**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:** 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 csv file.

**Description:** To find the maximally specific hypothesis from the set of hypothesis space.

Method: Begin with most specific possible hypothesis in H, then generalize this hypothesis each time it fails to cover an observed positive training example. Notations:

* D - Training data set
* X - Set of instances with in training data set
* x - particular instance in training example
* H - Set of possible hypothesis
* h - particular hypothesis described by conjunction of constraints on the attributes
* ai - constraint attribute of hypothesis, ai can have a value of 0 (no value), or any value(ai=sunny), or ?(any value)
* c - target concept

Algorithm: 1. Initialize h to the most specific hypothesis in H

1. For each positive training instance x

* For each attribute constraint ai is satisfied by x
* Then do nothing
* Else replace ai in h by the next more general constraint that is satisfied by x

3. Output hypothesis h

**Program:**

import csv

#!usr/bin/python #list creatin

hypo=['%','%','%','%','%','%']

with open('Training\_examples.csv') as csv\_file: readcsv = csv.reader(csv\_file, delimiter=',') print(readcsv)

data=[]

print("\nThe given training examples are:")

for row in readcsv:

print(row)

if row[len(row)-1] =='Yes':

data.append(row)

print("\nThe positive examples are:")

for x in data:

print(x)

print("\n")

TotalExamples=len(data)

i=0

j=0

k=0

print("The steps of the Find-s algorithm are\n",hypo)

list =[]

p=0

d=len(data[p])-1

for j in range(d):

list.append(data[i][j]) hypo=list

for i in range(1,TotalExamples):

for k in range(d):

if hypo[k]!=data[i][k]:

hypo[k]='?'

else:

hypo[k]

print(hypo)

print("---------------------------- ")

print("\nThe maximally specific Find-s hypothesis for the given training examples is");

list=[]

for i in range(d):

list.append(hypo[i])

print(list)

**OUTPUT**

The given training examples are:

['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']The positive examples are:

['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']

['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']

['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']

The steps of the Find-s algorithm are

['%', '%', '%', '%', '%', '%']

['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

['Sunny', 'Warm', '?', 'Strong', '?', '?']

The maximally specific Find-s hypothesis for the given training examples is

['Sunny', 'Warm', '?', 'Strong', '?', '?']

**Week - 2**

**For a given set of training data examples stored in a .CSV file, implement and demonstrate the CandidateElimination algorithm to output a description of the set of all hypotheses consistent with the training examples.**

**Aim:** For a given set of training data example stored in .csv file, implement and demonstrate the Candidate-Elimination Algorithm to output and describes the set of all hypotheses consistent with training example.

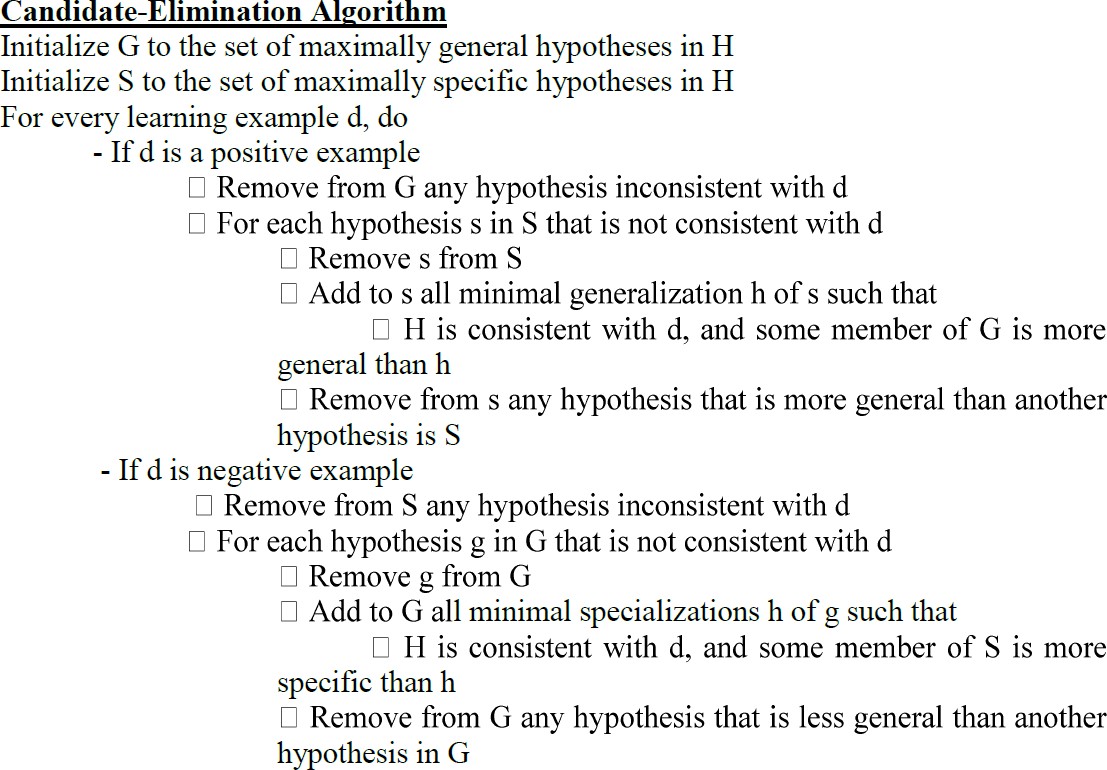
**Description:Candidate-Elimination Learning Algorithm**

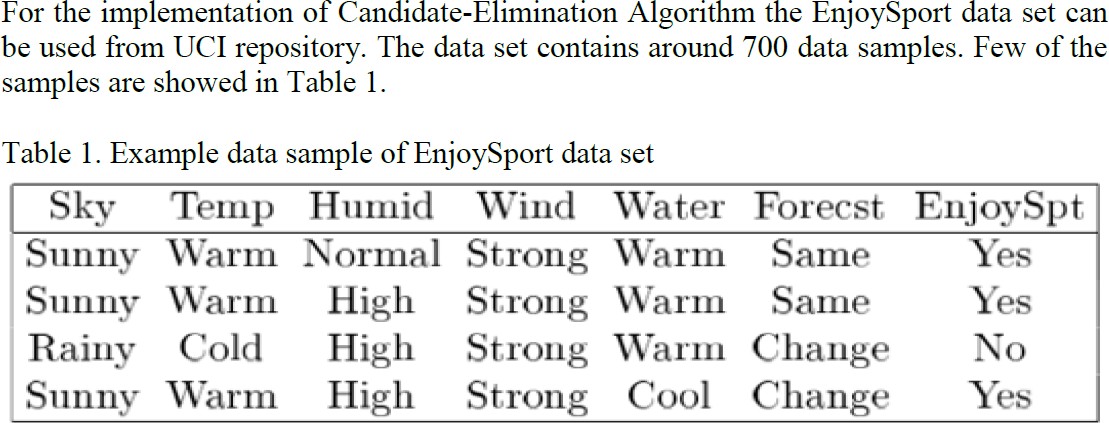
The Candidate-Elimination algorithm computes the version space containing all hypothesis from H that are consistent with an observed sequence of training examples. It begins by initializing the version space to the set of all hypotheses in H; that is by initializing the G boundary set to contain most general hypothesis in H

G0 = {(?, ?, ?, ?)}

Then initialize the S boundary set to contain most specific hypothesis in H

S0 = {(Θ, Θ, Θ, Θ)}

For each training example, these S and G boundary sets are generalized and specialized, respectively, to eliminate from the version space any hypothesis found inconsistent with the new training examples. After execution of all the training examples, the computed version space contains all the hypotheses consistent with these training examples. The algorithm is summarized as below:



**Program:**

import numpy as np

import pandas as pd

data = pd.DataFrame(data=pd.read\_csv('trainingexamples.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 ofspecific\_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] == "Y":

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

if h[x]!= specific\_h[x]:

specific\_h[x] ='?'

general\_h[x][x] ='?'

print(specific\_h)

print(specific\_h)

if target[i] == "N":

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")

#data.head()

**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']]

['Y' 'Y' 'N' 'Y']

initialization of specific\_h and general\_h

['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

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

['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

steps of Candidate Elimination Algorithm 1

['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

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

['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

steps of Candidate Elimination Algorithm

2['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

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

['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

steps of Candidate Elimination Algorithm

3['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?','?','?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' '?' 'Strong' '?' 'Same']

['Sunny' 'Warm' '?' 'Strong' '?' '?']

['Sunny' 'Warm' '?' 'Strong' '?' '?']

steps of Candidate Elimination Algorithm

4['Sunny' 'Warm' '?' 'Strong' '?' '?']

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?','?','?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific\_h:

['Sunny' 'Warm' '?' 'Strong' '?' '?']

Final General\_h:

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

**Week - 3**

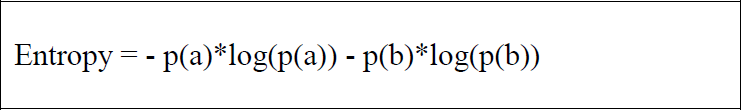
**Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.**

**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.

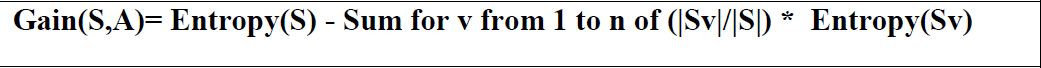
**Description: Following terminologies are used in this algorithm**

* **Entropy:** Entropy is a measure of impurity

It is defined for a binary class with values a/b as:



* **Information Gain :** measuring the expected reduction in Entropy



**THE PROCEDURE**

1) In the ID3 algorithm, begin with the original set of attributes as the root node.

2) On each iteration of the algorithm, iterate through every unused attribute of the remaining set and calculates the entropy (or information gain) of that attribute.

3) Then, select the attribute which has the smallest entropy (or largest information gain) value.

4) The set of remaining attributes is then split by the selected attribute to produce subsets of thedata.

5) The algorithm continues to recurs on each subset, considering only attributes never selectedbefore.

**Dataset Details**

playtennis dataset which has following structure

Total number of instances=15

Attributes=Outlook, Temperature, Humidity, Wind, Answer

Target Concept=Answer

**ID3 ( Learning Sets S, Attributes Sets A, Attributes values V) Return Decision Tree**

Begin

Load learning sets S first, create decision tree root node 'rootNode', add learning set S into root

node as its subset

For rootNode,

1) Calculate entropy of every attribute using the dataset

2) Split the set into subsets using the attribute for which entropy is minimum (or information gain

is maximum)

3) Make a decision tree node containing that attribute

4) Recurse on subsets using renaming attributes

End

This approach employs a top-down, greedy search through the space of possible decision trees.

* Algorithm starts by creating root node for the tree
* If all the examples are positive then return node with positive label
* If all the examples are negative then return node with negative label
* If Attributes is empty, Return the single-node tree Root, with label = most common value
* of Targetattribute in Example
* Otherwise -

1. Calculate the entropy of every attribute using the data set S using formula

**Entropy = - p(a)\*log(p(a)) - p(b)\*log(p(b))**

2. Split the set S into subsets using the attribute for which the resulting entropy (after splitting) is minimum (or, equivalently, information gain is maximum) using formula

**Gain(S,A)= Entropy(S) - Sum for v from 1 to n of (|Sv|/|S|) \* Entropy(Sv)**

3. Make a decision tree node containing that attribute

4. Recurring on subsets using remaining attributes.

**Program:**

import ast

import csv

#import sys

import math

import os

def load\_csv\_to\_header\_data(filename):

path = os.path.normpath(os.getcwd() + filename)

''' os.path.normpath(path)

Normalize a pathname by collapsing redundant separators and up-level references so that A//B,A/B/, A/./B and A/foo/../B all become A/B. This string manipulation may change the meaning of a path that contains symbolic links. On Windows, it converts forward slashes to backward slashes. To normalize case, use normcase().'''

print(path)

fs = csv.reader(open(path))

all\_row = []

for r in fs:

all\_row.append(r)

headers = all\_row[0]

idx\_to\_name, name\_to\_idx = get\_header\_name\_to\_idx\_maps(headers)

data = { 'header': headers,'rows': all\_row[1:],'name\_to\_idx': name\_to\_idx,'idx\_to\_name':

idx\_to\_name}

return data

def get\_header\_name\_to\_idx\_maps(headers):

name\_to\_idx = {}

idx\_to\_name = {}

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

name\_to\_idx[headers[i]] = i

idx\_to\_name[i] = headers[i]

#print(name\_to\_idx)

#print(idx\_to\_name)

return idx\_to\_name, name\_to\_idx

def project\_columns(data, columns\_to\_project):

data\_h = list(data['header'])

data\_r = list(data['rows'])

all\_cols = list(range(0,len(data\_h)))

columns\_to\_project\_ix = [data['name\_to\_idx'][name] for name in columns\_to\_project]

#print(columns\_to\_project\_ix)

columns\_to\_remove = [cidx for cidx in all\_cols if cidx not in columns\_to\_project\_ix]

#print(columns\_to\_remove)

for delc in sorted(columns\_to\_remove, reverse=True):

del data\_h[delc]

for r in data\_r:

del r[delc]

idx\_to\_name, name\_to\_idx = get\_header\_name\_to\_idx\_maps(data\_h)

return {'header': data\_h, 'rows': data\_r,'name\_to\_idx': name\_to\_idx,'idx\_to\_name':

idx\_to\_name}

def get\_uniq\_values(data):

idx\_to\_name = data['idx\_to\_name']

idxs = idx\_to\_name.keys()

#print(idxs)

val\_map = {}

for idx in iter(idxs):

val\_map[idx\_to\_name[idx]] = set()

#print(val\_map)

for data\_row in data['rows']:

for idx in idx\_to\_name.keys():

att\_name = idx\_to\_name[idx]

val = data\_row[idx]

if val not in val\_map.values():

val\_map[att\_name].add(val)

#print(val\_map)

return val\_map

**INPUTS AND OUTPUTS:**

**Input-** Input to the decision algorithm is a dataset stored in .csv file which consists of attributes,examples, target concept.

**Output-** For the given dataset decision tree algorithm produces the decision tree starting with

rootnode which has highest information gain.

{ 'data\_file' : '//tennis.csv', 'data\_mappers' : [], 'data\_project\_columns' : ['Outlook',

'Temperature', 'Humidity', 'Windy', 'PlayTennis'], 'target\_attribute' :'PlayTennis'}

{'data\_file': '//tennis.csv', 'data\_mappers': [], 'data\_project\_columns':['Outlook',

'Temperature', 'Humidity', 'Windy', 'PlayTennis'], 'target\_attribute': 'PlayTennis'}

E:\suthan\codes\Machine Learning Lab\3\tennis.csv

{'Outlook', 'Windy', 'Humidity', 'Temperature'}

IF Outlook EQUALS Sunny AND Humidity EQUALS Normal

THEN YesIF Outlook EQUALS Sunny AND Humidity EQUALS

High THEN No

IF Outlook EQUALS Rainy AND Windy EQUALS True

THEN No IF Outlook EQUALS Rainy AND Windy EQUALS

False THEN YesIF Outlook EQUALS Overcast THEN Yes

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

Aim: Exercises to solve the real-world problems using Linear Regression

Description: Simple linear regression allows us to study the correlation between only two variables:

One variable (X) is called independent variable or predictor.

The other variable (Y), is known as dependent variable or outcome.

and the simple linear regression equation is:

**Y = Β0 + Β1X**

Where:

**X** – the value of the independent variable,

**Y** – the value of the dependent variable.

**Β0** – is a constant (shows the value of Y when the value of X=0)

**Β1** – the regression coefficient (shows how much Y changes for each unit change in X)

**Problem Statement example for Simple Linear Regression:**

Here we are taking a dataset that has two variables: salary (dependent variable) and experience (Independent variable). The goals of this problem is:

* **We want to find out if there is any correlation between these two variables**
* **We will find the best fit line for the dataset.**
* **How the dependent variable is changing by changing the independent variable.**

In this section, we will create a Simple Linear Regression model to find out the best fitting line for representing the relationship between these two variables.

To implement the Simple Linear regression model in machine learning using Python, we need to follow the below steps:

**Step-1: Data Pre-processing**

The first step for creating the Simple Linear Regression model is data pre-processing. We have already done it earlier in this tutorial. But there will be some changes, which are given in the below steps:

* First, we will import the three important libraries, which will help us for loading the dataset, plotting the graphs, and creating the Simple Linear Regression model.

import numpy as nm

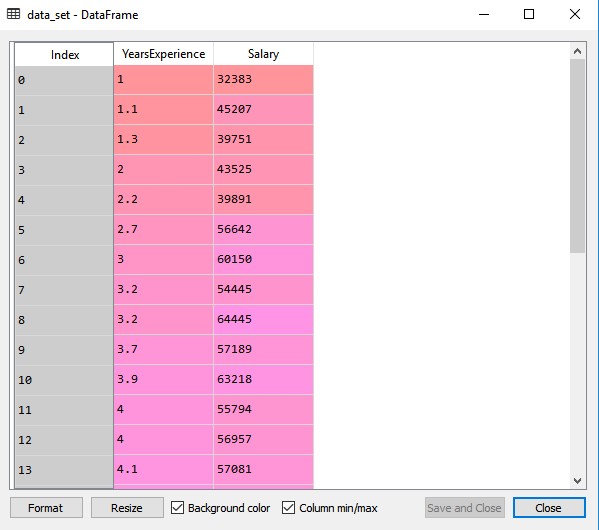
import matplotlib.pyplot as mtp

import pandas as pd

Next, we will load the dataset into our code:

data\_set= pd.read\_csv('Salary\_Data.csv')

By executing the above line of code (ctrl+ENTER), we can read the dataset on our Spyder IDE screen by clicking on the variable explorer option.



The above output shows the dataset, which has two variables: Salary and Experience.

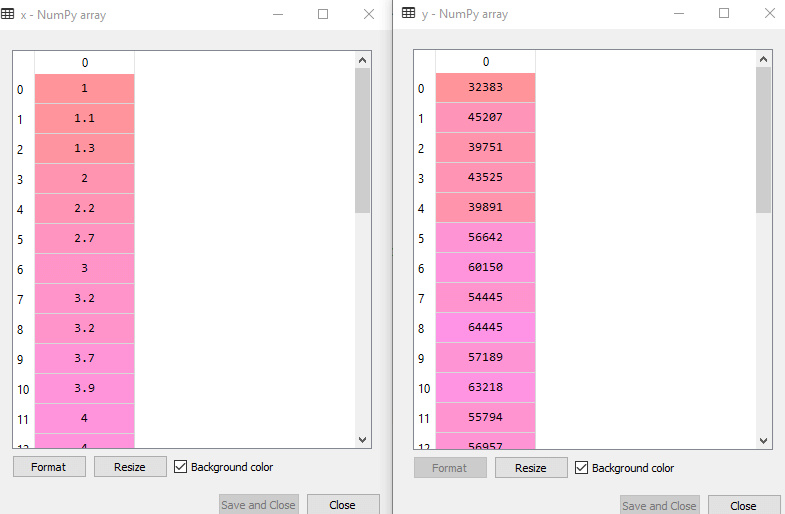
After that, we need to extract the dependent and independent variables from the given dataset. The independent variable is years of experience, and the dependent variable is salary. Below is code for it:

x= data\_set.iloc[:, :-1].values

y= data\_set.iloc[:, 1].values

In the above lines of code, for x variable, we have taken -1 value since we want to remove the last column from the dataset. For y variable, we have taken 1 value as a parameter, since we want to extract the second column and indexing starts from the zero.

By executing the above line of code, we will get the output for X and Y variable as



In the above output image, we can see the X (independent) variable and Y (dependent) variable has been extracted from the given dataset.

Next, we will split both variables into the test set and training set. We have 30 observations, so we will take 20 observations for the training set and 10 observations for the test set. We are splitting our dataset so that we can train our model using a training dataset and then test the model using a test dataset. The code for this is given below:

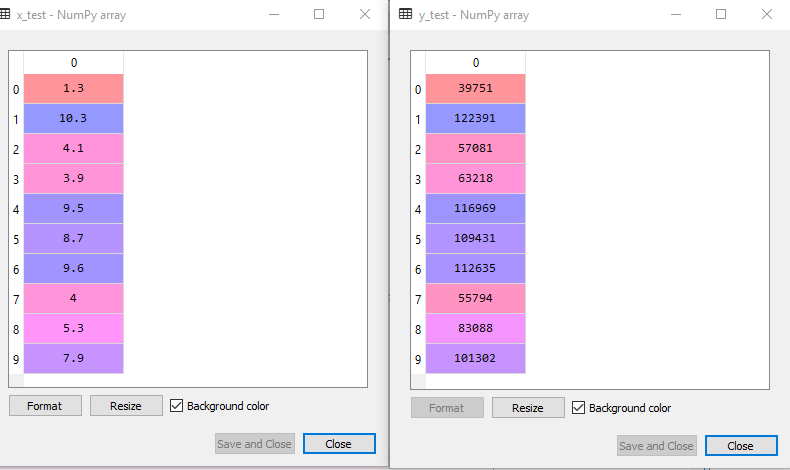
# Splitting the dataset into training and test set.

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

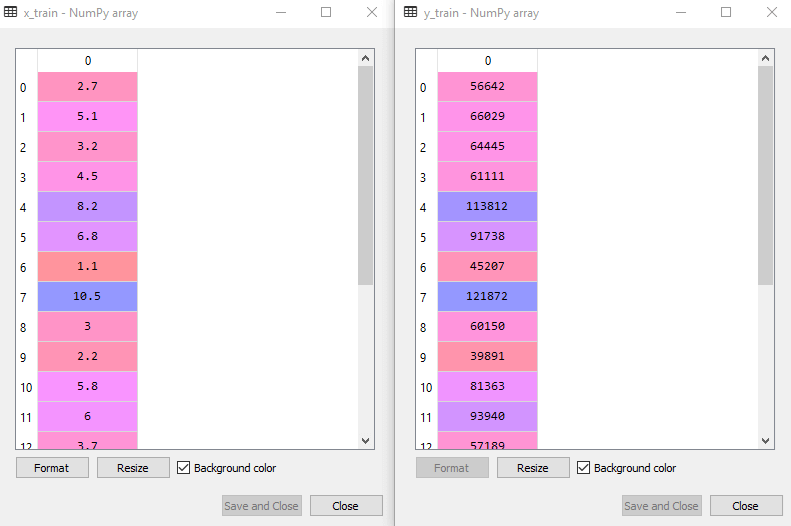
x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 1/3, random\_state=0)

By executing the above code, we will get x-test, x-train and y-test, y-train dataset. Consider the below images:

Test-dataset:

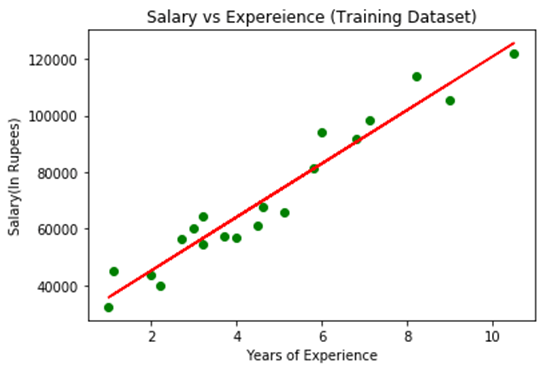


**Training Dataset:**



**Output:**

By executing the above lines of code, we will get the below graph plot as an output.



**Week - 5**

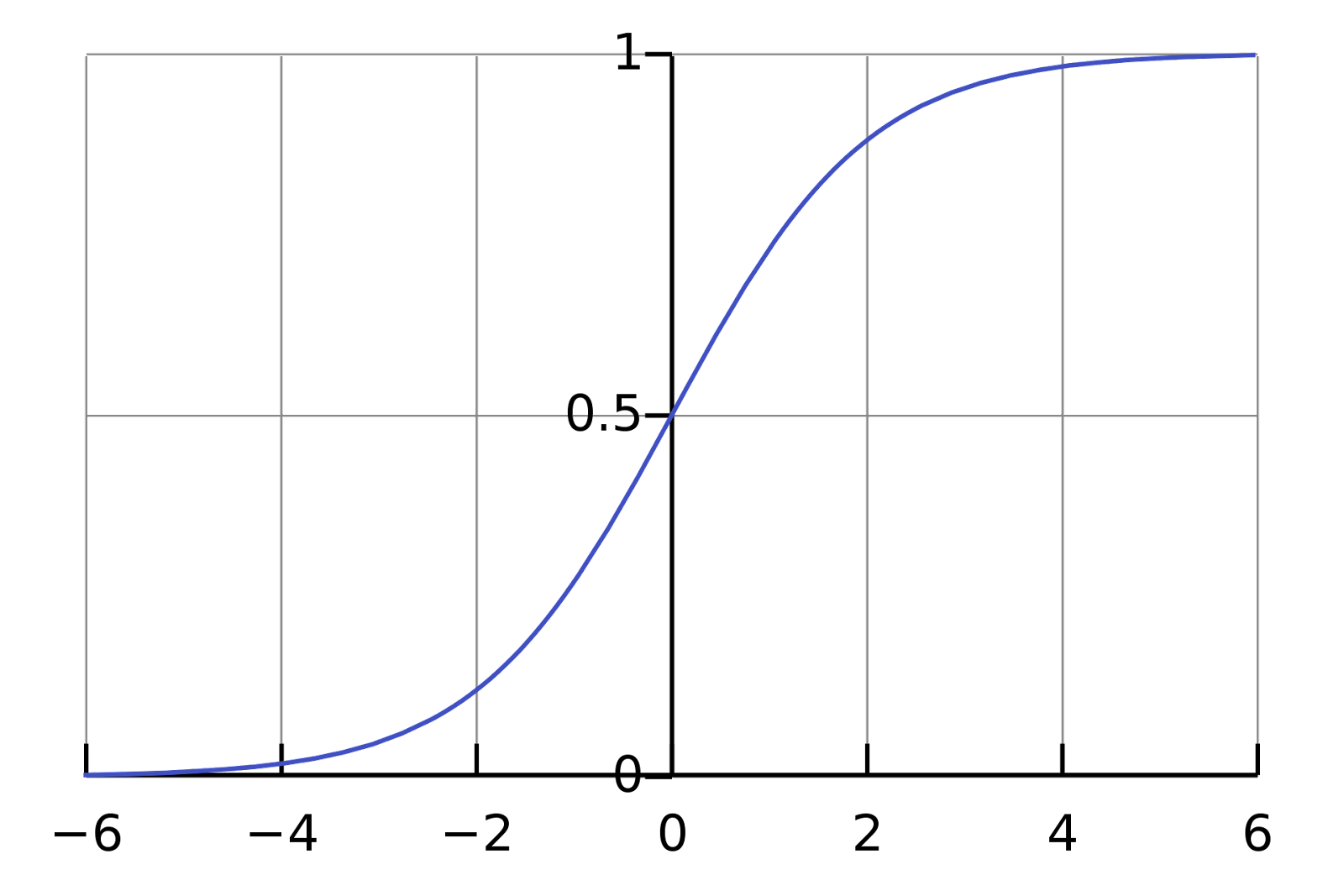
**Exercises to solve the real-world problems using Logistic Regression.**

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

**Description:** Logistic regression is a machine learning method used in the classification problem when you need to distinguish one class from another. The simplest case is a binary classification. This is like a question that we can answer with either “yes” or “no.” We only have two classes: a positive class and negative class. Usually, a positive class points to the presence of some entity while negative class points to the absence of it.

In this case, we need to predict a single value - the probability that entity is present. To do so, it will be good for us to have a function that maps any real value to value in the interval between 0 and 1.

Let’s look at this function plot.



It shows a pretty decent mapping between R and the (0, 1) interval. It suits our requirements.

This is the so-called sigmoid function and it is defined this way:

Most far from 0 values of x are mapped close to 0 or close to 1 values of y. Values close to 0 of x will be a good approximation of probability in our algorithm. Then we can choose a threshold value and transform probability to 0 or 1 prediction.

Sigmoid is an activation function for logistic regression. Now let’s define the cost function for our optimization algorithm.

The first thing that comes into mind when we think about cost function is a classic square error function.

Where

m - number of examples,

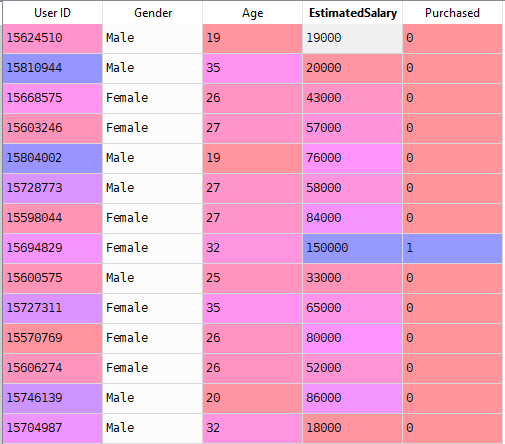
x(i) - feature vector for i-th example,

y(i) - actual value for i-th example,

θ - parameters vector.

**REAL-WORLD EXAMPLE WITH PYTHON:**

Now we’ll solve a real-world problem with Logistic Regression. We have a Data set having 5 columns namely: User ID, Gender, Age, EstimatedSalary and Purchased. Now we have to build a model that can predict whether on the given parameter a person will buy a car or not.



**Steps To Build the Model:**

1. Importing the libraries

Here we’ll import libraries which will be needed to build the model.

import numpy as np

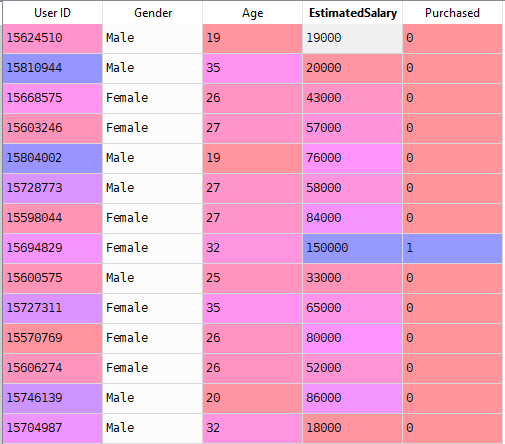
import matplotlib.pyplot as plt

import pandas as pd

2. Importing the Data set

We’ll import our Data set in a variable (i.e dataset) using pandas.

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



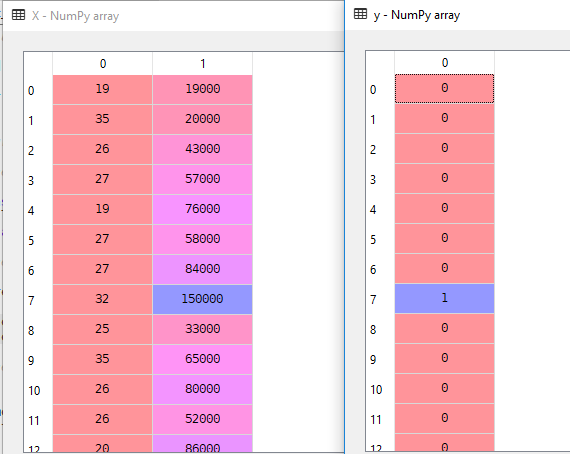
**3. Splitting our Data set in Dependent and Independent variables.**

In our Data set we’ll consider **Age** and  **EstimatedSalary**as Independent variable and **Purchased** as Dependent Variable.

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

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

Here **X** is Independent variable and **y**is Dependent variable.

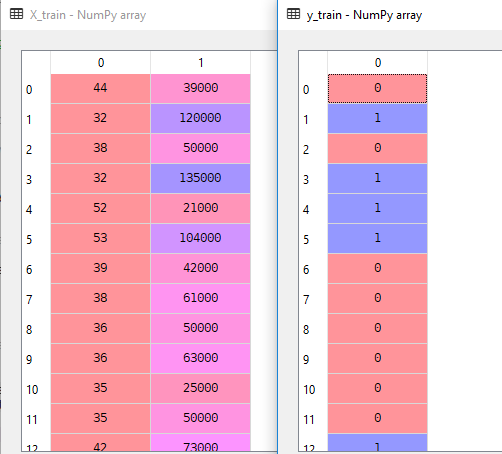


**3. Splitting the Data set into the Training Set and Test Set**

Now we’ll split our Data set into Training Data and Test Data. Training data will be used to train our  
Logistic model and Test data will be used to validate our model. We’ll use **Sklearn** to split our data. We’ll import  **train\_test\_split** from **sklearn.model\_selection**

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)



**4. Feature Scaling**

Now we’ll do feature scaling to scale our data between 0 and 1 to get better accuracy.  
Here Scaling is important because there is a huge difference between **Age**and **EstimatedSalay.**

Import **StandardScaler** from **sklearn.preprocessing**

Then make an instance **sc\_X** of the object **StandardScaler**

Then fit and transform **X\_train** and transform **X\_test**

from sklearn.preprocessing import StandardScaler

sc\_X = StandardScaler()

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

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



**5. Fitting Logistic Regression to the Training Set**

Now we’ll build our classifier (Logistic).

Import **LogisticRegression** from**sklearn.linear\_model**

Make an instance **classifier** of the object **LogisticRegression** and give  
**random\_state =  0** to get the same result every time.

Now use this classifier to fit **X\_train** and **y\_train**

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state=0)

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

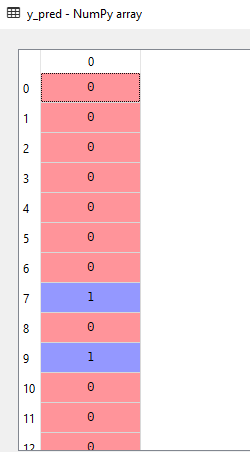
Cheers!! After executing the above command you’ll have a classifier that can predict whether a person will buy a car or not.

Now use the **classifier** to make the prediction for the Test Data set and find the accuracy using Confusion matrix.

**6. Predicting the Test set results**

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

Now we’ll get **y\_pred**



Now we can use **y\_test** (Actual Result) and **y\_pred** ( Predicted Result) to get the accuracy of our model.

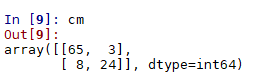
**7. Making the Confusion Matrix**

Using Confusion matrix we can get accuracy of our model.

from sklearn.metrics import confusion\_matrix

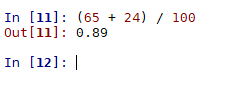
cm = confusion\_matrix(y\_test, y\_pred)

You’ll get a matrix  **cm** .



**Use cm to calculate accuracy as shown below:**

**Accuracy**=**(**cm[0][0] **+** cm[1][1]**) /** **(** Total test data points **)**



Here we are getting accuracy of 89 % . Cheers!! we are getting a good accuracy.

Finally, we’ll Visualise our Training set result and Test set result. We’ll use matplotlib to plot our Data set.

**Visualizing the Training Set result**

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)

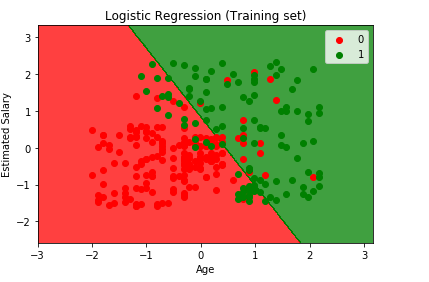
plt.title('Logistic Regression (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()



**Visualizing the Test Set result**

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)

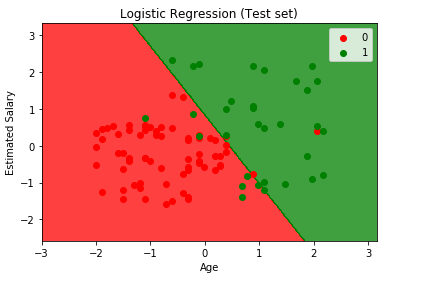
plt.title('Logistic Regression (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()



**Week - 6**

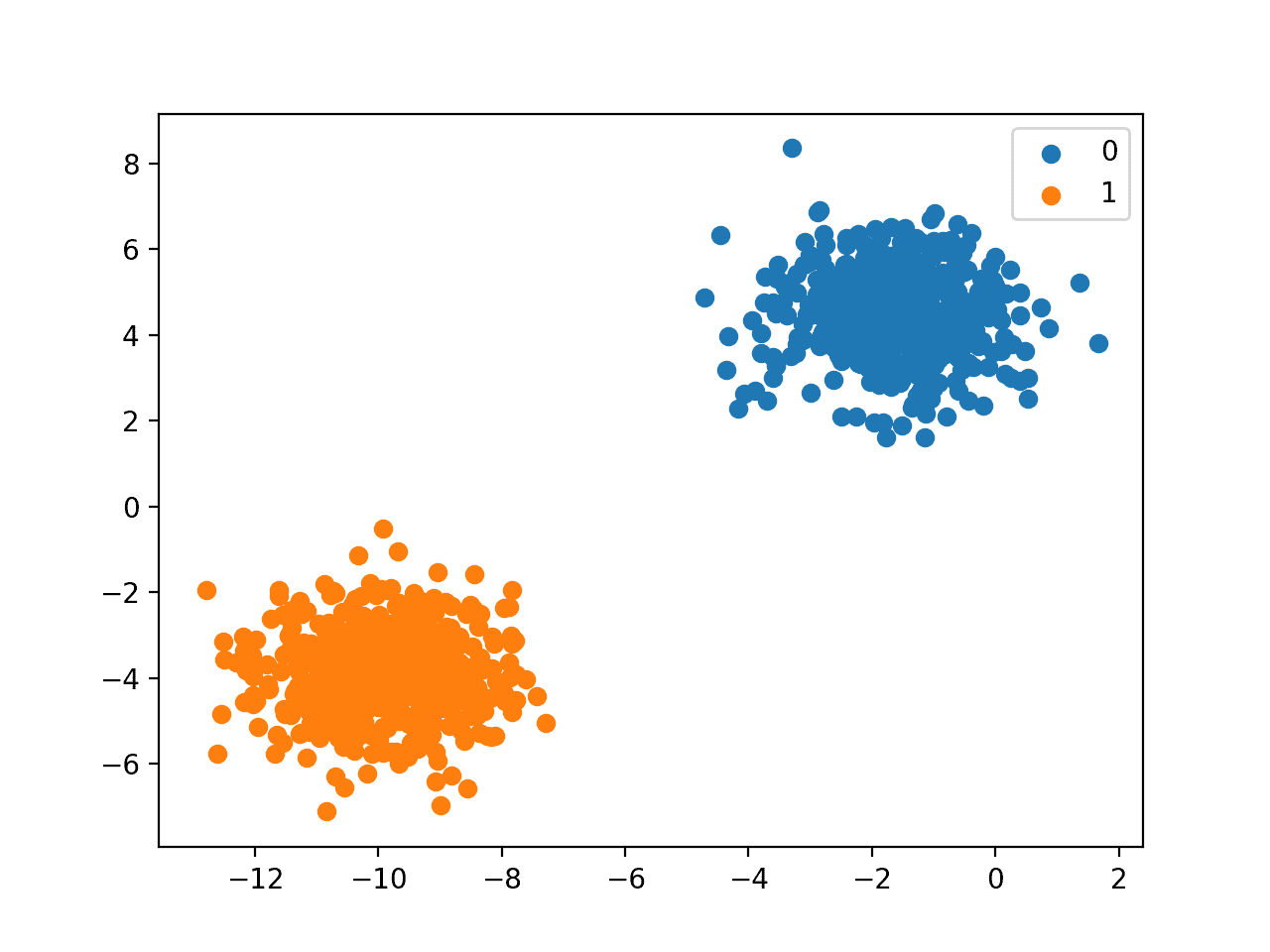
**Exercises to solve the real-world problems using Binary Classifier**

**Aim:** Exercises to solve the real-world problems using Binary Classifier

**Description:**

Binary classification is the task of classifying the elements of a set into two groups on the basis of a classification rule.

In the below graph, we have two classes ‘0’ and ‘1’, where 0→ is represented with blue and 1→ with orange color. Based on the classes a graph is plotted to differentiate two classes.



**Program:**

The first and foremost step to build any model is to import libraries like pandas, sklearn. Keras and Tensorflow are mostly used libraries to build a model because they provide better accuracy and fewer entropy values.

from flask import Flask, render\_template, request

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes import MultinomialNB

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

**Data Set Description**

So let’s take an easy example to implement a binary classification algorithm. Here we are going to consider the Spam classifier model. Here we have 87% of “Not Spam” and 13% of “Spam”.

**Load the Dataset**

In the below code we are loading our data set and labeling our features as “Spam” → 1 and “Not spam/ Ham” → 0.

df = pd.read\_csv("spam.csv", encoding="latin-1")

df.drop(['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], axis=1, inplace=True)

# Features and Labels

df['label'] = df['class'].map({'ham': 0, 'spam': 1})

X = df['message']

y = df['label']

**Extract Feature With CountVectorizer**

Generally, whenever a data set is taken, it always needs to be cleaned. In terms of machine learning, we need to extract the important features from the dataset. To extract the features here we are using the CountVectorizer algorithm.

# Extract Feature With CountVectorizer

   cv = CountVectorizer()

**Fit the Data**

To fit the data into a model we need to split our data set into train and test data. In general, the data is divided in the ratio of 8:2.

X = cv.fit\_transform(X)  # Fit the Data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42)

**Naive Bayes Classifier**

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.  
Naive Bayes uses a similar method to predict the probability of different classes based on various attributes. This algorithm is mostly used in text classification and with problems having multiple classes.

To train the Spam Classifier we are using Naive Bayes as this is one of the best algorithms for classifying.

# Naive Bayes Classifier

 clf = MultinomialNB()

 clf.fit(X\_train, y\_train)

 clf.score(X\_test, y\_test)

 if request.method == 'POST':

     message = request.form['message']

     data = [message]

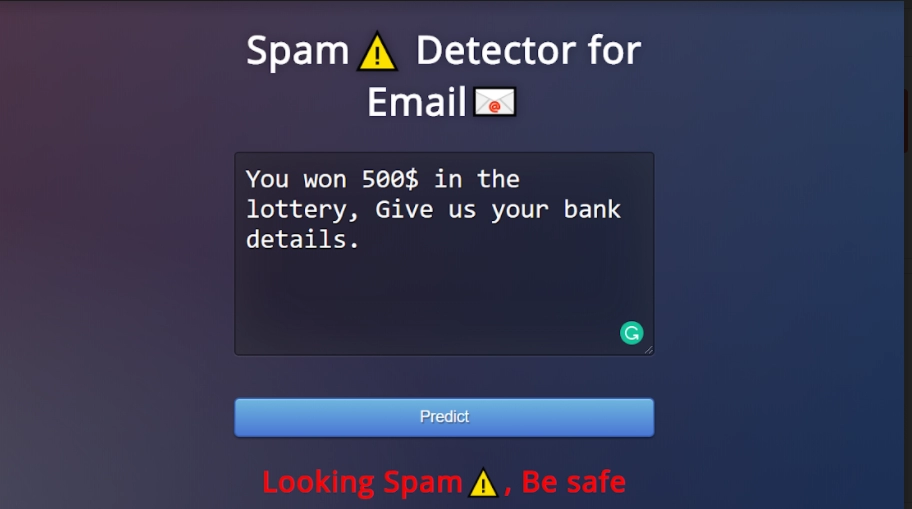
     vect = cv.transform(data).toarray()

      my\_prediction = clf.predict(vect)

 return render\_template('index.html', prediction=my\_prediction)

This code snippet returns the prediction of your given text and Spam classifier.

**Sample Output:**



**Visualizing the Output and Data**

**Visualizing data is the most important part while developing any machine learning model.**

**Sample Output**

import matplotlib.pyplot as plt

%matplotlib inline

import seaborn as sns

sns.set()

acc = hist.history['accuracy']

val = hist.history['val\_accuracy']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, '-', label='Training accuracy')

plt.plot(epochs, val, ':', label='Validation accuracy')

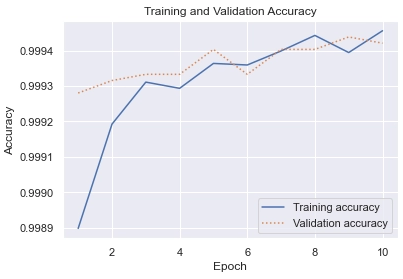
plt.title('Training and Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend(loc='lower right')

plt.plot()



**Week - 7**

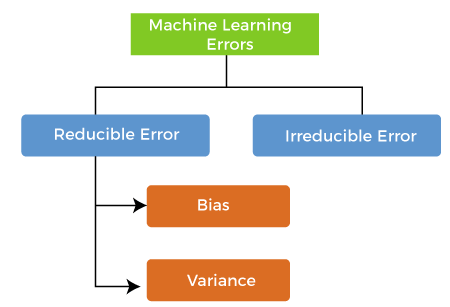
**Develop a program for Bias, Variance, Remove duplicates , Cross Validation.**

**Aim:** Develop a program for Bias, Variance, Remove duplicates , Cross Validation.

**Description:** In machine learning, an error is a measure of how accurately an algorithm can make predictions for the previously unknown dataset. On the basis of these errors, the machine learning model is selected that can perform best on the particular dataset. There are mainly two types of errors in machine learning, which are:

**Reducible errors:** These errors can be reduced to improve the model accuracy. Such errors can further be classified into bias and Variance.

**Irreducible errors:** These errors will always be present in the model



**1.BIAS:**

While making predictions, a difference occurs between prediction values made by the model and actual values/expected values, and this difference is known as bias errors or Errors due to bias.

**Low Bias:** A low bias model will make fewer assumptions about the form of the target function.

**High Bias:** A model with a high bias makes more assumptions, and the model becomes unable to capture the important features of our dataset. **A high bias model also cannot perform well on new data.**

**2.VARIANCE ERROR:**

The variance would specify the amount of variation in the prediction if the different training data was used. In simple words, variance tells that how much a random variable is different from its expected value. Ideally, a model should not vary too much from one training dataset to another, which means the algorithm should be good in understanding the hidden mapping between inputs and output variables. Variance errors are either of low variance or high variance.

Low variance means there is a small variation in the prediction of the target function with changes in the training data set. At the same time, High variance shows a large variation in the prediction of the target function with changes in the training dataset.

First, you must install the mlxtend library; for example:

**sudo pip install mlxtend**

The example below loads the Boston housing dataset directly via URL, splits it into train and test sets, then estimates the mean squared error (MSE) for a linear regression as well as the bias and variance for the model error over 200 bootstrap samples.

# estimate the bias and variance for a regression model

from pandas import read\_csv

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

from sklearn.linear\_model import LinearRegression

from mlxtend.evaluate import bias\_variance\_decomp

# load dataset

url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv'

dataframe = read\_csv(url, header=None)

# separate into inputs and outputs

data = dataframe.values

X, y = data[:, :-1], data[:, -1]

# split the data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=1)

# define the model

model = LinearRegression()

# estimate bias and variance

mse, bias, var = bias\_variance\_decomp(model, X\_train, y\_train, X\_test, y\_test, loss='mse', num\_rounds=200, random\_seed=1)

# summarize results

print('MSE: %.3f' % mse)

print('Bias: %.3f' % bias)

print('Variance: %.3f' % var)

**OUTPUT:**

MSE: 22.487

Bias: 20.726

Variance: 1.761

**3.Removing Duplicates**

Duplicate entries are problematic for multiple reasons. An entry appearing more than once receives disproportionate weight during training. Models that succeed on frequent entries only *look* like they perform well. Duplicate entries can ruin the split between [train, validation, and test sets](https://deepchecks.com/training-validation-and-test-sets-what-are-the-differences/) where identical entries are not all in the same set. This can lead to biased performance estimates that result in disappointing the model in production.

df

FirstName LastName PhoneNo

0 A B 1

1 A B 1

2 A B 2

df.drop\_duplicates(subset=["FirstName", "LastName"])

FirstName LastName PhoneNo

0 A B 1

**4.CROSS VALIDATION:**

Cross-validation is a technique for evaluating ML models by training several ML models on subsets of the available input data and evaluating them on the complementary subset of the data. Use cross-validation to detect overfitting, ie, failing to generalize a pattern.

**Methods used for Cross-Validation:**

**Validation Set Approach**

**Leave-P-out cross-validation**

**Leave one out cross-validation**

**K-fold cross-validation**

**Stratified k-fold cross-validation**

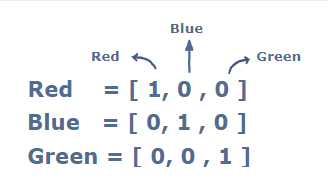
**Week - 8**

**Write a program to implement One-hot Encoding.**

**Aim:** Write a program to implement One-hot Encoding

**Description**: Most of the existing machine learning algorithms cannot be executed on categorical data. Instead, the categorical data needs to first be converted to numerical data. One-hot encoding is one of the techniques used to perform this conversion. This method is mostly used when deep learning techniques are to be applied to​ sequential classification problems.

One-hot encoding is essentially the representation of categorical variables as binary vectors. These categorical values are first mapped to integer values. Each integer value is then represented as a binary vector that is all 0s (except the index of the integer which is marked as 1).



**Program:**

import numpy as np

### Categorical data to be converted to numeric data

colors = ["red", "green", "yellow", "red", "blue"]

### Universal list of colors

total\_colors = ["red", "green", "blue", "black", "yellow"]

### map each color to an integer

mapping = {}

for x in range(len(total\_colors)):

  mapping[total\_colors[x]] = x

one\_hot\_encode = []

for c in colors:

  arr = list(np.zeros(len(total\_colors), dtype = int))

  arr[mapping[c]] = 1

  one\_hot\_encode.append(arr)

print(one\_hot\_encode)

**OUTPUT:**

[[1, 0, 0, 0, 0], [0, 1, 0, 0, 0], [0, 0, 0, 0, 1], [1, 0, 0, 0, 0], [0, 0, 1, 0, 0]]

**Week – 9**

**Write a program to implement Categorical Encoding.**

**Aim:** Write a program to implement Categorical Encoding

**Description:** Categorical data is a common type of non-numerical data that contains label values and not numbers. Some examples include:

Colors: White, Black, Green. Cities: Mumbai, Pune, Delhi. Gender: Male, Female.

In order to various encoding techniques we are going to use the below dataset:

# importing libraries

import pandas as pd

# creating dataset

data = {'Subject': ['s1', 's2', 's3', 's1', 's4',

's3', 's2', 's1', 's2', 's4', 's1'],

'Target': [1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0]}

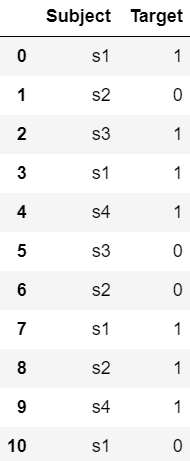
# convert to dataframe

df = pd.DataFrame(data)

# display the dataset

Df

**OUTPUT:**



**Week - 10**

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

**Aim:** Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

**Description:** Artificial neural networks (ANNs) are powerful tools for machine learning with applications in many areas including speech recognition, image classification, medical diagnosis, and spam filtering. It has been shown that ANNs can approximate any function to any degree of accuracy given enough neurons and training time. However, there is no guarantee on the number of neurons required or the time it will take to train them. These are the main disadvantages of using ANNs. Here we develop the BackPropogation algorithm which learns the weights for a multilayer network, given a network with a fixed set of units and interconnections. It employs gradient descent to attempt to minimize the squared error between the network output values and target values for these outputs.

**BACK PROPAGATION ALGORITHM:**

Multiple layer perceptron is effectively applied to handle tricky problems if trained with a vastly accepted algorithm identified as the back-propagation algorithm (error) in a supervised manner. It functions on learning law with error-correction. It is also a simplified version for the least mean square (LMS) filtering algorithm which is equally popular to error back-propagation algorithm.

In Error back-propagation training there are two computational passes via several network layers:

In forward pass, vector input is applied to the nodes of the system propagating each layer„s outcome to the next layer via network. To get the accurate response of the network, these outputs pass on from several layers and arrive at a set of outputs. In forward pass network weights are permanent. On other hand in the backward pass, weights are adjusted according to rule for error correction. Error signal is the actual response of the network minus the desired response.

The propagation of this error signal through the network is towards backward in direction opposite to the connections of synaptic. The move the real response of network closer to the favored response, tuning of weights is to be done. There are three unique features of a multilayer perception:

1)For each neuron in any system, its illustration has an activation function that is non-linear. The logistical function is used to define a function which is sigmoid.

2)There are layer(s) of hidden neurons not contained in the input or the output present in the neural network. The study over complex tasks is facilitated by these hidden neurons.

3)Connectivity degree is high in network. Weight's population should be changed if there is a requirement to alter the connectivity of the network.

The stochastic gradient descent version of the BACKPROPAGATION algorithm for feed forward networks containing two layers of sigmoid units.

**Step 1:** begins by constructing a network with the desired number of hidden and output units and initializing all network weights to small random values. . For each training example, it applies the network to the example, calculates the error of the network output for this example, computes the gradient with respect to the error on this example, then updates all weights in the network. This gradient descent step is iterated (often thousands of times, using the same training examples multiple times) until the network performs acceptably well.

**Step 2:** The gradient descent weight-update rule is similar to the delta training rule The only difference is that the error (t - o) in the delta rule is replaced by a more complex error term aj.

**Step 3:** updates weights incrementally, following the Presentation of each training example. This corresponds to a stochastic approxi- mation to gradient descent. To obtain the true gradient of E one would sum the Sj, xji values over all training examples before altering weight values.

**Step 4:** The weight-update loop in BACKPROPAGATION may be iterated thousands of times in a typical application. A variety of termination conditions can be used to halt the procedure.

One may choose to halt after a fixed number of iterations through the loop, or once the error on the training examples falls below some threshold.

**Program:**

from math import exp from random import seed

from random import random

# Initialize a network

def initialize\_network(n\_inputs, n\_hidden, n\_outputs): network = list()

hidden\_layer = [{'weights':[random() for i in range(n\_inputs + 1)]} for i in range(n\_hidden)] network.append(hidden\_layer)

#print(network)

output\_layer = [{'weights':[random() for i in range(n\_hidden + 1)]} for i in range(n\_outputs)] network.append(output\_layer)

#print(network) return network

# Calculate neuron activation for an input def activate(weights, inputs):

activation = weights[-1]

for i in range(len(weights)-1):

activation += weights[i] \* inputs[i] return activation

# Transfer neuron activation def transfer(activation):

return 1.0 / (1.0 + exp(-activation))

# Forward propagate input to a network output def forward\_propagate(network, row):

inputs = row

for layer in network: #print(layer) new\_inputs = [] for neuron in layer:

activation = activate(neuron['weights'], inputs) neuron['output'] = transfer(activation) new\_inputs.append(neuron['output'])

inputs = new\_inputs #print(inputs)

#print(inputs) return inputs

# Calculate the derivative of an neuron output def transfer\_derivative(output):

return output \* (1.0 - output)

# Backpropagate error and store in neurons

def backward\_propagate\_error(network, expected):

for i in reversed(range(len(network))): layer = network[i]

#print(layer) errors = list()

if i != len(network)-1:

for j in range(len(layer)): error = 0.0

for neuron in network[i + 1]:

error += (neuron['weights'][j] \* neuron['delta']) errors.append(error)

else:

for j in range(len(layer)): neuron = layer[j]

errors.append(expected[j] - neuron['output']) for j in range(len(layer)):

neuron = layer[j]

neuron['delta'] = errors[j] \* transfer\_derivative(neuron['output'])

# Update network weights with error

def update\_weights(network, row, l\_rate): for i in range(len(network)):

inputs = row[:-1] if i != 0:

inputs = [neuron['output'] for neuron in network[i - 1]] for neuron in network[i]:

for j in range(len(inputs)):

neuron['weights'][j] += l\_rate \* neuron['delta'] \* inputs[j] neuron['weights'][-1] += l\_rate \* neuron['delta']

# Train a network for a fixed number of epochs

def train\_network(network, train, l\_rate, n\_epoch, n\_outputs): for epoch in range(n\_epoch):

sum\_error = 0 for row in train:

outputs = forward\_propagate(network, row) #print(outputs)

expected = [0 for i in range(n\_outputs)] #print(expected)

expected[row[-1]] = 1

#print(expected)

sum\_error += sum([(expected[i]-outputs[i])\*\*2 for i in range(len(expected))]) #print(sum\_error)

backward\_propagate\_error(network, expected) update\_weights(network, row, l\_rate)

print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l\_rate, sum\_error))

# Test training backprop algorithm seed(1)

dataset = [[2.7810836,2.550537003,0],

[1.465489372,2.362125076,0],

[3.396561688,4.400293529,0],

[1.38807019,1.850220317,0],

[3.06407232,3.005305973,0],

[7.627531214,2.759262235,1],

[5.332441248,2.088626775,1],

[6.922596716,1.77106367,1],

[8.675418651,-0.242068655,1],

[7.673756466,3.508563011,1]]

n\_inputs = len(dataset[0]) - 1 #print(n\_inputs)

n\_outputs = len(set([row[-1] for row in dataset])) #print(n\_outputs)

network = initialize\_network(n\_inputs, 2, n\_outputs) train\_network(network, dataset, 0.5, 20, n\_outputs) for layer in network:

print(layer)

**OUTPUT**

>epoch=0, lrate=0.500, error=6.350

>epoch=1, lrate=0.500, error=5.531

>epoch=2, lrate=0.500, error=5.221

>epoch=3, lrate=0.500, error=4.951

>epoch=4, lrate=0.500, error=4.519

>epoch=5, lrate=0.500, error=4.173

>epoch=6, lrate=0.500, error=3.835

>epoch=7, lrate=0.500, error=3.506

>epoch=8, lrate=0.500, error=3.192

>epoch=9, lrate=0.500, error=2.898

>epoch=10, lrate=0.500, error=2.626

>epoch=11, lrate=0.500, error=2.377

>epoch=12, lrate=0.500, error=2.153

>epoch=13, lrate=0.500, error=1.953

>epoch=14, lrate=0.500, error=1.774

>epoch=15, lrate=0.500, error=1.614

>epoch=16, lrate=0.500, error=1.472

>epoch=17, lrate=0.500, error=1.346

>epoch=18, lrate=0.500, error=1.233

>epoch=19, lrate=0.500, error=1.132

[{'weights': [-1.4688375095432327, 1.850887325439514, 1.0858178629550297], 'output':

0.029980305604426185, 'delta': -0.0059546604162323625}, {'weights':

[0.37711098142462157, -0.0625909894552989, 0.2765123702642716], 'output':

0.9456229000211323, 'delta': 0.0026279652850863837}]

[{'weights': [2.515394649397849, -0.3391927502445985, -0.9671565426390275],

'output':

0.23648794202357587, 'delta': -0.04270059278364587}, {'weights': [- 2.5584149848484263, 1.0036422106209202, 0.42383086467582715], 'output': 0.7790535202438367, 'delta': 0.03803132596437354}]

**Week - 11**

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

**Aim:** Write a program to implement K-nearest neighbour algorithm to classify iris dataset. Print both correct and wrong predication using python machine learning.

**Description:**

k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression.[1] In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression.

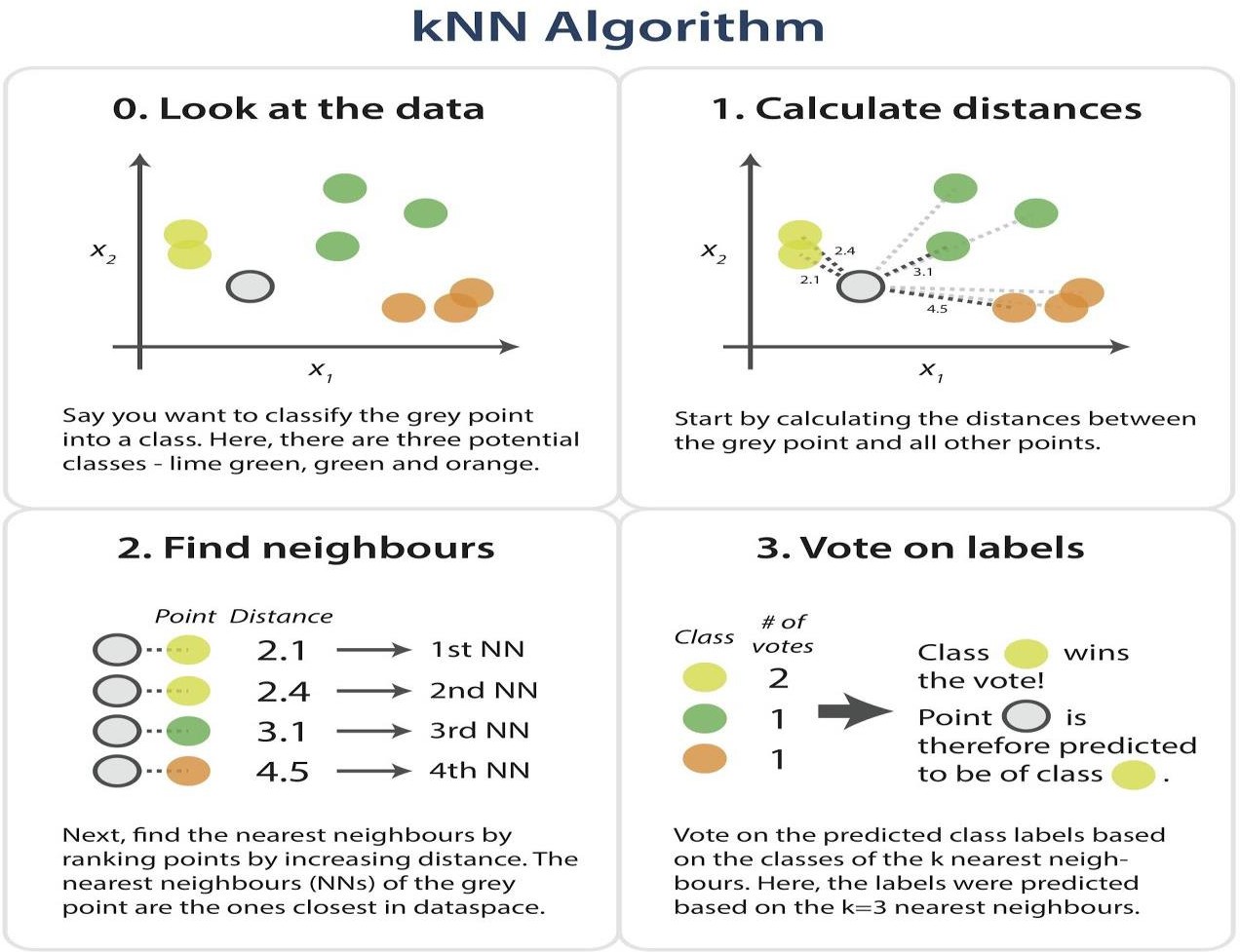
k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms.

The kNN task can be broken down into writing 3 primary functions:

Calculate the distance between any two points

Find the nearest neighbours based on these pair wise distances

Majority vote on a class labels based on the nearest neighbour list



**Dataset**

Iris dataset, consists of flower measurements for three species of iris flower. Our task is to predict the species labels of a set of flowers based on their flower measurements. Since you’ll be building a predictor based on a set of known correct classifications .

The data set contains 3 classes of 151 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

**Predicted attribute:** class of iris plant

**Attribute Information:**

1.sepal length in cm 2. sepal width in cm 3. petal length in cm 4. petal width in cm

**Class: - Iris Setosa - Iris Versicolour - Iris Virginica**

**Program:**

import csv

import random

import math

import operator

def loadDataset(filename, split, trainingSet=[], testSet=[]):

with open(filename) as csvfile:

lines = csv.reader(csvfile)

dataset = list(lines)

for x in range(len(dataset)-1):

for y in range(4):

dataset[x][y] = float(dataset[x][y])

if random.random() < split:

trainingSet.append(dataset[x])

else:

testSet.append(dataset[x])

def euclideanDistance(instance1, instance2, length):

distance = 0

for x in range(length):

distance += pow((instance1[x] - instance2[x]), 2)

return math.sqrt(distance)

def getNeighbors(trainingSet, testInstance, k):

distances = []

length = len(testInstance)-1

for x in range(len(trainingSet)):

dist = euclideanDistance(testInstance, trainingSet[x], length) distances.append((trainingSet[x], dist))

distances.sort(key=operator.itemgetter(1))

neighbors = []

for x in range(k):

neighbors.append(distances[x][0])

return neighbors

def getResponse(neighbors):

classVotes = {}

for x in range(len(neighbors)): response = neighbors[x][-1]

if response in classVotes:

classVotes[response] += 1

else:

classVotes[response] = 1

sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)

return sortedVotes[0][0]

def getAccuracy(testSet, predictions):

correct = 0

for x in range(len(testSet)):

if testSet[x][-1] == predictions[x]:

correct += 1

return (correct/float(len(testSet))) \* 100.0

def main():

# prepare data

trainingSet=[]

testSet=[]

split = 0.67

loadDataset('iris\_data.csv', split, trainingSet, testSet)

print ('\n Number of Training data: ' + (repr(len(trainingSet))))

print (' Number of Test Data: ' + (repr(len(testSet))))

# generate predictions

predictions=[]

k = 3

print('\n The predictions are: ')

for x in range(len(testSet)):

neighbors = getNeighbors(trainingSet, testSet[x], k)

result = getResponse(neighbors)

predictions.append(result)

print(' predicted=' + repr(result) + ', actual=' + repr(testSet[x][-1]))

accuracy = getAccuracy(testSet, predictions)

print('\n The Accuracy is: ' + repr(accuracy) + '%')

main()

**OUTPUT**

Number of Training data: 99

Number of Test Data: 50

The predictions are:

predicted='Iris-setosa', actual='Iris-setosa'

predicted='Iris-setosa', actual='Iris-setosa'

predicted='Iris-setosa', actual='Iris-setosa'

predicted='Iris-versicolor', actual='Iris-versicolor'

predicted='Iris-versicolor', actual='Iris-versicolor'

predicted='Iris-versicolor', actual='Iris-versicolor'

predicted='Iris-versicolor', actual='Iris-versicolor'

predicted='Iris-versicolor', actual='Iris-versicolor'

predicted='Iris-versicolor', actual='Iris-versicolor'

predicted='Iris-versicolor', actual='Iris-versicolor'

predicted='Iris-versicolor', actual='Iris-versicolor'

predicted='Iris-versicolor', actual='Iris-versicolor'

predicted='Iris-versicolor', actual='Iris-versicolor'

predicted='Iris-virginica', actual='Iris-virginica'

predicted='Iris-virginica', actual='Iris-virginica'

predicted='Iris-virginica', actual='Iris-virginica'

predicted='Iris-virginica', actual='Iris-virginica'

predicted='Iris-virginica', actual='Iris-virginica'

**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 to fit data points. Select appropriate data set for your experiment and draw graphs.

**Description: Locally Weighted Regression** –

Nonparametric regression is a category of regression analysis in which the predictor does not take a predetermined form but is constructed according to information derived from the data (training examples).

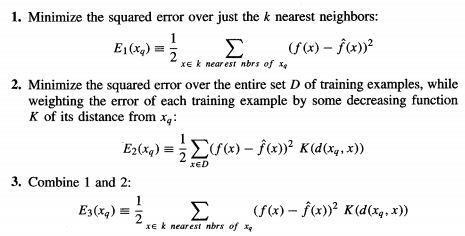
Nonparametric regression requires larger sample sizes than regression based on parametric

models. Because larger the data available, accuracy will be high.

Locally Weighted Linear Regression –

Locally weighted regression is called local because the function is approximated based a only on data near the query point, weighted because the contribution of each training example is weighted by its distance from the query point.

Query point is nothing but the point nearer to the target function , which will help in finding the actual position of the target function.

Let us consider the case of locally weighted regression in which the target function f is

approximated near x, using a linear function of the form.

from math import ceil

import numpy as np from scipy import linalg

def lowess(x, y, f=2./3., iter=3): 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)

**Program:**

import pylab as pl pl.clf()

pl.plot(x, y, label='y noisy') pl.plot(x, yest, label='y pred')

w = (1 - w\*\*3)\*\*3 yest = np.zeros(n) delta = np.ones(n)

for iteration in range(iter): 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

if name == ' main ': import math

n = 100

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

print("==========================values of x=====================")

print(x)

y = np.sin(x) + 0.3\*np.random.randn(n) print("================================Values of y===================")

print(y) f = 0.25

yest = lowess(x, y, f=f, iter=3)

pl.legend()

pl.show()

**OUTPUT**

==========================values of x=====================

[0. 0.06346652 0.12693304 0.19039955 0.25386607 0.31733259

0.38079911 0.44426563 0.50773215 0.57119866 0.63466518 0.6981317

0.76159822 0.82506474 0.88853126 0.95199777 1.01546429 1.07893081

1.14239733 1.20586385 1.26933037 1.33279688 1.3962634 1.45972992

1.52319644 1.58666296 1.65012947 1.71359599 1.77706251 1.84052903

1.90399555 1.96746207 2.03092858 2.0943951 2.15786162 2.22132814

2.28479466 2.34826118 2.41172769 2.47519421 2.53866073 2.60212725

2.66559377 2.72906028 2.7925268 2.85599332 2.91945984 2.98292636

3.04639288 3.10985939 3.17332591 3.23679243 3.30025895 3.36372547

3.42719199 3.4906585 3.55412502 3.61759154 3.68105806 3.74452458

3.8079911 3.87145761 3.93492413 3.99839065 4.06185717 4.12532369

4.1887902 4.25225672 4.31572324 4.37918976 4.44265628 4.5061228

4.56958931 4.63305583 4.69652235 4.75998887 4.82345539 4.88692191

4.95038842 5.01385494 5.07732146 5.14078798 5.2042545 5.26772102

5.33118753 5.39465405 5.45812057 5.52158709 5.58505361 5.64852012

5.71198664 5.77545316 5.83891968 5.9023862 5.96585272 6.02931923

6.09278575 6.15625227 6.21971879 6.28318531]

================================Values of y===================

[ 0.12909628 0.5378001 0.08507775 0.08261955 0.08748326 0.46390454

0.39007129 0.49168683 0.44534231 0.55328598 0.24690547 1.19597387

0.92244303 0.56004488 0.9561929 1.13800942 1.12911587 0.76110236

1.23982502 0.39462141 1.02459433 0.81259471 0.55535331 0.64550225

1.45040127 0.70659902 1.01732347 0.81062276 1.706929 0.73681414

1.26884138 0.76529336 0.36909709 0.85530574 0.90229748 0.87607598

1.36419146 0.88365564 0.58595606 0.51983462 0.2214239 0.07172939

0.18989997 0.51956736 0.51702737 0.35407817 0.02826523 0.04505194

0.20336912 0.13206237 -0.08791493 0.59561087 -0.02677494 -0.17386743

-0.25492254 -0.5663511 -0.38921533 -0.88414287 -0.41859126 -0.23967376

-0.39954993 -1.16303626 -0.31169895 -0.63970167 -0.852607 -0.57347235

-1.32173606 -0.9393813 -0.46767613 -1.17360841 -0.59243843 -0.81945994

-1.07030086 -0.75048246 -1.27563295 -0.96278885 -0.42760459 -1.04809671

-1.46120317 -1.46428521 -0.94471363 -0.47953744 -0.98878457 -0.26220194

-0.37523066 -0.61756717 -0.80162952 -0.21397883 -0.65632663 -0.98149854

-0.41874218 -0.61484032 -0.09496718 -0.77232454 -0.19296106 -0.64690202

-0.56981287 -0.03604453 -0.44445902 0.31323543]

