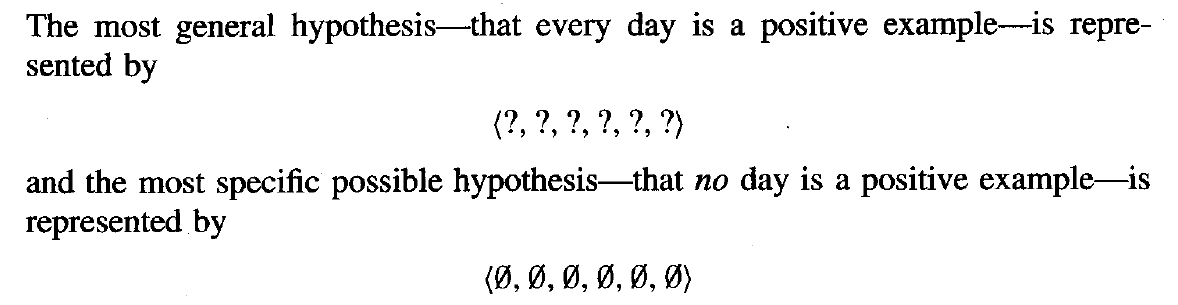
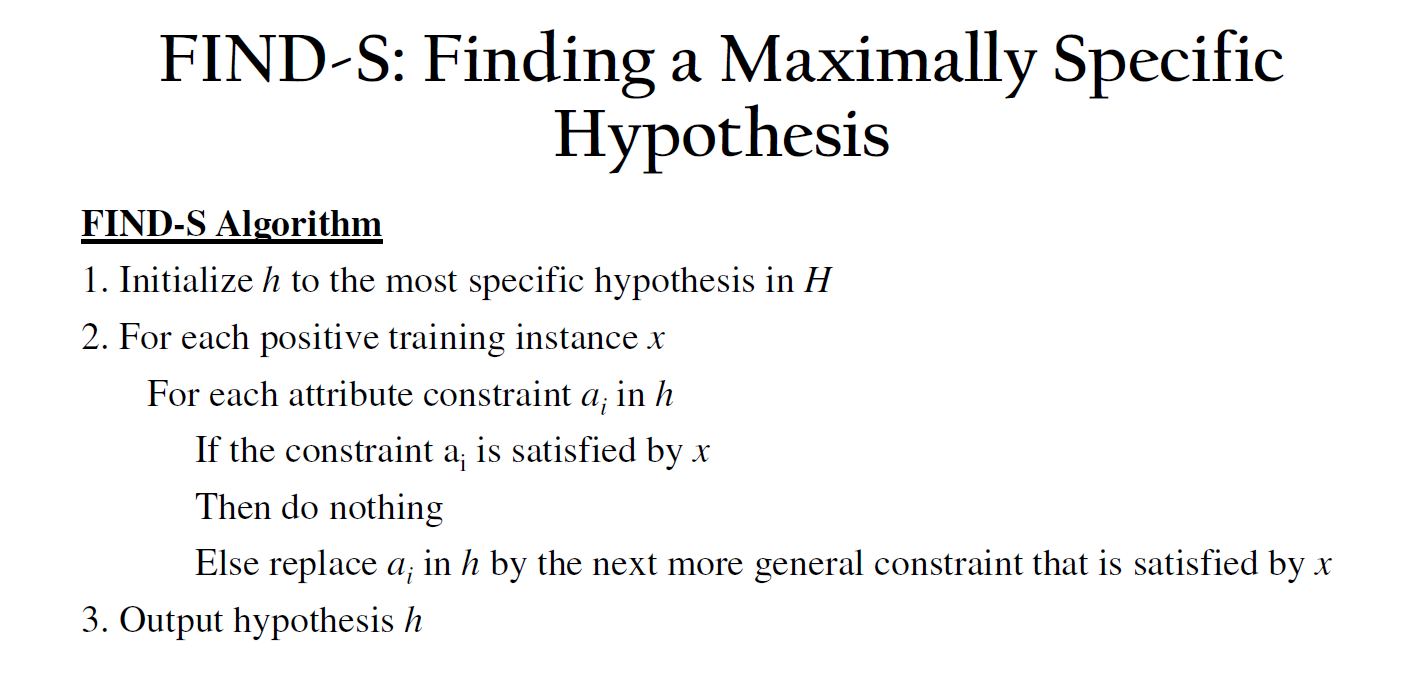
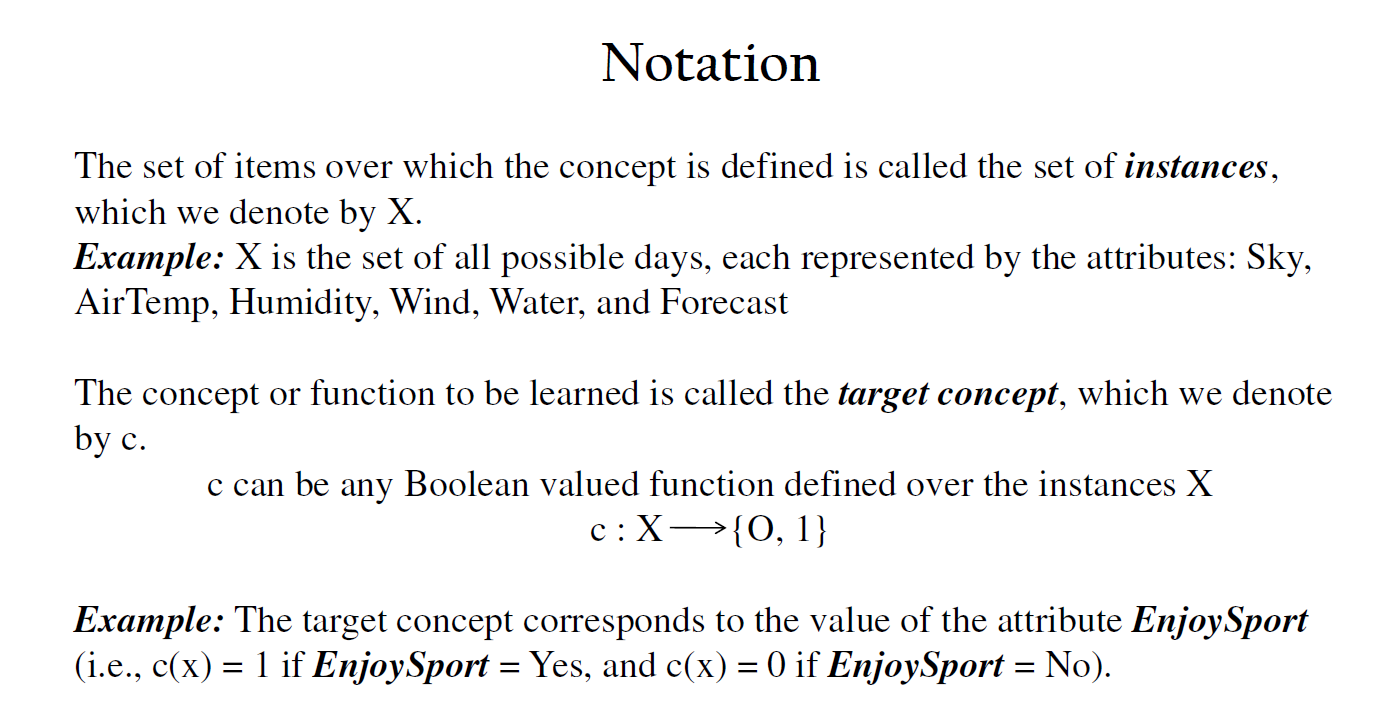
**Week-1: Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.**

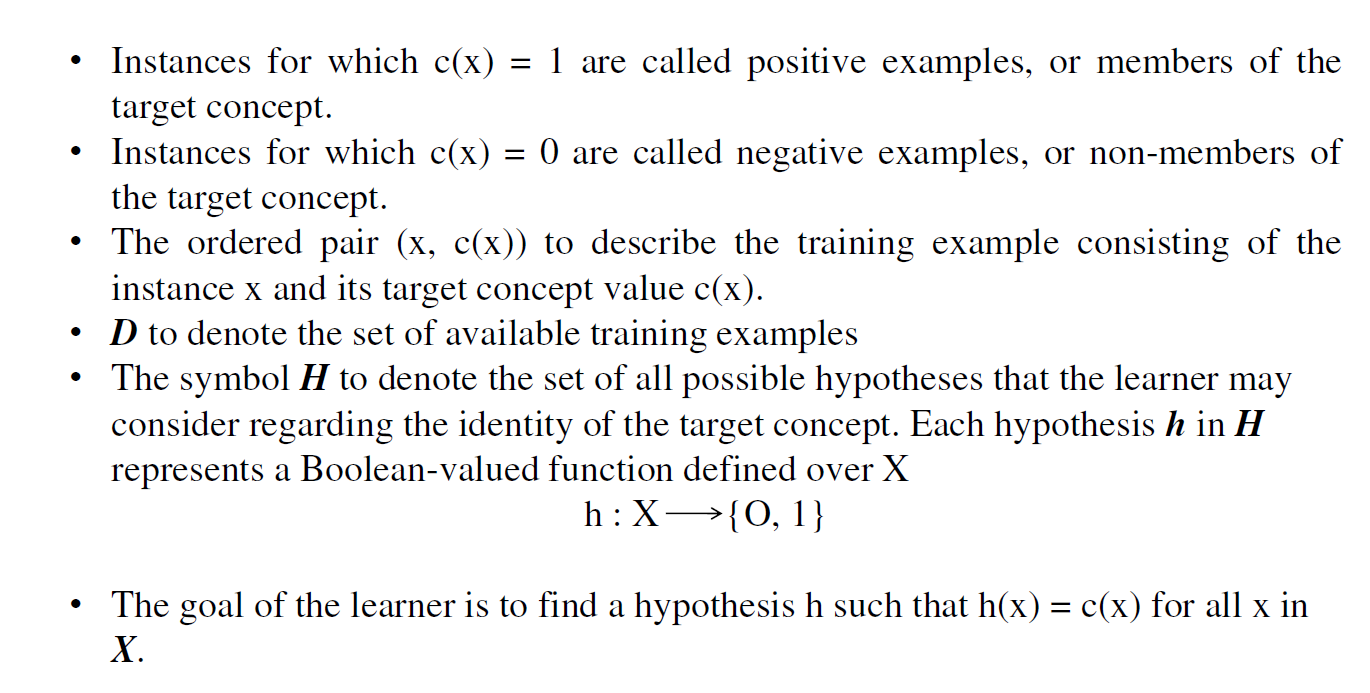
**Aim:** To implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

**Description:** The find-S algorithm is a basic concept learning algorithm in machine learning. The find-S algorithm finds the most specific hypothesis that fits all the positive examples. We have to note here that the algorithm considers only those positive training example. The find-S algorithm starts with the most specific hypothesis and generalizes this hypothesis each time it fails to classify an observed positive training data. Hence, the Find-S algorithm moves from the most specific hypothesis to the most general hypothesis.









**EnjoySport.csv:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| sky | airtemp | humidity | wind | water | forecast | enjoysport |
| sunny | warm | normal | strong | warm | same | yes |
| sunny | warm | high | strong | warm | same | yes |
| rainy | cold | high | strong | warm | change | no |
| sunny | warm | high | strong | cool | change | Yes |

**Program:**

import random

import csv

attributes=[["Sunny","Rainy"],["Warm","Cold"],["Normal","High"],["Strong","Weak"],["Warm","Cool"],["Same","Change"]]

num\_attributes=len(attributes)

print("The most general hypothesis: ['?','?','?','?','?','?']")

print("The most general hypothesis: ['0','0','0','0','0','0']")

a=[]

print("The given training dataset: ")

with open('/content/Week-1.csv','r') as csvFile:

  reader=csv.reader(csvFile)

  for row in reader:

    a.append(row)

    print(row)

print("The initial value of hypothesis: ")

hypothesis=['0']\*num\_attributes

print(hypothesis)

for j in range(0,num\_attributes):

  hypothesis[j]=a[0][j]

print("FIND-S: Finding a Maximality Specific Hypothesis")

for i in range(0,len(a)):

  if a[i][num\_attributes]=="yes":

    for j in range(0,num\_attributes):

      if a[i][j]!=hypothesis[j]:

        hypothesis[j]='?'

      else:

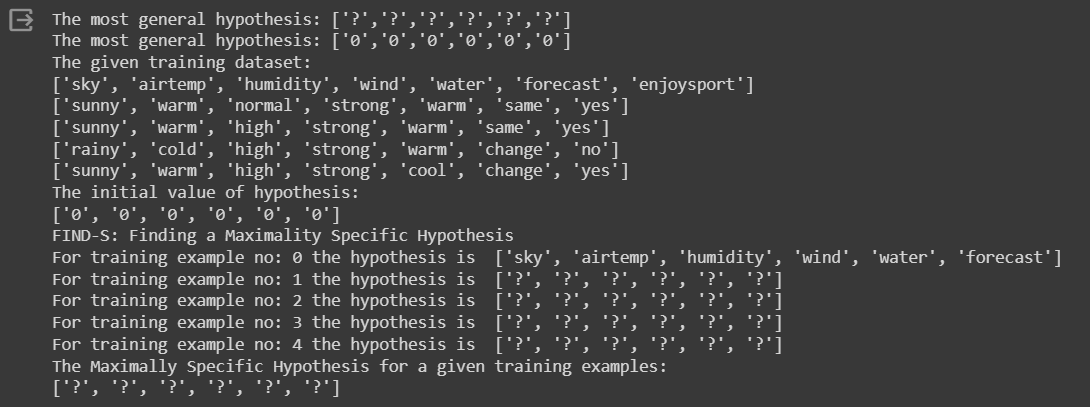
        hypothesis[j]=a[i][j]

  print("For training example no: {0} the hypothesis is ".format(i),hypothesis)

print("The Maximally Specific Hypothesis for a given training examples:")

print(hypothesis)

**Output:**

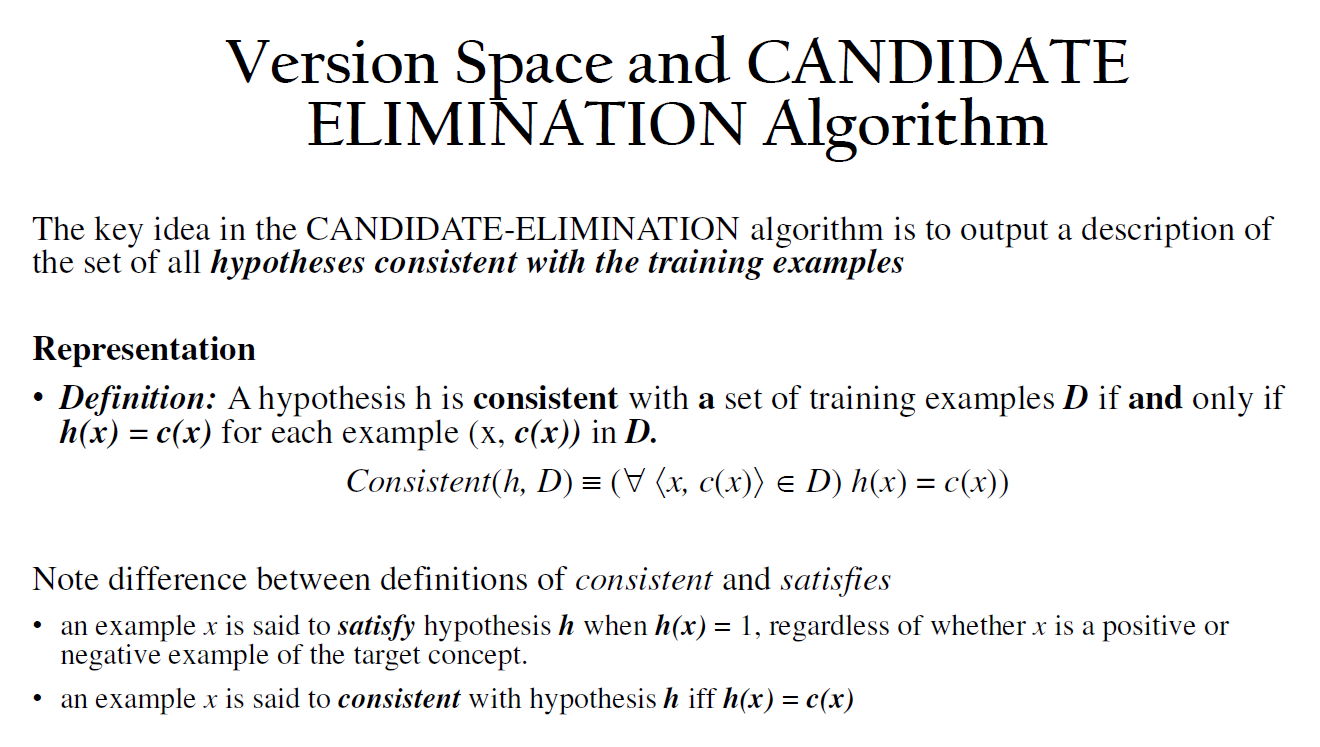


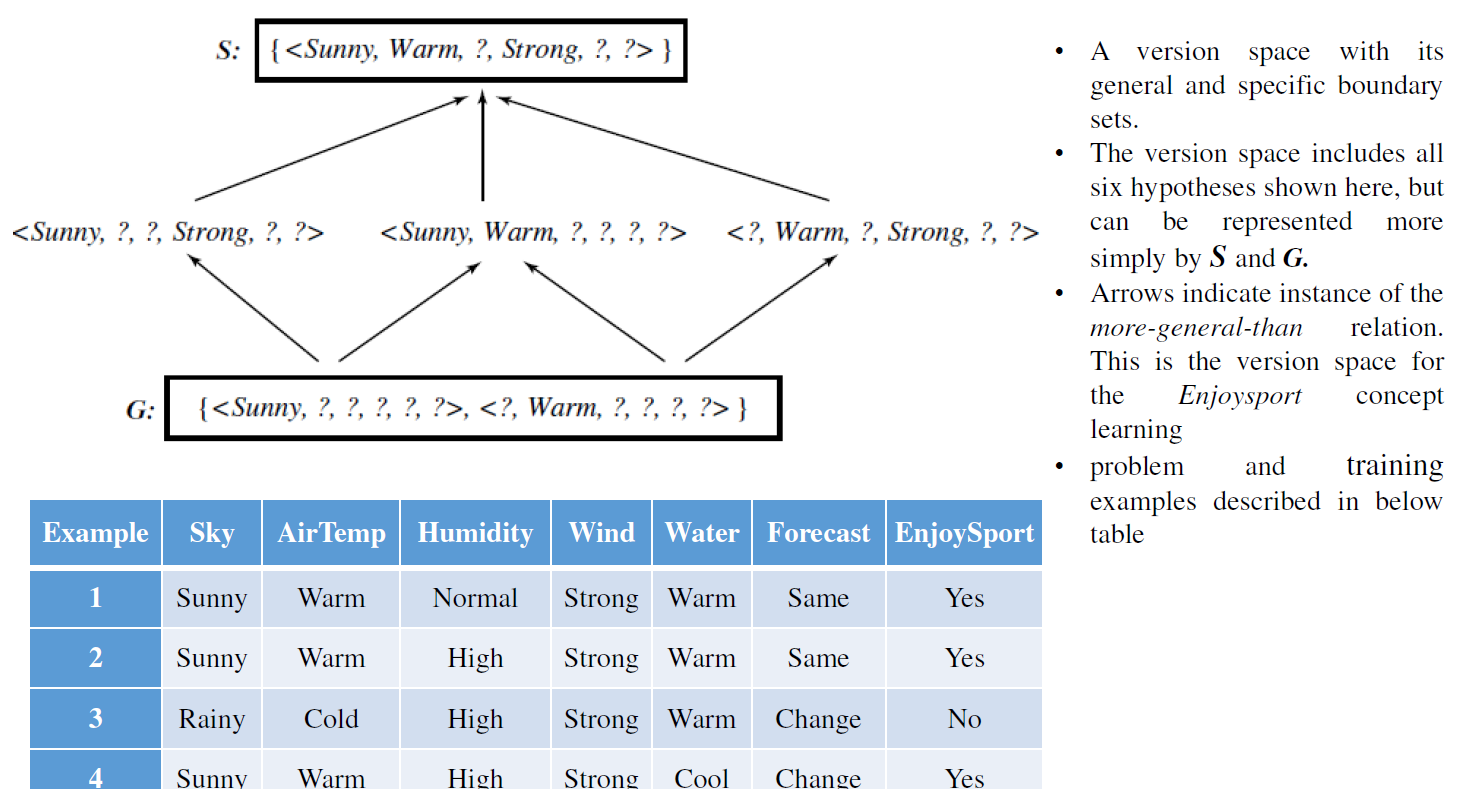
**Week-2: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.**

**Aim:** To implement and demonstrate the Candidate-Elimination algorithm for a given set of training data examples stored in a .CSV file, to output a description of the set of all hypotheses consistent with the training examples.

**Description:** The candidate Elimination algorithm finds all hypotheses that match all the given training examples. Unlike in Find-S algorithm and List-then-Eliminate algorithm, it goes through both negative and positive examples, eliminating any inconsistent hypothesis.

It incrementally builds the version space given a hypothesis space H and a set E of examples. The examples are added one by one; each example possibly shrinks the version space by removing the hypotheses that are inconsistent with the example. The candidate elimination algorithm does this by updating the general and specific boundary for each new example.





**EnjoySport.csv:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| sky | airtemp | humidity | wind | water | forecast | enjoysport |
| sunny | warm | normal | strong | warm | same | yes |
| sunny | warm | high | strong | warm | same | yes |
| rainy | cold | high | strong | warm | change | no |
| sunny | warm | high | strong | cool | change | Yes |

**Program:**

import numpy as np

import pandas as pd

data=pd.DataFrame(data=pd.read\_csv('/content/enjoysport.csv'))

concepts=np.array(data.iloc[:,0:-1])

print(concepts)

target=np.array(data.iloc[:,-1])

print(target)

def learn(concepts,target):

  specific\_h=concepts[0].copy()

  print("Initialization of specific\_h and general\_h:")

  print(specific\_h)

  general\_h=[["?" for i in range(len(specific\_h))] for i in range(len(specific\_h))]

  print(general\_h)

  for i,h in enumerate(concepts):

    if target[i]=="yes":

      for x in range(len(specific\_h)):

        if h[x]!=specific\_h[x]:

          specific\_h[x]='?'

          general\_h[x][x]='?'

    if target[i]=="no":

      for x in range(len(specific\_h)):

        if h[x]!=specific\_h[x]:

          general\_h[x][x]=specific\_h[x]

        else:

          general\_h[x][x]='?'

    print("Steps of candidate elimination algorithm",i+1)

    print(specific\_h)

    print(general\_h)

  indices=[i for i,val in enumerate(general\_h) if val==['?','?','?','?','?','?']]

  for i in indices:

    general\_h.remove(['?','?','?','?','?','?'])

  return specific\_h,general\_h

s\_final,g\_final=learn(concepts,target)

print("Final Specific\_h: ",s\_final,sep="\n")

print("Final General\_h: ",g\_final,sep="\n")

**Output:**

[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

['sunny' 'warm' 'high' 'strong' 'warm' 'same']

['rainy' 'cold' 'high' 'strong' 'warm' 'change']

['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

['yes' 'yes' 'no' 'yes']

Initialization of specific\_h and general\_h:

['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of candidate elimination algorithm 1

['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of candidate elimination algorithm 2

['sunny' 'warm' '?' 'strong' 'warm' 'same']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of candidate elimination algorithm 3

['sunny' 'warm' '?' 'strong' 'warm' 'same']

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]

Steps of candidate elimination algorithm 4

['sunny' 'warm' '?' 'strong' '?' '?']

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific\_h:

['sunny' 'warm' '?' 'strong' '?' '?']

Final General\_h:

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

**Week-3: Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.**

**Aim:** Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

**Description:** ID3 Algorithm is used in machine learning for building decision trees from a given dataset. It was developed in 1986 by Ross Quinlan. It is a greedy algorithm that builds a decision tree by recursively partitioning the data set into smaller and smaller subsets until all data points in each subset belong to the same class. It employs a top-down approach, recursively selecting features to split the dataset based on information gain.

ID3(Examples, Target\_attribute, Attributes)

Examples are the training examples. Target\_attribute is the attribute whose value is to be

predicted by the tree. Attributes is a list of other attributes that may be tested by the

learned decision tree. Returns a decision tree that correctly classifies the given Examples.

* Create a Root node for the tree
* If all Examples are positive, Return the single-node tree Root, with label = +
* If all Examples are negative, Return the single-node tree Root, with label = -
* If Attributes is empty, Return the single-node tree Root, with label = most common value of Target\_attribute in Examples
* Otherwise Begin
  + A ← the attribute from Attributes that best classifies Examples
  + The decision attribute for Root ← A
  + For each possible value, vi, of A,
    - Add a new tree branch below Root, corresponding to the test A = vi
    - Let Examples vi, be the subset of Examples that have value vi for A
    - If Examples vi , is empty
      * Then below this new branch add a leaf node with label = most common value of Target\_attribute in Examples
      * Else below this new branch add the subtree

ID3(Examples vi, Targe\_tattribute, Attributes – {A}))

* End
* Return Root

**ENTROPY:** Entropy measures the impurity of a collection of examples.

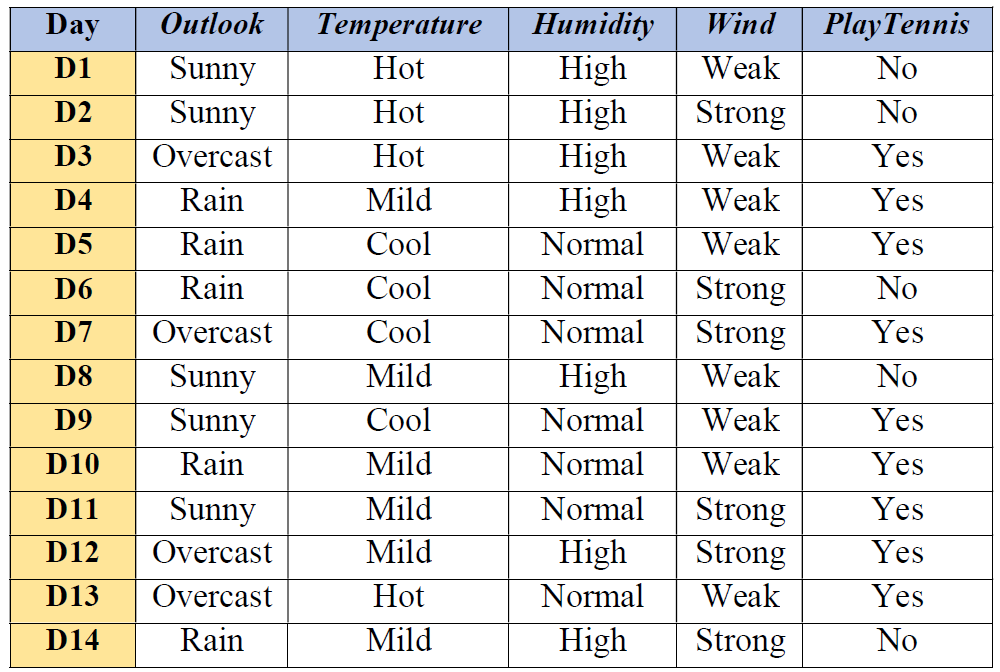


Where, p+ is the proportion of positive examples in S and p- is the proportion of negative examples in S.

**INFORMATION GAIN:** Information gain, is the expected reduction in entropy caused by partitioning the examples according to this attribute. The information gain, Gain(S, A) of an attribute A, relative to a collection of examples S, is defined as:



**Training Dataset:**



**Program:**

import pandas as pd

import math

def id3(df, target\_attribute\_name, attribute\_names, default\_class=None):

# Base cases for recursion

# If all instances have the same class, return that class

if len(set(df[target\_attribute\_name])) == 1:

return df[target\_attribute\_name].iloc[0]

# If attribute\_names is empty, return the default class

elif len(attribute\_names) == 0:

return default\_class

else:

# Calculate information gain for each attribute

gains = {attribute\_name: information\_gain(df, attribute\_name, target\_attribute\_name) for attribute\_name in attribute\_names}

# Choose the attribute with the highest information gain

best\_attribute = max(gains, key=gains.get)

# Create an empty tree

tree = {best\_attribute: {}}

# Remove the best attribute from the list of attributes

remaining\_attributes = [attr for attr in attribute\_names if attr != best\_attribute]

# Recursively build the tree for each value of the best attribute

for value in df[best\_attribute].unique():

subset = df[df[best\_attribute] == value]

subtree = id3(subset, target\_attribute\_name, remaining\_attributes, default\_class)

tree[best\_attribute][value] = subtree

return tree

# Define functions for entropy and information gain

def entropy(probs):

return sum([-prob \* math.log(prob, 2) for prob in probs if prob != 0])

def entropy\_of\_list(a\_list):

total\_instances = len(a\_list)

class\_counts = a\_list.value\_counts()

probs = class\_counts / total\_instances

return entropy(probs)

def information\_gain(df, split\_attribute\_name, target\_attribute\_name):

total\_entropy = entropy\_of\_list(df[target\_attribute\_name])

subset\_entropy = df.groupby(split\_attribute\_name)[target\_attribute\_name].apply(entropy\_of\_list)

subset\_sizes = df.groupby(split\_attribute\_name).size()

weighted\_entropy = (subset\_entropy \* subset\_sizes / len(df)).sum()

return total\_entropy - weighted\_entropy

# Read the dataset

df = pd.read\_csv('/content/id3.csv')

# Get attribute names and remove the target attribute

attribute\_names = list(df.columns)

target\_attribute\_name = 'Answer'

attribute\_names.remove(target\_attribute\_name)

# Build the decision tree

tree = id3(df, target\_attribute\_name, attribute\_names)

# Print the resultant decision tree

print("Decision Tree:")

print(tree)

**Output:**

Decision Tree:

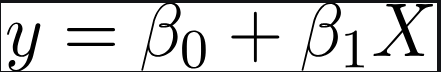
{'Outlook': {'sunny': {'Humidity': {'high': 'no', 'normal': 'yes'}}, 'overcast': 'yes', 'rain': {'Wind': {'weak': 'yes', 'strong': 'no'}}}}

**Week-4: Exercises to solve the real-world problems using Linear Regression.**

**Aim:** To write a program to solve the real-world problems using Linear Regression.

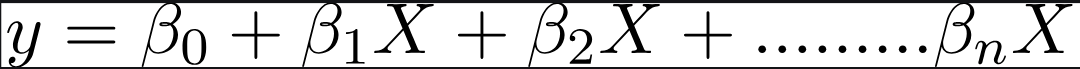
**Description:** Linear regression is a type of supervised machine learning algorithm that computes the linear relationship between a dependent variable and one or more independent features. When the number of the independent feature, is 1 then it is known as Univariate Linear regression, and in the case of more than one feature, it is known as multivariate linear regression. There are two main types of linear regression:

**Simple Linear Regression:** This is the simplest form of linear regression, and it involves only one independent variable and one dependent variable. The equation for simple linear regression



Y is the dependent variable, X is the independent variable, β0 is the intercept, β1 is the slope.

**Multiple Linear Regression:** This involves more than one independent variable and one dependent variable. The equation for multiple linear regression is:



Y is the dependent variable, X1, X2, …, Xp are the independent variables, β0 is the intercept, β1, β2, …, βn are the slopes. The goal of the algorithm is to find the best Fit Line equation that can predict the values based on the independent variables.

**Program and Output:**

import pandas as pd

import numpy as np

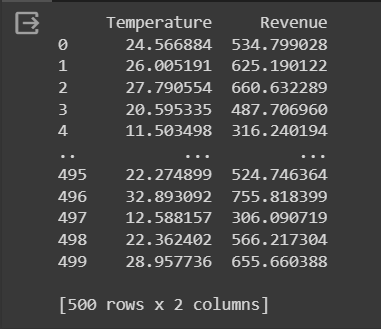
import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

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

IceCream=pd.read\_csv('/content/IceCreamData.csv')

print(IceCream)



# Divide the data into “Attributes” and “labels”

X = IceCream[['Temperature']]

y = IceCream['Revenue']

# Split 80% of the data to the training set while 20% of the data to test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

# Create a Linear Regression model and fit it

regressor =LinearRegression(fit\_intercept=True)

regressor.fit(X\_train,y\_train)

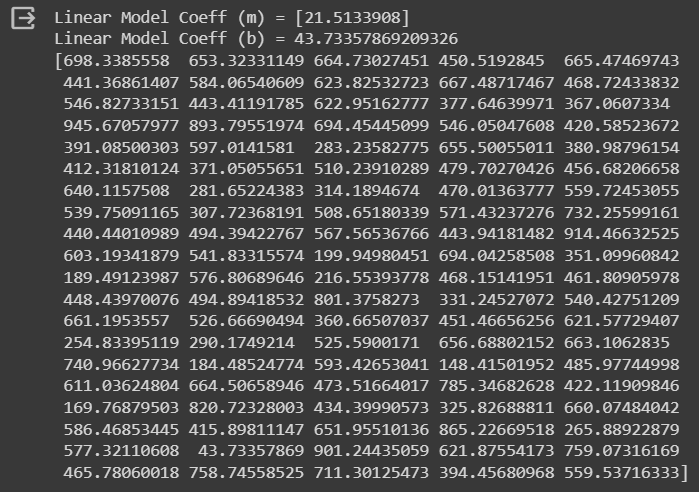
print('Linear Model Coeff (m) =' , regressor.coef\_)

print('Linear Model Coeff (b) =' , regressor.intercept\_)

# Predicting the data

y\_predict=regressor.predict(X\_test)

print(y\_predict)



# Scatter plot on Training Data

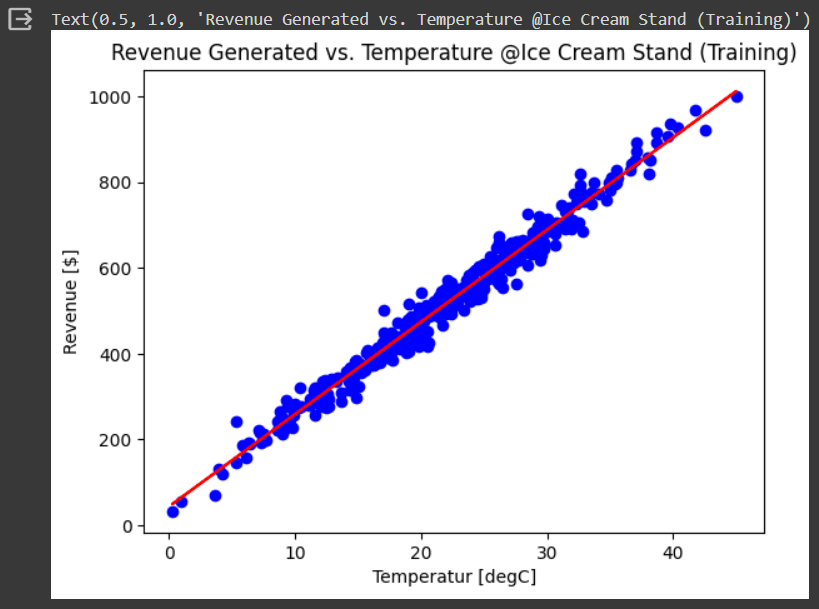
plt.scatter(X\_train,y\_train,color='blue')

plt.plot(X\_train,regressor.predict(X\_train),color='red')

plt.ylabel('Revenue [$]')

plt.xlabel('Temperatur [degC]')

plt.title('Revenue Generated vs. Temperature @Ice Cream Stand (Training)')



# Scatter plot on Testing Data

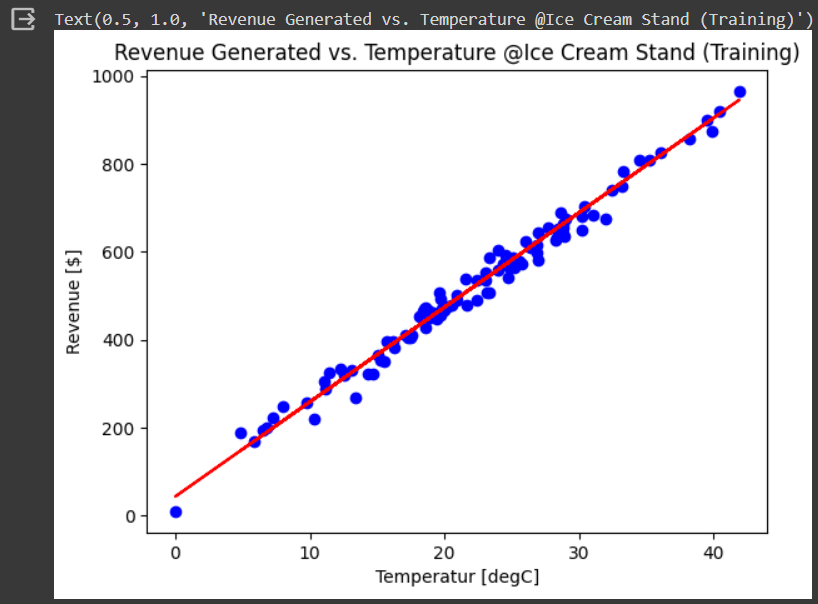
plt.scatter(X\_test,y\_test,color='blue')

plt.plot(X\_test,regressor.predict(X\_test),color='red')

plt.ylabel('Revenue [$]')

plt.xlabel('Temperatur [degC]')

plt.title('Revenue Generated vs. Temperature @Ice Cream Stand (Training)')



# Prediction the revenve using Temperature Value directly

print('---------0---------')

Temp = -0

Revenue = regressor.predict([[Temp]])

print(Revenue)

print('--------35----------')

Temp = 35

Revenue = regressor.predict([[Temp]])

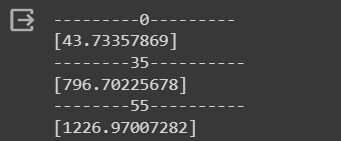
print(Revenue)

print('--------55----------')

Temp = 55

Revenue = regressor.predict([[Temp]])

print(Revenue)



**Week-5: Exercises to solve the real-world problems using Logistic Regression**

**Aim:** To write a program to solve the real-world problem using Logistic Regression.

**Description:** Logistic regression is used for binary classification where we use sigmoid function, that takes input as independent variables and produces a probability value between 0 and 1. Logistic regression predicts the output of a categorical dependent variable. Therefore, the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1. In Logistic regression, instead of fitting a regression line, we fit an “S” shaped logistic function, which predicts two maximum values (0 or 1).

**Types of Logistic Regression:**

On the basis of the categories, Logistic Regression can be classified into three types:

**Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.

**Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as “cat”, “dogs”, or “sheep”.

**Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as “low”, “Medium”, or “High”.

**Program and Output:**

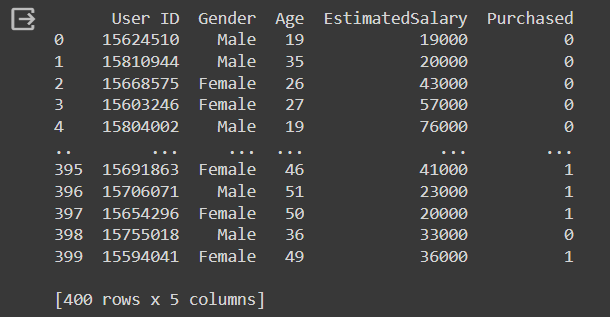
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('/content/Social\_Network\_Ads.csv')

print(dataset)



X = dataset.iloc[:, [2,3]].values

y = dataset.iloc[:, 4].values

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

from sklearn.preprocessing import StandardScaler

sc\_X = StandardScaler()

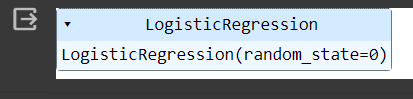
X\_train = sc\_X.fit\_transform(X\_train)

X\_test = sc\_X.transform(X\_test)

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state=0)

classifier.fit(X\_train, y\_train)



y\_pred = classifier.predict(X\_test)

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

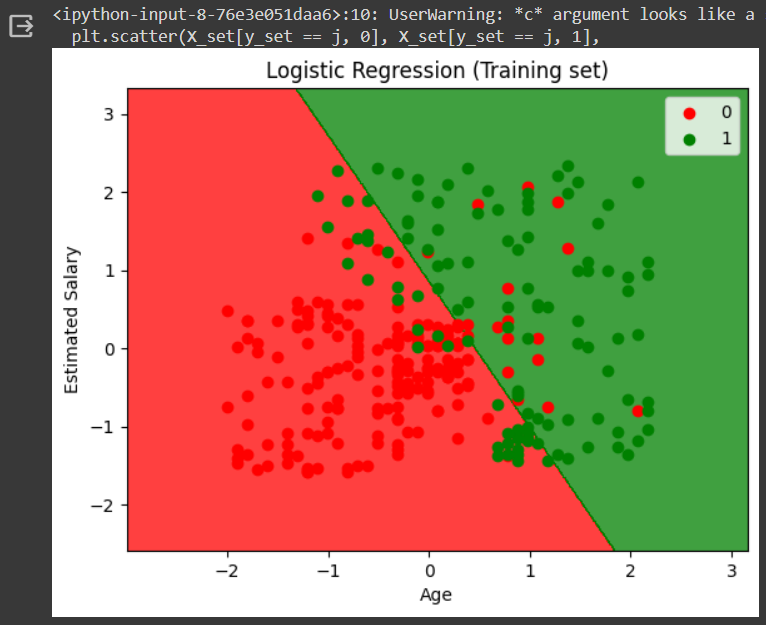
plt.title('Logistic Regression (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()



from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Logistic Regression (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()



**Week-6: Exercises to solve the real-world problems using Binary Classifier.**

**Aim:** To write a program to solve the real-world problem using Binary Classifier.

**Description:** Binary Classification is a type of machine learning algorithm used to classify data into one of two categories. It predicts a binary outcome, where the result can either be positive or negative. For example, binary classification can be used to predict if a customer will buy a product or not, or if an email is spam or not. We can evaluate a binary classifier based on the following parameters:

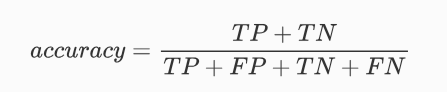
True Positive (TP): The patient is diseased and the model predicts "diseased"

False Positive (FP): The patient is healthy but the model predicts "diseased"

True Negative (TN): The patient is healthy and the model predicts "healthy"

False Negative (FN): The patient is diseased and the model predicts "healthy"

After obtaining these values, we can compute the accuracy score of the binary classifier as follows:





**Program and Output:**

import matplotlib.pyplot as plt

from sklearn.datasets import load\_breast\_cancer

dataset=load\_breast\_cancer(as\_frame=True)

X=dataset['data']

y=dataset['target']

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

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25,random\_state=0)

from sklearn.preprocessing import StandardScaler

ss\_train=StandardScaler()

X\_train=ss\_train.fit\_transform(X\_train)

ss\_test=StandardScaler()

X\_test=ss\_test.fit\_transform(X\_test)

models={}

# Logistic Regression

from sklearn.linear\_model import LogisticRegression

models['Logistic Regression']=LogisticRegression()

# Support Vector Machines

from sklearn.svm import LinearSVC

models['Support Vector Machines']=LinearSVC()

# Decision Trees

from sklearn.tree import DecisionTreeClassifier

models['Decision Trees']=DecisionTreeClassifier()

# Random Forest

from sklearn.ensemble import RandomForestClassifier

models['Random Forest']=RandomForestClassifier()

# Naive Bayes

from sklearn.naive\_bayes import GaussianNB

models['Naive Bayes']=GaussianNB()

# K-Nearest Neighbors

from sklearn.neighbors import KNeighborsClassifier

models['K-Nearest Neighbor']=KNeighborsClassifier()

from sklearn.metrics import accuracy\_score,precision\_score,recall\_score

accuracy,precision,recall={},{},{}

for key in models.keys():

models[key].fit(X\_train,y\_train)

predictions=models[key].predict(X\_test)

accuracy[key]=accuracy\_score(predictions,y\_test)

precision[key]=precision\_score(predictions,y\_test)

recall[key]=recall\_score(predictions,y\_test)

import pandas as pd

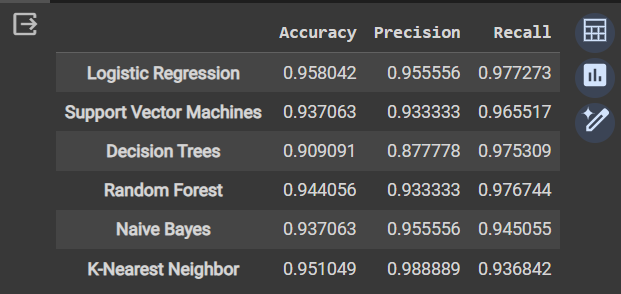
df\_model=pd.DataFrame(index=models.keys(),columns=['Accuracy','Precision','Recall'])

df\_model['Accuracy']=accuracy.values()

df\_model['Precision']=precision.values()

df\_model['Recall']=recall.values()

df\_model



ax=df\_model.plot.barh()

ax.legend(

ncol=len(models.keys()),

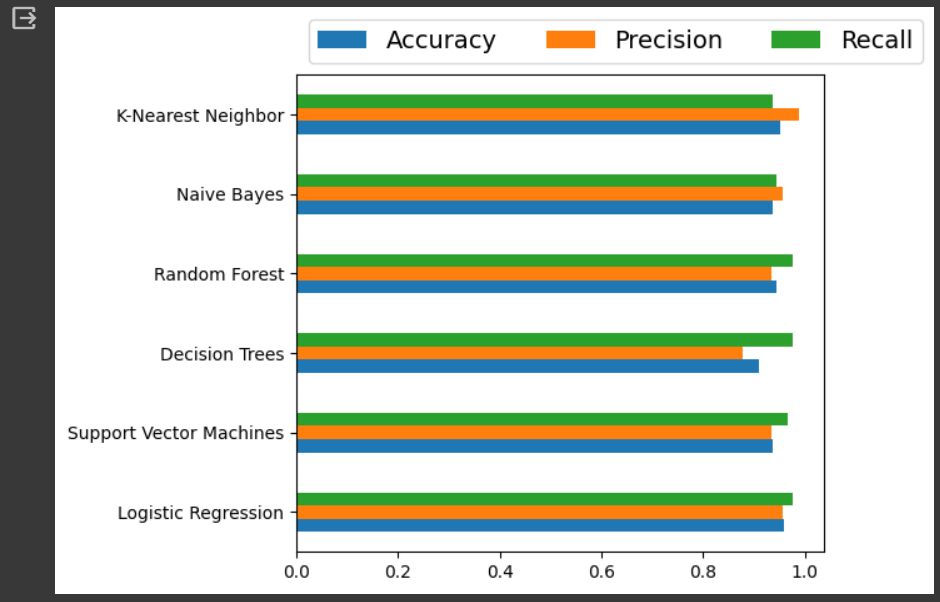
bbox\_to\_anchor=(0,1),

loc="lower left",

prop={'size':14}

)

plt.tight\_layout()



from sklearn.metrics import confusion\_matrix

cm=confusion\_matrix(y\_test,predictions)

TN,FP,FN,TP=confusion\_matrix(y\_test,predictions).ravel()

print("True Positive(TP):",TP)

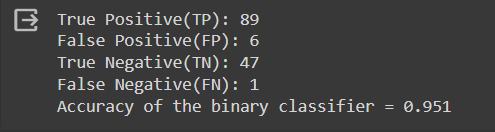
print("False Positive(FP):",FP)

print("True Negative(TN):",TN)

print("False Negative(FN):",FN)

accuracy=(TP+TN)/(TP+FP+TN+FN)

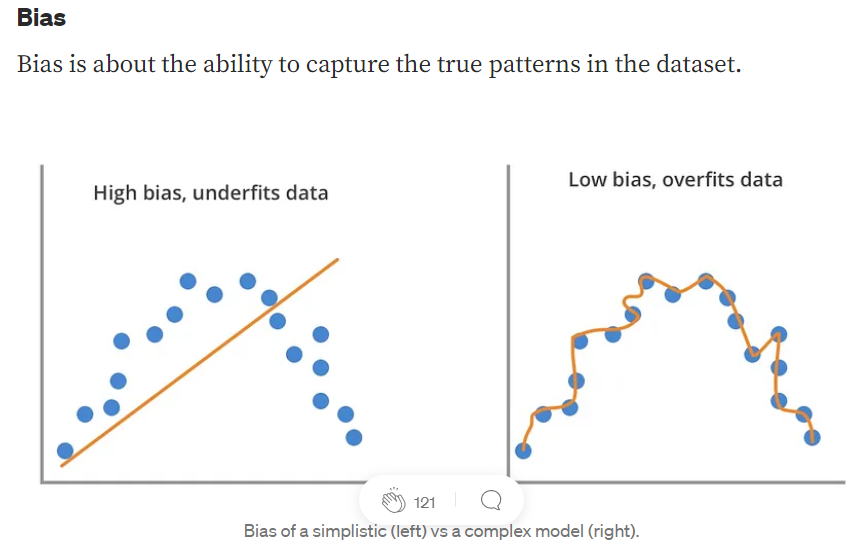
print("Accuracy of the binary classifier = {:0.3f}".format(accuracy))

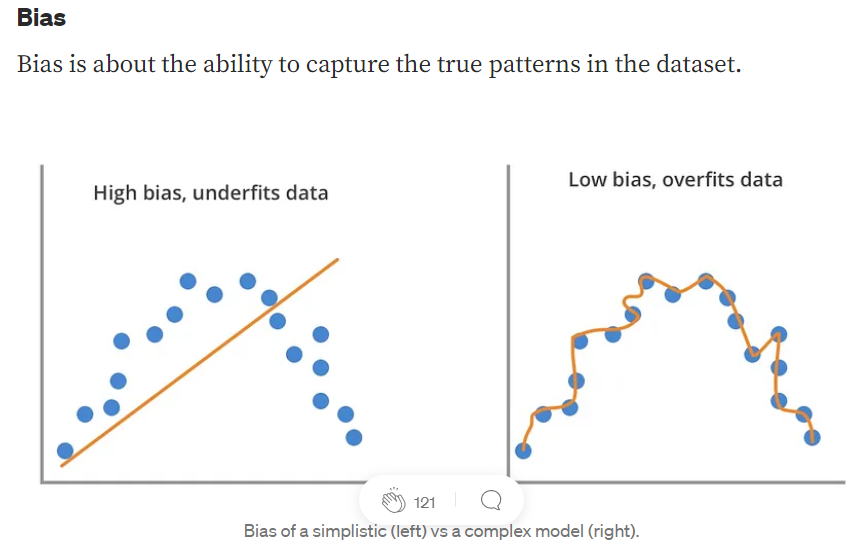


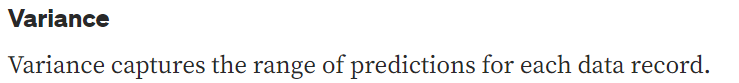
**Week-7: Develop a program for Bias, Variance, Remove duplicates, Cross Validation**

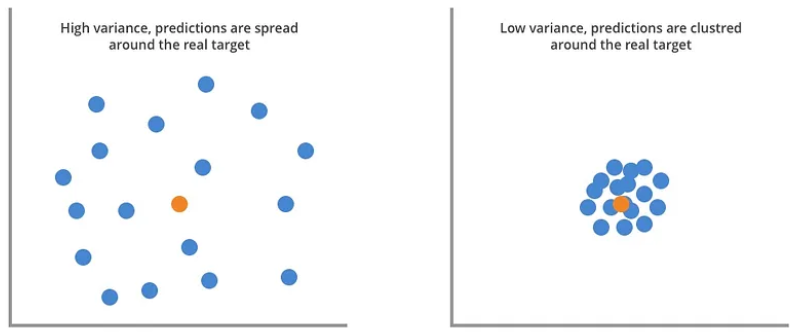
**Aim:** To develop a program for Bias, Variance, Remove duplicates, Cross Validation

**Description:**

****

****

****

****

**Cross Validation:** Cross validation is a technique used in machine learning to evaluate the performance of a model on unseen data. It involves dividing the available data into multiple folds or subsets, using one of these folds as a validation set, and training the model on the remaining folds. This process is repeated multiple times, each time using a different fold as the validation set. Finally, the results from each validation step are averaged to produce a more robust estimate of the model’s performance.

**Program:**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.linear\_model import LogisticRegression

from sklearn.preprocessing import LabelEncoder

iris\_data = pd.read\_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")

print("Original dataset:")

print(iris\_data.head())

# Remove duplicates

iris\_data\_no\_duplicates = iris\_data.drop\_duplicates()

# Split dataset into features and target variable

X = iris\_data\_no\_duplicates.drop(columns=['species'])

y = iris\_data\_no\_duplicates['species']

# Convert categorical labels into numerical values

label\_encoder = LabelEncoder()

y = label\_encoder.fit\_transform(y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = LogisticRegression(max\_iter=1000)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

# Calculate bias and variance

bias = np.mean((y\_test - y\_pred) \*\* 2)

variance = np.mean(np.var(y\_pred, axis=0))

print("Bias:", bias)

print("Variance:", variance)

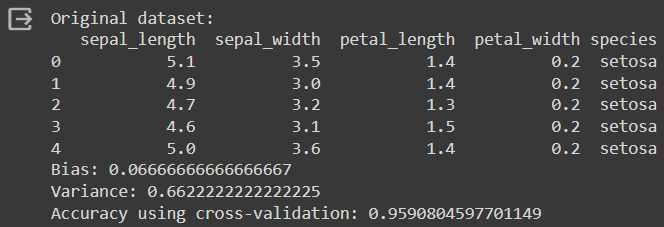
# Cross-validation

scores = cross\_val\_score(model, X, y, cv=5, scoring='accuracy')

accuracy\_cv = scores.mean()

print("Accuracy using cross-validation:", accuracy\_cv)

**Output:**



**Week-8: Write a program to implement One-hot Encoding.**

**Aim:** To write a program to implement One-hot Encoding.

**Description:** One-hot encoding is a technique in machine learning that turns categorical data, like colors (red, green, blue), into numerical data for machines to understand. It creates new binary columns for each category, with a 1 marking the presence of that category and 0s elsewhere. This allows machine learning algorithms to process the information in categorical data without misinterpreting any order between the categories. The advantages of using one hot encoding include:

* It allows the use of categorical variables in models that require numerical input.
* It can improve model performance by providing more information to the model about the categorical variable.
* It can help to avoid the problem of ordinality, which can occur when a categorical variable has a natural ordering (e.g. “small”, “medium”, “large”).

**Program:**

import pandas as pd

from sklearn.preprocessing import OneHotEncoder

data = {'Employee id': [10, 20, 15, 25, 30],

'Gender': ['M', 'F', 'F', 'M', 'F'],

'Remarks': ['Good', 'Nice', 'Good', 'Great', 'Nice'],

}

df = pd.DataFrame(data)

print(f"Employee data : \n{df}")

#Here we extract the columns with object datatype as they are the categorical columns

categorical\_columns = df.select\_dtypes(include=['object']).columns.tolist()

encoder = OneHotEncoder(sparse\_output=False)

# Apply one-hot encoding to the categorical columns

one\_hot\_encoded = encoder.fit\_transform(df[categorical\_columns])

one\_hot\_df =pd.DataFrame(one\_hot\_encoded,

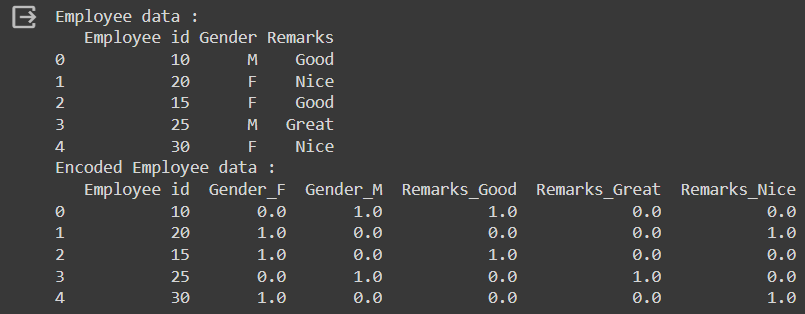
columns=encoder.get\_feature\_names\_out(categorical\_columns))

df\_encoded = pd.concat([df, one\_hot\_df], axis=1)

df\_encoded = df\_encoded.drop(categorical\_columns, axis=1)

print(f"Encoded Employee data : \n{df\_encoded}")

**Output:**



**Week-9: Write a program to implement Categorical Encoding.**

**Aim:** To write a program to implement Categorical Encoding.

**Description:**

A combination of numerical as well as categorical variables. A machine can only understand the numbers. It cannot understand the text.

**Categorical Encoding:** Categorical Encoding refers to the process of converting categorical (or qualitative) data into a numerical format that machine learning algorithms can understand. Categorical data represents categories or groups and does not have a natural numerical representation. In machine learning, it's common to encounter datasets with categorical features such as colors, shapes, cities, or labels.

**Label Encoding:** Label Encoding is a popular encoding technique for handling categorical variables. In this technique, each label is assigned a unique integer based on alphabetical order.

**Program:**

import pandas as pd

from sklearn.preprocessing import LabelEncoder

data = {

'Color': ['Red', 'Blue', 'Green', 'Red', 'Green'],

'Size': ['Small', 'Large', 'Medium', 'Medium', 'Small'],

'Shape': ['Circle', 'Square', 'Triangle', 'Circle', 'Square'],

'Label': ['A', 'B', 'C', 'A', 'B']

}

df = pd.DataFrame(data)

print("Original dataset:")

print(df)

label\_encoder = LabelEncoder()

df\_encoded = df.copy()

for col in df.columns:

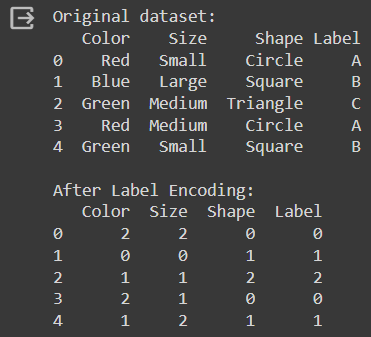
if df[col].dtype == 'object':

df\_encoded[col] = label\_encoder.fit\_transform(df[col])

print("\nAfter Label Encoding:")

print(df\_encoded)

**Output:**



**Week-10: Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.**

**Aim:** To build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

**Description:** Backpropagation is an iterative algorithm, that helps to minimize the cost function by determining which weights and biases should be adjusted. During every epoch, the model learns by adapting the weights and biases to minimize the loss by moving down toward the gradient of the error. Thus, it involves the two most popular optimization algorithms, such as gradient descent or stochastic gradient descent.

**Program:**

import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X,axis=0)

y = y/100

def sigmoid (x): return 1/(1 + np.exp(-x))

def derivatives\_sigmoid(x): return x \* (1 - x)

epoch=5000

lr=0.1

inputlayer\_neurons = 2

hiddenlayer\_neurons = 3

output\_neurons = 1

wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons))

bh=np.random.uniform(size=(1,hiddenlayer\_neurons))

wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons))

bout=np.random.uniform(size=(1,output\_neurons))

for i in range(epoch):

hinp1=np.dot(X,wh);hinp=hinp1 + bh;hlayer\_act = sigmoid(hinp);

outinp1=np.dot(hlayer\_act,wout);outinp= outinp1+ bout;output = sigmoid(outinp);

EO = y-output;outgrad = derivatives\_sigmoid(output);d\_output = EO\* outgrad;

EH = d\_output.dot(wout.T);hiddengrad = derivatives\_sigmoid(hlayer\_act);

d\_hiddenlayer = EH \* hiddengrad;wout += hlayer\_act.T.dot(d\_output) \*lr;

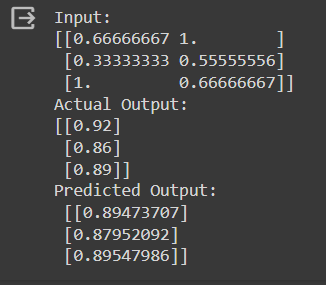
wh += X.T.dot(d\_hiddenlayer) \*lr

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)

**Output:**



**Week-11: Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions.**

**Aim:** To write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions.

**Description:** KNN is one of the most basic yet essential classification algorithms in machine learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining, and intrusion detection. It is widely disposable in real-life scenarios since it is non-parametric, meaning it does not make any underlying assumptions about the distribution of data (as opposed to other algorithms such as GMM, which assume a Gaussian distribution of the given data). We are given some prior data (also called training data), which classifies coordinates into groups identified by an attribute.

**Program:**

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn import datasets

iris=datasets.load\_iris()

x\_train, x\_test, y\_train, y\_test = train\_test\_split(iris.data,iris.target,test\_size=0.1)

for i in range(len(iris.target\_names)):

print("Label", i , "-",str(iris.target\_names[i]))

classifier = KNeighborsClassifier(n\_neighbors=2)

classifier.fit(x\_train, y\_train)

y\_pred=classifier.predict(x\_test)

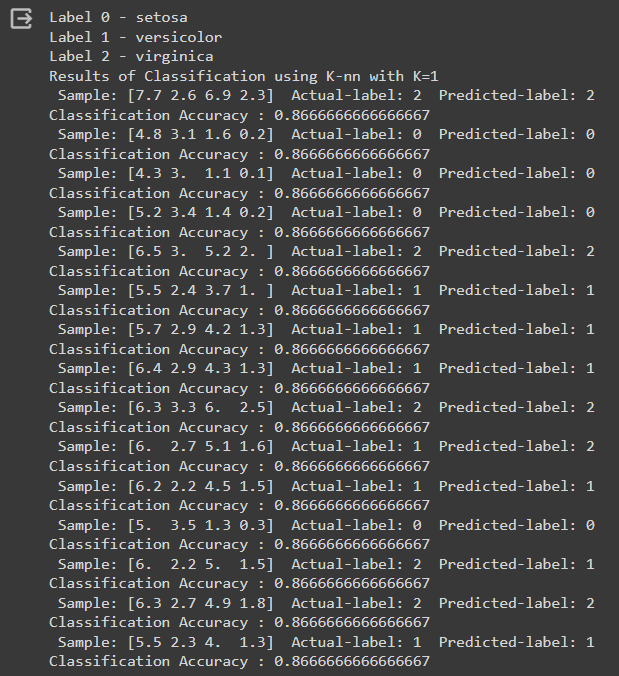
print("Results of Classification using K-nn with K=1 ")

for r in range(0,len(x\_test)):

print(" Sample:", str(x\_test[r]), " Actual-label:", str(y\_test[r])," Predicted-label:", str(y\_pred[r]))

print("Classification Accuracy :" , classifier.score(x\_test,y\_test))

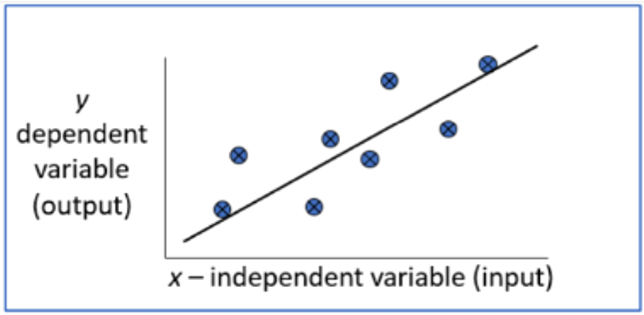
**Output:**



**Week-12: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.**

**Aim:** Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

**Description:** LWLR manifests as a non-parametric regression algorithm that discerns the connection between a dependent variable and several independent variables. LWLR functions on the premise that the association between the dependent and independent variables adheres to linearity; however, this relationship is allowed to exhibit variability across distinct sections within the dataset. This is achieved by employing an individual linear regression model for each prediction, employing a weighted least squares technique.



**Loess Regression:** Loess regression is a nonparametric technique that uses local weighted regression to fit a smooth curve through points in a scatter plot. Loess curves are can reveal trends and cycles in data that might be difficult to model with a parametric curve.

**Program:**

from math import ceil

import numpy as np

from scipy import linalg

def lowess(x, y, f, iterations):

n = len(x)

r = int(ceil(f \* n))

h = [np.sort(np.abs(x - x[i]))[r] for i in range(n)]

w = np.clip(np.abs((x[:, None] - x[None, :]) / h), 0.0, 1.0)

w = (1 - w \*\* 3) \*\* 3

yest = np.zeros(n)

delta = np.ones(n)

for iteration in range(iterations):

for i in range(n):

weights = delta \* w[:, i]

b = np.array([np.sum(weights \* y), np.sum(weights \* y \* x)])

A = np.array([[np.sum(weights), np.sum(weights \* x)],[np.sum(weights \* x), np.sum(weights \* x \* x)]])

beta = linalg.solve(A, b)

yest[i] = beta[0] + beta[1] \* x[i]

residuals = y - yest

s = np.median(np.abs(residuals))

delta = np.clip(residuals / (6.0 \* s), -1, 1)

delta = (1 - delta \*\* 2) \*\* 2

return yest

import math

import matplotlib.pyplot as plt

n = 100

x = np.linspace(0, 2 \* math.pi, n)

y = np.sin(x) + 0.3 \* np.random.randn(n)

f = 0.25

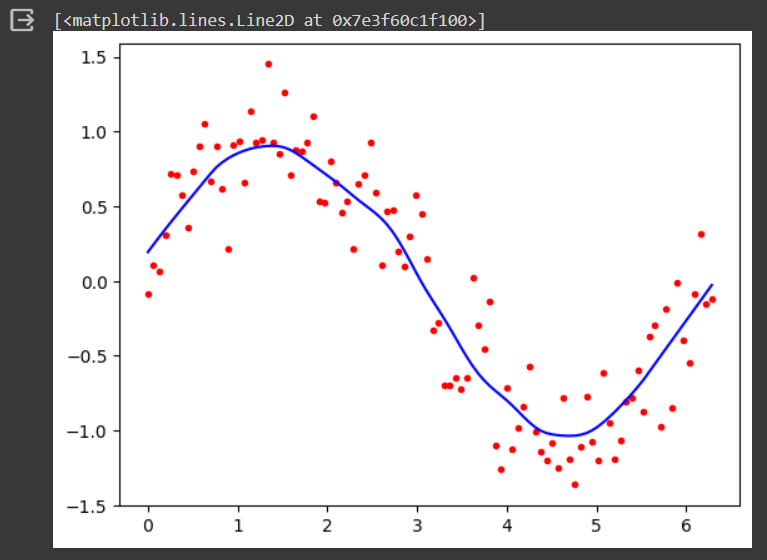
iterations=3

yest = lowess(x, y, f, iterations)

plt.plot(x,y,"r.")

plt.plot(x,yest,"b-")

**Output:**



**Augmented Experiments**

**Experiment-13: Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.**

**Aim:** Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

**Program:**

from sklearn.datasets import fetch\_20newsgroups

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci.med']

twenty\_train = fetch\_20newsgroups (subset='train', categories=categories, shuffle=True)

twenty\_test = fetch\_20newsgroups (subset='test', categories=categories, shuffle=True)

from sklearn.feature\_extraction.text import CountVectorizer

count\_vect = CountVectorizer()

X\_train\_tf = count\_vect.fit\_transform(twenty\_train.data)

from sklearn.feature\_extraction.text import TfidfTransformer

tfidf\_transformer = TfidfTransformer()

X\_train\_tfidf = tfidf\_transformer.fit\_transform(X\_train\_tf)

X\_train\_tfidf.shape

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import accuracy\_score

from sklearn import metrics

mod = MultinomialNB()

mod.fit(X\_train\_tfidf, twenty\_train.target)

X\_test\_tf = count\_vect.transform(twenty\_test.data)

X\_test\_tfidf = tfidf\_transformer.transform(X\_test\_tf)

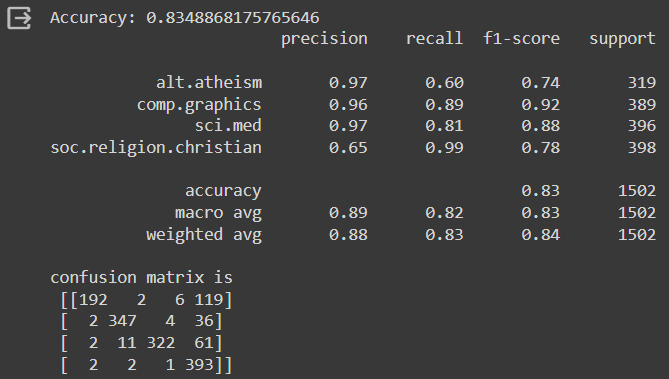
predicted = mod.predict(X\_test\_tfidf)

print("Accuracy:", accuracy\_score(twenty\_test.target, predicted))

print(classification\_report(twenty\_test.target, predicted, target\_names=twenty\_test.target\_names))

print("confusion matrix is \n", metrics.confusion\_matrix(twenty\_test.target, predicted))

**Output:**



**Experiment-14: Apply EM algorithm to cluster a Heart Disease Data Set. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.**

**Aim:** Apply EM algorithm to cluster a Heart Disease Data Set. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

**Program:**

from sklearn.cluster import KMeans

from sklearn import preprocessing

from sklearn.mixture import GaussianMixture

from sklearn.datasets import load\_iris

import sklearn.metrics as sm

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

dataset=load\_iris()

X=pd.DataFrame (dataset.data)

X.columns=['Sepal\_Length', 'Sepal\_width', 'Petal\_Length', 'Petal\_width']

y=pd.DataFrame(dataset.target)

y.columns=['Targets']

plt.figure(figsize=(14,7))

colormap=np.array(['red', 'lime', 'black'])

plt.subplot(1,3,1)

plt.scatter(X.Petal\_Length,X.Petal\_width,c=colormap[y.Targets],s=40)

plt.title('Real')

plt.subplot(1,3,2)

model=KMeans(n\_clusters=3)

model.fit(X)

predY=np.choose(model.labels\_,[0,1,2]).astype(np.int64)

plt.scatter(X.Petal\_Length, X. Petal\_width, c=colormap[predY], s=40)

plt.title('KMeans')

scaler=preprocessing.StandardScaler()

scaler.fit(X)

xsa=scaler.transform(X)

xs=pd.DataFrame(xsa,columns=X.columns)

gmm=GaussianMixture(n\_components=3)

gmm.fit(xs)

y\_cluster\_gmm=gmm.predict(xs)

plt.subplot(1,3,3)

plt.scatter(X.Petal\_Length, X.Petal\_width, c=colormap[y\_cluster\_gmm],s=40)

plt.title('GMM Classification')

**Output:**

