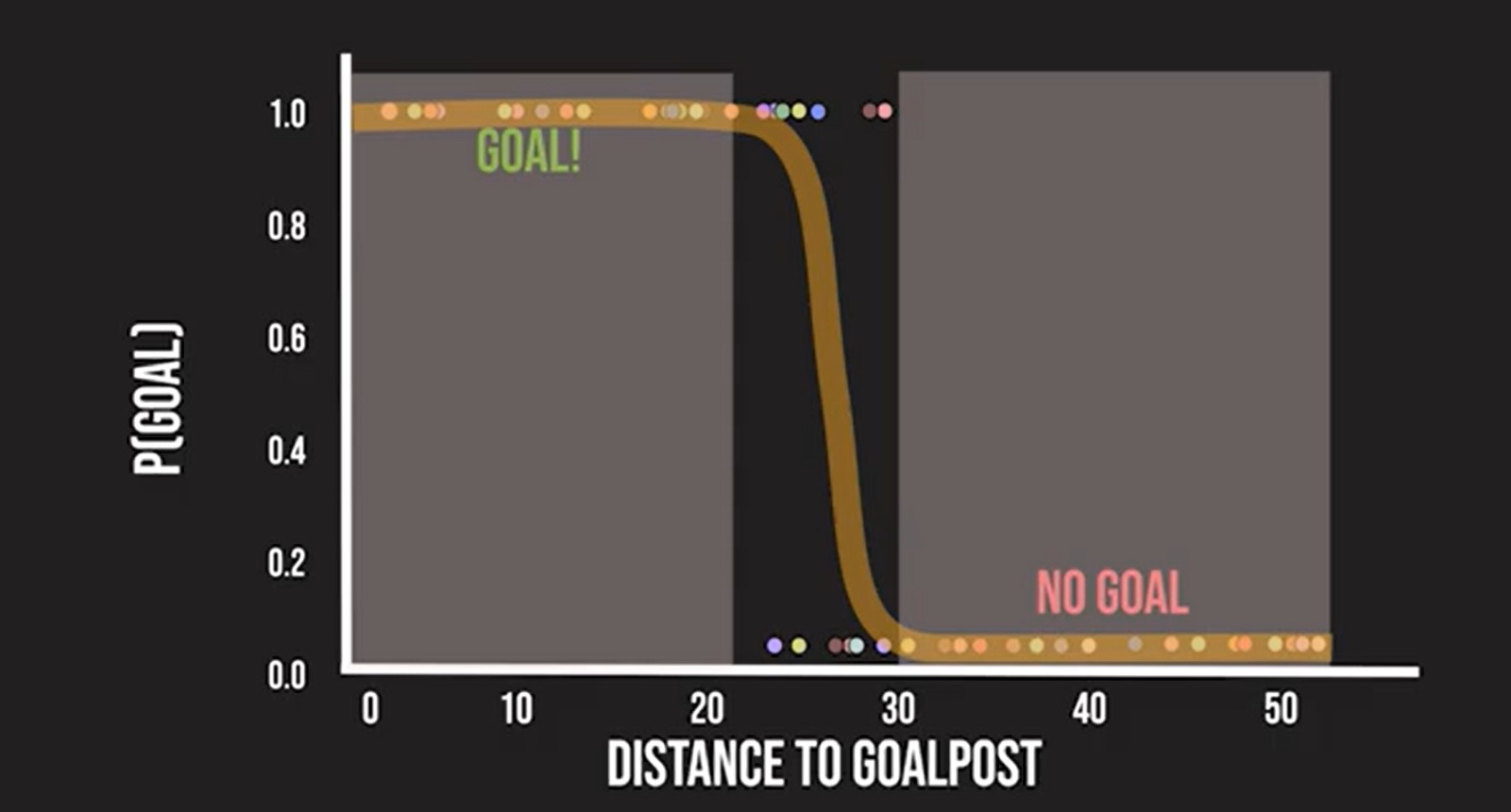
* Sigmoid function is a S-shaped function with peak at 1 and 0 at valley.
* When model is very confident, Narrow Decision Boundary
* When model is very not confident, Wide Decision Boundary



**Ex:**

After Plotting distance vs goals bwtween 20 and 30m, the probability reduces from 1 to 0

But logistic regression accepts only 2 classes.

So, A threshold variable(0.5) is set

The Model Prediction is:

* P>0.5, it’s a **Goal!** -Class A(Y=1)
* P=<0.5, it’s a **Miss!** -Class B(Y=0)



**Code:**

import numpy as np

from sklearn.linear\_model import LogisticRegression

# Distance and corresponding probability data

distances = np.array([1,2,5,10,15,20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30,35,40,41,47,50]).reshape(-1, 1)

probabilities = np.array([1,1,1,1,1,1,0.9, 0.85, 0.73, 0.67, 0.5, 0.47, 0.39, 0.31, 0.25, 0.15,0,0,0,0,0])

#convert probability to binary labels

threshold=0.5

binary\_labels=(probabilities > threshold).astype(int)

#create and fit logistic regression model

logr=logosticRegression()

logr.fit(distances,binary\_labels)

p=logr.predict([[10]])

print(p)

if(p==1):

  print('goal')

else:

  print('miss')

**Output:**

[1]

goal

**Ex-2:**

**#predict 100 distances from 1-50**

**#generate distances for prediction**

dist=np.linspace(1,50,100).reshape(-1,1)

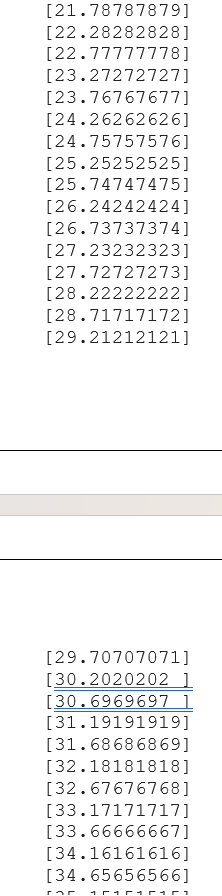
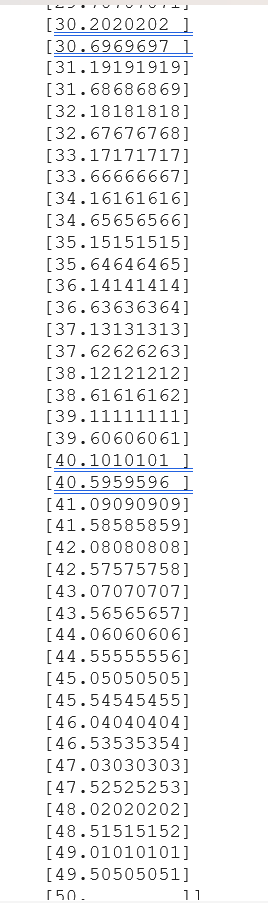
print(dist) **#distances**

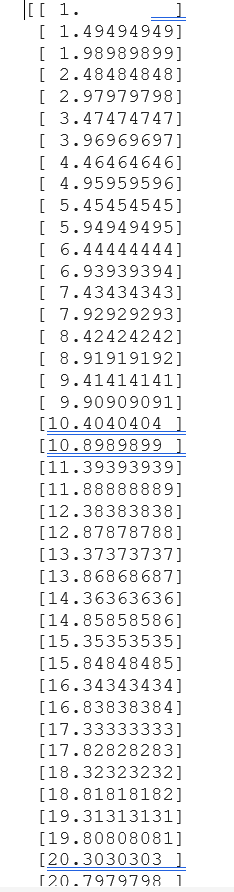
**#make predictions using the model**

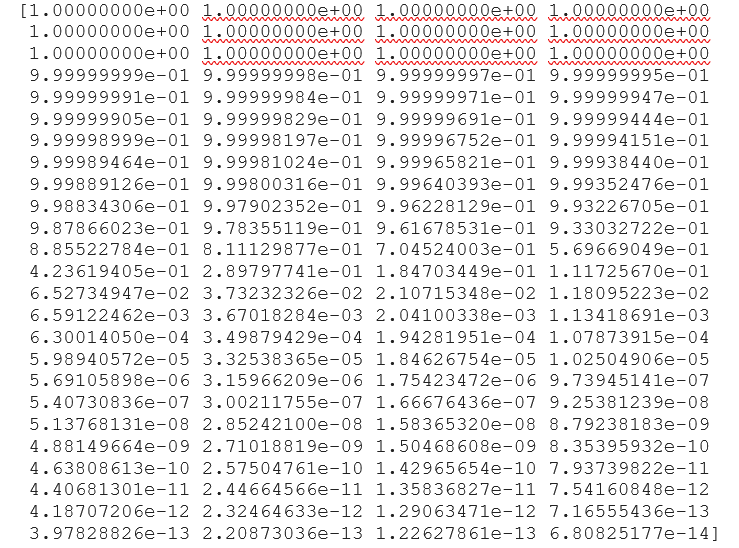
prob=logr.predict\_proba(dist)[:,1] #predictions

print(prob)

**Output:**

****

****



**Plotting:**

import matplotlib.pyplot as plt

#plotting actual data - train

plt.scatter(distances, binary\_labels, color='k', label='Data')

#plotting test data with predictions - valid/test

plt.plot(dist,prob,color='b',label='LogisticRegression')

plt.title('Distance vs Probability of scoring a goal')

plt.xlabel('Distance')

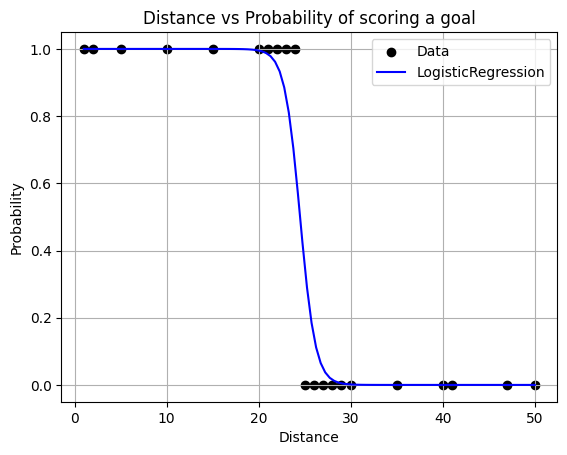
plt.ylabel('Probability')

plt.legend()

plt.grid(True)

plt.show()

**Output:**

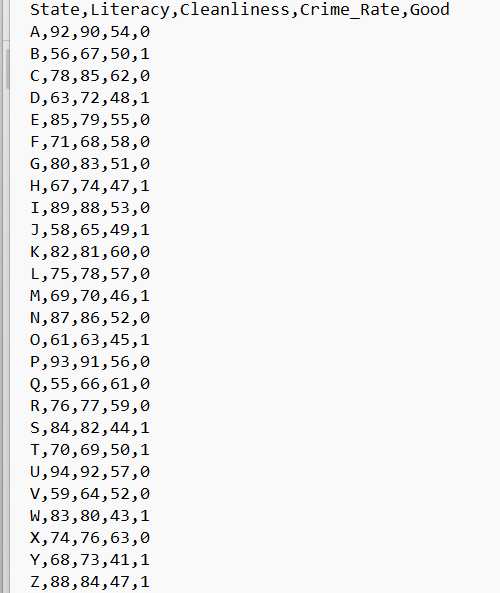


**3.Decision Trees**

* Decision Trees in machine learning provide an effective method for making decisions because they lay out the problem and all the possible outcomes.
* Have nodes and Leaves
* Node: Condition having True and False Branches.
* Leaf: Result- showing the dataset that is True and False to the condition

**Case Study:**

* Taking a dataset of 26 states with features like Literacy, Cleanliness, Crime Rate and targeting(predicting) Good or Bad State!
* Good is called Target variable here, it has value of 0s and 1s



Now let us create a Decision Tree

* Decision tree recurrently (continuously) splits the data until it gets pure leafs.
* Let us view a DT based on Crime Rate.

Building Decision Tree

1. Node1🡪 CR>60

True=[C,Q,X=0](Pure Leaf)

False=[A,E,F,G,I,K,L,N,P,R,U,V=0],[B,DH,J,M,O,S,T,W,X,Y,Z=1]

(Mixed Leaf)

Reason:The mixed leaf has target variable as both as 0s and 1s hence this data is splitted once again.

1. Node2🡪 CR>50

True=[A,E,F,G,I,K,L,N,P,R,U.V=0](Pure Leaf)

False=[ B,DH,J,M,O,S,T,W,X,Y,Z=1](Pure Leaf)

Result:

|  |  |
| --- | --- |
| CONDITION | Good |
| Crime Rate >60 | Can’t be determined |
| Crime Rate >50 | 1 |

Prediction:

* CR=63 good=0
* CR=45 good=1

**Ex:**

import pandas as pd

df=pd.read\_csv('/content/demodt.txt')

df

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | State | Literacy | Cleanliness | Crime\_Rate | Good |
| 0 | A | 92 | 90 | 54 | 0 |
| 1 | B | 56 | 67 | 50 | 1 |
| 2 | C | 78 | 85 | 62 | 0 |
| 3 | D | 63 | 72 | 48 | 1 |
| 4 | E | 85 | 79 | 55 | 0 |
| 5 | F | 71 | 68 | 58 | 0 |
| 6 | G | 80 | 83 | 51 | 0 |
| 7 | H | 67 | 74 | 47 | 1 |
| 8 | I | 89 | 88 | 53 | 0 |
| 9 | J | 58 | 65 | 49 | 1 |
| 10 | K | 82 | 81 | 60 | 0 |
| 11 | L | 75 | 78 | 57 | 0 |
| 12 | M | 69 | 70 | 46 | 1 |
| 13 | N | 87 | 86 | 52 | 0 |
| 14 | O | 61 | 63 | 45 | 1 |
| 15 | P | 93 | 91 | 56 | 0 |
| 16 | Q | 55 | 66 | 61 | 0 |
| 17 | R | 76 | 77 | 59 | 0 |
| 18 | S | 84 | 82 | 44 | 1 |
| 19 | T | 70 | 69 | 50 | 1 |
| 20 | U | 94 | 92 | 57 | 0 |
| 21 | V | 59 | 64 | 52 | 0 |
| 22 | W | 83 | 80 | 43 | 1 |
| 23 | X | 74 | 76 | 63 | 0 |
| 24 | Y | 68 | 73 | 41 | 1 |
| 25 | Z | 88 | 84 | 47 | 1 |

from sklearn.tree import DecisionTreeClassifier

LR=DecisionTreeClassifier()

target=df.Good

feat\_list=['Literacy','Cleanliness','Crime\_Rate']

feat=df[feat\_list]

LR.fit(feat,target)

pred=LR.predict([[90,90,78]])

print(pred)

if(pred==1):

print('Good')

else:

print('Bad')

**Output:**

[0]

Bad

**Problem:**

Enter the values manually and predict whether the state is good.

**Code:**

from sklearn.tree import DecisionTreeClassifier

LR=DecisionTreeClassifier()

target=df.Good

feat\_list=['Literacy','Cleanliness','Crime\_Rate']

feat=df[feat\_list]

LR.fit(feat,target)

Literacy=int(input("enter literacy number:"))

Cleanliness=int(input("enter cleanliness number:"))

Crime\_Rate=int(input("enter crime rate number:"))

pred=LR.predict([[Literacy,Cleanliness,Crime\_Rate]])

print(pred)

if(pred==1):

print('Good')

else:

print('Bad')

**Output:**

enter literacy number:89

enter cleanliness number:90

enter crime rate number:78

[0]

Bad

**4.Random Forest:**

* Collection of Many Decision Trees!
* Keywords: Bootstrapping, Aggregation

**Case Study:**

y=target

x0,x1,x2,x3,x4=features

**Bootstapping:**

Splitting the parent dataset into different child dataset.

**Condition of child dataset:**

Having same no. of rows and should have different row combinations.