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

import numpy as np

import pandas as pd

data = pd.read\_csv(path+'/enjoysport.csv')

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

print("\nInstances are:\n",concepts)

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

print("\nTarget Values are: ",target)

def learn(concepts, target):

specific\_h = concepts[0].copy()

print("\nInitialization of specific\_h and genearal\_h")

print("\nSpecific Boundary: ", specific\_h)

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

print("\nGeneric Boundary: ",general\_h)

for i, h in enumerate(concepts):

print("\nInstance", i+1 , "is ", h)

if target[i] == "yes":

print("Instance is Positive ")

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

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

specific\_h[x] ='?'

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

if target[i] == "no":

print("Instance is Negative ")

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("Specific Bundary after ", i+1, "Instance is ", specific\_h)

print("Generic Boundary after ", i+1, "Instance is ", general\_h)

print("\n")

1. **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.**

#Import libraries and read data using read\_csv() function. Remove the target from the data and store #attributes in the features variable.

import pandas as pd

import math

import numpy as np

data = pd.read\_csv("Dataset/4-dataset.csv")

features = [feat for feat in data]

features.remove("answer")

# Create a class named Node with four members children, value, isLeaf and pred.

class Node:

def \_\_init\_\_(self):

self.children = []

self.value = ""

self.isLeaf = False

self.pred = ""

# Define a function called entropy to find the entropy oof the dataset

def entropy(examples):

pos = 0.0

neg = 0.0

for \_, row in examples.iterrows():

if row["answer"] == "yes":

pos += 1

else:

neg += 1

if pos == 0.0 or neg == 0.0:

return 0.0

else:

p = pos / (pos + neg)

n = neg / (pos + neg)

return -(p \* math.log(p, 2) + n \* math.log(n, 2))

# Define a function named info\_gain to find the gain of the attribute

def info\_gain(examples, attr):

uniq = np.unique(examples[attr])

#print ("\n",uniq)

gain = entropy(examples)

#print ("\n",gain)

for u in uniq:

subdata = examples[examples[attr] == u]

#print ("\n",subdata)

sub\_e = entropy(subdata)

gain -= (float(len(subdata)) / float(len(examples))) \* sub\_e

#print ("\n",gain)

return gain

# Define a function named ID3 to get the decision tree for the given dataset

def ID3(examples, attrs):

root = Node()

max\_gain = 0

max\_feat = ""

for feature in attrs:

#print ("\n",examples)

gain = info\_gain(examples, feature)

if gain > max\_gain:

max\_gain = gain

max\_feat = feature

root.value = max\_feat

#print ("\nMax feature attr",max\_feat)

uniq = np.unique(examples[max\_feat])

#print ("\n",uniq)

for u in uniq:

#print ("\n",u)

subdata = examples[examples[max\_feat] == u]

#print ("\n",subdata)

if entropy(subdata) == 0.0:

newNode = Node()

newNode.isLeaf = True

newNode.value = u

newNode.pred = np.unique(subdata["answer"])

root.children.append(newNode)

else:

dummyNode = Node()

dummyNode.value = u

new\_attrs = attrs.copy()

new\_attrs.remove(max\_feat)

child = ID3(subdata, new\_attrs)

dummyNode.children.append(child)

root.children.append(dummyNode)

return root

# Define a function named printTree to draw the decision tree

def printTree(root: Node, depth=0):

for i in range(depth):

print("\t", end="")

print(root.value, end="")

if root.isLeaf:

print(" -> ", root.pred)

print()

for child in root.children:

printTree(child, depth + 1)

# Define a function named classify to classify the new example

def classify(root: Node, new):

for child in root.children:

if child.value == new[root.value]:

if child.isLeaf:

print ("Predicted Label for new example", new," is:", child.pred)

exit

else:

classify (child.children[0], new)

# Finally, call the ID3, printTree and classify functions

root = ID3(data, features)

print("Decision Tree is:")

printTree(root)

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

new = {"outlook":"sunny", "temperature":"hot", "humidity":"normal", "wind":"strong"}

classify (root, new)

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

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) #maximum of X array longitudinally

y = y/100

#Sigmoid Function

def sigmoid (x):

return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function

def derivatives\_sigmoid(x):

return x \* (1 - x)

#Variable initialization

epoch=5 #Setting training iterations

lr=0.1 #Setting learning rate

inputlayer\_neurons = 2 #number of features in data set

hiddenlayer\_neurons = 3 #number of hidden layers neurons

output\_neurons = 1 #number of neurons at output layer

#weight and bias initialization

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))

#draws a random range of numbers uniformly of dim x\*y

for i in range(epoch):

#Forward Propogation

hinp1=np.dot(X,wh)

hinp=hinp1 + bh

hlayer\_act = sigmoid(hinp)

outinp1=np.dot(hlayer\_act,wout)

outinp= outinp1+bout

output = sigmoid(outinp)

#Backpropagation

EO = y-output

outgrad = derivatives\_sigmoid(output)

d\_output = EO \* outgrad

EH = d\_output.dot(wout.T)

hiddengrad = derivatives\_sigmoid(hlayer\_act)#how much hidden layer wts contributed to error

d\_hiddenlayer = EH \* hiddengrad

wout += hlayer\_act.T.dot(d\_output) \*lr # dotproduct of nextlayererror and currentlayerop

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

print ("-----------Epoch-", i+1, "Starts----------")

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

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

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

print ("-----------Epoch-", i+1, "Ends----------\n")

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

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

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

RESULT:

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.79623473]

[0.78886338]

[0.7952609 ]]

-----------Epoch- 1 Ends----------

-----------Epoch- 2 Starts----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.79781876]

[0.79037225]

[0.79683561]]

-----------Epoch- 2 Ends----------

-----------Epoch- 3 Starts----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.79935983]

[0.79184099]

[0.79836775]]

-----------Epoch- 3 Ends----------

-----------Epoch- 4 Starts----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.80085967]

[0.79327116]

[0.79985899]]

-----------Epoch- 4 Ends----------

-----------Epoch- 5 Starts----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.80231989]

[0.79466427]

[0.80131096]]

-----------Epoch- 5 Ends----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.80231989]

[0.79466427]

[0.80131096]]

1. **Write a program for Implementation of K-Nearest Neighbors (K-NN) in Python**

import numpy as np

import pandas as pd

dataset = pd.read\_csv("D:/GEO/BE COURSES/LAB/DATASET/knn1 (1).csv")

"""

The breast cancer dataset has the following features: Sample code number, Clump Thickness, Uniformity of Cell Size,

Uniformity of Cell Shape, Marginal Adhesion, Single Epithelial Cell Size, Bare Nuclei, Bland Chromatin,

Normal Nucleoli, Mitosis, Class.

"""

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

dataset.shape

#splitting the dataset into the Training set 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 = 0.20, random\_state = 42)

#Feature Scaling

"""

Feature scaling is the process of converting the data into a given range.

In this case, the standard scalar technique is used.

"""

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

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

#Training the K-Nearest Neighbors (K-NN) Classification model on the Training set

"""

Once the dataset is scaled, next, the K-Nearest Neighbors (K-NN) classifier algorithm is used to create a model.

The hyperparameters such as n\_neighbors, metric, and p are set to 5, Minkowski, and 2 respectively.

The remaining hyperparameters are set to default values.

"""

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors = 5, metric = 'minkowski', p = 2)

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

"""

Display the results (confusion matrix and accuracy)

Here evaluation metrics such as confusion matrix and accuracy are used to evaluate the performance of the model built using a decision tree classifier.

"""

from sklearn.metrics import confusion\_matrix, accuracy\_score

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

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

print(cm)

accuracy\_score(y\_test, y\_pred)

1. **Write a program to implement Naïve Bayes algorithm in python and to display the results using confusion matrix and accuracy. Java/Python ML library classes can be used for this problem.**

# -\*- coding: utf-8 -\*-

Implementation of Naive Bayes in Python – Machine Learning

In this tutorial, we will understand the Implementation of Naive Bayes in Python – Machine Learning.

Importing the Necessary libraries

To begin the implementation first we will import the necessary libraries

like NumPy for numerical computation and pandas for reading the dataset.

"""

import numpy as np

import pandas as pd

#Importing the dataset

"""

Next, we import or read the dataset. Click here to download the breast cancer dataset used in this implementation.

After reading the dataset, divide the dataset into concepts and targets. Store the concepts into X and

targets into y.

"""

dataset = pd.read\_csv("D:/GEO/BE COURSES/LAB/DATASET/breastcancer.csv")

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

"""

Splitting the dataset into the Training set and Test set

Once the dataset is read into the memory, next, divide the dataset into two parts, training and

testing using the train\_test\_split function from sklearn.

The test\_size and random\_state attributes are set to 0.25 and 0 respectively.

You can change these attributes as per your requirements.

"""

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)

#Feature Scaling

"""

Feature scaling is the process of converting the data into a min-max range. In this case,

the standard scalar method is used.

"""

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

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

"""

Training the Naive Bayes Classification model on the Training set

Once the dataset is scaled, next, the Naive Bayes classifier algorithm is used to create a model.

The GaussianNB function is imported from sklearn.naive\_bayes library. The hyperparameters such as kernel,

and random\_state to linear, and 0 respectively. The remaining hyperparameters of the support vector machine

algorithm are set to default values.

"""

from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

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

#Naive Bayes classifier model

GaussianNB(priors=None, var\_smoothing=1e-09)

#Display the results (confusion matrix and accuracy)

"""

Here evaluation metrics such as confusion matrix and accuracy are used to evaluate the performance of

the model built using a decision tree classifier.

"""

from sklearn.metrics import confusion\_matrix, accuracy\_score

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

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

print(cm)

accuracy\_score(y\_test, y\_pred)

1. **Write a program to implement Linear Regression (LR) algorithm in python**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('D:/GEO/BE COURSES/LAB/DATASET/Salary\_Data.csv')

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

dataset.head()

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)

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

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

LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=None, normalize=False)

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

pd.DataFrame(data={'Actuals': y\_test, 'Predictions': y\_pred})

#Visualising the Training set results Here scatter plot is used to visualize the results.

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

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

plt.title('Salary vs Experience (Training set)')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.show()

1. **Implementation Of Linear And Polynomial Regression In Python**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('D:/GEO/BE COURSES/LAB/DATASET/Position\_Salaries.csv')

X = dataset.iloc[:, 1:-1].values

y = dataset.iloc[:, -1].values

"""

Training the Linear Regression model on the Whole dataset

A Linear regression algorithm is used to create a model.

A LinearRegression function is imported from sklearn.linear\_model library.

"""

from sklearn.linear\_model import LinearRegression

lin\_reg = LinearRegression()

lin\_reg.fit(X, y)

#Linear Regression classifier model

LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=None, normalize=False)

"""

Training the Polynomial Regression model on the Whole dataset

A polynomial regression algorithm is used to create a model.

"""

from sklearn.preprocessing import PolynomialFeatures

poly\_reg = PolynomialFeatures(degree = 4)

X\_poly = poly\_reg.fit\_transform(X)

lin\_reg\_2 = LinearRegression()

lin\_reg\_2.fit(X\_poly, y)

#Polynomial Regression classifier model

LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=None, normalize=False)

"""

Visualising the Linear Regression results

Here scatter plot is used to visualize the results. The title of the plot is set to Truth or Bluff

(Linear Regression), xlabel is set to Position Level , and ylabel is set to Salary.

"""

plt.scatter(X, y, color = 'red')

plt.plot(X, lin\_reg.predict(X), color = 'blue')

plt.title('Truth or Bluff (Linear Regression)')

plt.xlabel('Position Level')

plt.ylabel('Salary')

plt.show()

#Visualising the Polynomial Regression results

"""

The title of the plot is set to Truth or Bluff (Polynomial Regression), xlabel is set to Position level,

and ylabel is set to Salary.

"""

plt.scatter(X, y, color = 'red')

plt.plot(X, lin\_reg\_2.predict(poly\_reg.fit\_transform(X)), color = 'blue')

plt.title('Truth or Bluff (Polynomial Regression)')

plt.xlabel('Position level')

plt.ylabel('Salary')

plt.show()

1. **Write a program to implement Logistic Regression (LR) algorithm in python**

Implementation of Logistic Regression (LR) in Python – Machine Learning

In this tutorial, we will understand the Implementation of Logistic Regression (LR) in Python – Machine Learning.

Importing the libraries

"""

import numpy as np

import pandas as pd

#"Importing the dataset

"""

After importing the necessary libraries, next, we import or read the dataset.

Click here to download the breast cancer dataset used in this implementation.

The breast cancer dataset has the following features:

Sample code number, Clump Thickness, Uniformity of Cell Size, Uniformity of Cell Shape, Marginal Adhesion,

Single Epithelial Cell Size, Bare Nuclei, Bland Chromatin, Normal Nucleoli, Mitosis, Class.

"""

# divide the dataset into concepts and targets. Store the concepts into X and targets into y.

dataset = pd.read\_csv("D:/GEO/BE COURSES/LAB/DATASET/breastcancer.csv")

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

#Splitting the dataset into the Training set and Test

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.30, random\_state = 2)

#Feature Scaling

"""

Feature scaling is the process of converting the data into a given range. In this case, the standard scalar technique is used.

from sklearn.preprocessing import StandardScaler

"""

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

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

"""

Training the Logistic Regression (LR) Classification model on the Training set

Once the dataset is scaled, next, the Logistic Regression (LR) classifier algorithm is used to create a model.

The hyperparameters such as random\_state to 0 respectively.

The remaining hyperparameters Logistic Regression (LR) are set to default values.

"""

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state = 0)

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

#Logistic Regression (LR) classifier model

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,

intercept\_scaling=1, l1\_ratio=None, max\_iter=100,

multi\_class='warn', n\_jobs=None, penalty='l2',

random\_state=0, solver='warn', tol=0.0001, verbose=0,

warm\_start=False)

#Display the results (confusion matrix and accuracy)

"""

Here evaluation metrics such as confusion matrix and accuracy are used to evaluate the performance of the model

built using a decision tree classifier.

"""

from sklearn.metrics import confusion\_matrix, accuracy\_score

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

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

print(cm)

accuracy\_score(y\_test, y\_pred)

1. **There are three credit scores that banks and credit card companies use to label their customers: Good, Standard, Poor A person with a good credit score will get loans from any bank and financial institution. Write a program for the task of Credit Score Classification, we need a labelled dataset with credit scores.**

**Credit Score Classification**

There are three credit scores that banks and credit card companies use to label their customers:

Good

Standard

Poor

A person with a good credit score will get loans from any bank and financial institution. For the task of Credit Score Classification, we need a labelled dataset with credit scores.

import pandas as pd

import numpy as np

import plotly.express as px

import plotly.graph\_objects as go

import plotly.io as pio

pio.templates.default = "plotly\_white"

import plotly.io as io

io.renderers.default='browser'

data = pd.read\_csv('D:/GEO/BE COURSES/LAB/DATASET/Credit-Score-Data/Credit Score Data/train.csv')

print(data.head())

print(data.info())

#the dataset has any null values or not:

print(data.isnull().sum())

#The dataset doesn’t have any null values. As this dataset is labelled, let’s have a look at the Credit\_Score column values:

data["Credit\_Score"].value\_counts()

data.shape

#Data Exploration

"""

The dataset has many features that can train a Machine Learning model for credit score classification.

Let’s explore all the features one by one.

I will start by exploring the occupation feature to know if the occupation of the person affects credit scores:

"""

fig = px.box(data,

x="Occupation",

color="Credit\_Score",

title="Credit Scores Based on Occupation",

color\_discrete\_map={'Poor':'red',

'Standard':'yellow',

'Good':'green'})

fig.show()

"""

There’s not much difference in the credit scores of all occupations mentioned in the data. Now let’s explore

whether the Annual Income of the person impacts your credit scores or not:

"""

fig = px.box(data,

x="Credit\_Score",

y="Annual\_Income",

color="Credit\_Score",

title="Credit Scores Based on Annual Income",

color\_discrete\_map={'Poor':'red',

'Standard':'yellow',

'Good':'green'})

fig.update\_traces(quartilemethod="exclusive")

fig.show()

"""

let’s explore whether the monthly in-hand salary impacts credit scores or not:

"""

fig = px.box(data,

x="Credit\_Score",

y="Monthly\_Inhand\_Salary",

color="Credit\_Score",

title="Credit Scores Based on Monthly Inhand Salary",

color\_discrete\_map={'Poor':'red',

'Standard':'yellow',

'Good':'green'})

fig.update\_traces(quartilemethod="exclusive")

fig.show()

fig = px.box(data,

x="Credit\_Score",

y="Num\_Bank\_Accounts",

color="Credit\_Score",

title="Credit Scores Based on Number of Bank Accounts",

color\_discrete\_map={'Poor':'red',

'Standard':'yellow',

'Good':'green'})

fig.update\_traces(quartilemethod="exclusive")

fig.show()

# impact on credit scores based on the number of credit cards you have:

fig = px.box(data,

x="Credit\_Score",

y="Num\_Credit\_Card",

color="Credit\_Score",

title="Credit Scores Based on Number of Credit cards",

color\_discrete\_map={'Poor':'red',

'Standard':'yellow',

'Good':'green'})

fig.update\_traces(quartilemethod="exclusive")

fig.show()

fig = px.box(data,

x="Credit\_Score",

y="Interest\_Rate",

color="Credit\_Score",

title="Credit Scores Based on the Average Interest rates",

color\_discrete\_map={'Poor':'red',

'Standard':'yellow',

'Good':'green'})

fig.update\_traces(quartilemethod="exclusive")

fig.show()

data["Credit\_Mix"] = data["Credit\_Mix"].map({"Standard": 1,

"Good": 2,

"Bad": 0})

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

x = np.array(data[["Annual\_Income", "Monthly\_Inhand\_Salary",

"Num\_Bank\_Accounts", "Num\_Credit\_Card",

"Interest\_Rate", "Num\_of\_Loan",

"Delay\_from\_due\_date", "Num\_of\_Delayed\_Payment",

"Credit\_Mix", "Outstanding\_Debt",

"Credit\_History\_Age", "Monthly\_Balance"]])

y = np.array(data[["Credit\_Score"]])

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y,

test\_size=0.33,

random\_state=42)

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()

model.fit(xtrain, ytrain)

print("Credit Score Prediction : ")

a = float(input("Annual Income: "))

b = float(input("Monthly Inhand Salary: "))

c = float(input("Number of Bank Accounts: "))

d = float(input("Number of Credit cards: "))

e = float(input("Interest rate: "))

f = float(input("Number of Loans: "))

g = float(input("Average number of days delayed by the person: "))

h = float(input("Number of delayed payments: "))

i = input("Credit Mix (Bad: 0, Standard: 1, Good: 3) : ")

j = float(input("Outstanding Debt: "))

k = float(input("Credit History Age: "))

l = float(input("Monthly Balance: "))

features = np.array([[a, b, c, d, e, f, g, h, i, j, k, l]])

print("Predicted Credit Score = ", model.predict(features))

1. **Iris Flower Classification using KNN**

Iris Flower Classification using Python

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

iris = pd.read\_csv("D:/GEO/BE COURSES/LAB/DATASET/IRIS.csv")

#first five rows of this dataset:

print(iris.head())

print(iris.describe())

#The target labels of this dataset are present in the species column, let’s have a quick look at the target labels:

print("Target Labels", iris["species"].unique())

#plot the data using a scatter plot which will plot the iris species according to the sepal length and sepal width:

import plotly.io as io

io.renderers.default='browser'

import plotly.express as px

fig = px.scatter(iris, x="sepal\_width", y="sepal\_length", color="species")

fig.show()

#Iris Classification Model

x = iris.drop("species", axis=1)

y = iris["species"]

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

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

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=1)

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

x\_new = np.array([[6, 2.9, 1, 0.2]])

prediction = knn.predict(x\_new)

print("Prediction: {}".format(prediction))

1. **Car Price Prediction Model using Python**

car price prediction

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.tree import DecisionTreeRegressor

data = pd.read\_csv("D:/GEO/BE COURSES/LAB/DATASET/CarPrice.csv")

data.head()

data.shape

data.isnull().sum()

#So this dataset doesn’t have any null values, now let’s look at some of the other important insights to get

#an idea of what kind of data we’re dealing with:

data.info()

data.describe()

data.CarName.unique()

sns.set\_style("whitegrid")

plt.figure(figsize=(15, 10))

sns.distplot(data.price)

plt.show()

#Now let’s have a look at the correlation among all the features of this dataset:

print(data.corr())

plt.figure(figsize=(20, 15))

correlations = data.corr()

sns.heatmap(correlations, cmap="coolwarm", annot=True)

plt.show()

#Training a Car Price Prediction Model

predict = "price"

data = data[["symboling", "wheelbase", "carlength",

"carwidth", "carheight", "curbweight",

"enginesize", "boreratio", "stroke",

"compressionratio", "horsepower", "peakrpm",

"citympg", "highwaympg", "price"]]

x = np.array(data.drop([predict], 1))

y = np.array(data[predict])

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

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.2)

from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor()

model.fit(xtrain, ytrain)

predictions = model.predict(xtest)

from sklearn.metrics import mean\_absolute\_error

model.score(xtest, predictions)

1. **House Price Prediction using Python**

# -\*- coding: utf-8 -\*-

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

dataset = pd.read\_excel("HousePricePrediction.xlsx")

# Printing first 5 records of the dataset

print(dataset.head(5))

dataset.shape

obj = (dataset.dtypes == 'object')

object\_cols = list(obj[obj].index)

print("Categorical variables:",len(object\_cols))

int\_ = (dataset.dtypes == 'int')

num\_cols = list(int\_[int\_].index)

print("Integer variables:",len(num\_cols))

fl = (dataset.dtypes == 'float')

fl\_cols = list(fl[fl].index)

print("Float variables:",len(fl\_cols))

plt.figure(figsize=(12, 6))

sns.heatmap(dataset.corr(),

cmap = 'BrBG',

fmt = '.2f',

linewidths = 2,

annot = True)

unique\_values = []

for col in object\_cols:

unique\_values.append(dataset[col].unique().size)

plt.figure(figsize=(10,6))

plt.title('No. Unique values of Categorical Features')

plt.xticks(rotation=90)

sns.barplot(x=object\_cols,y=unique\_values)

plt.figure(figsize=(18, 36))

plt.title('Categorical Features: Distribution')

plt.xticks(rotation=90)

index = 1

for col in object\_cols:

y = dataset[col].value\_counts()

plt.subplot(11, 4, index)

plt.xticks(rotation=90)

sns.barplot(x=list(y.index), y=y)

index += 1

dataset.drop(['Id'],

axis=1,

inplace=True)

dataset['SalePrice'] = dataset['SalePrice'].fillna(dataset['SalePrice'].mean())

new\_dataset = dataset.dropna()

new\_dataset.isnull().sum()

from sklearn.preprocessing import OneHotEncoder

s = (new\_dataset.dtypes == 'object')

object\_cols = list(s[s].index)

print("Categorical variables:")

print(object\_cols)

print('No. of. categorical features: ',len(object\_cols))

OH\_encoder = OneHotEncoder(sparse=False)

OH\_cols = pd.DataFrame(OH\_encoder.fit\_transform(new\_dataset[object\_cols]))

OH\_cols.index = new\_dataset.index

OH\_cols.columns = OH\_encoder.get\_feature\_names()

df\_final = new\_dataset.drop(object\_cols, axis=1)

df\_final = pd.concat([df\_final, OH\_cols], axis=1)

from sklearn.metrics import mean\_absolute\_error

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

X = df\_final.drop(['SalePrice'], axis=1)

Y = df\_final['SalePrice']

# Split the training set into

# training and validation set

X\_train, X\_valid, Y\_train, Y\_valid = train\_test\_split(X, Y, train\_size=0.8, test\_size=0.2, random\_state=0)

#Model and Accuracy

#svm

from sklearn import svm

from sklearn.svm import SVC

from sklearn.metrics import mean\_absolute\_percentage\_error

model\_SVR = svm.SVR()

model\_SVR.fit(X\_train,Y\_train)

Y\_pred = model\_SVR.predict(X\_valid)

print(mean\_absolute\_percentage\_error(Y\_valid, Y\_pred))

#Random forest

from sklearn.ensemble import RandomForestRegressor

model\_RFR = RandomForestRegressor(n\_estimators=10)

model\_RFR.fit(X\_train, Y\_train)

Y\_pred = model\_RFR.predict(X\_valid)

mean\_absolute\_percentage\_error(Y\_valid, Y\_pred)

#LinearRegression

from sklearn.linear\_model import LinearRegression

model\_LR = LinearRegression()

model\_LR.fit(X\_train, Y\_train)

Y\_pred = model\_LR.predict(X\_valid)

print(mean\_absolute\_percentage\_error(Y\_valid, Y\_pred))

1. **Mobile Price Classification using Python**

Mobile Price Classification using Python

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

sns.set()

import plotly.io as io

io.renderers.default='browser'

data = pd.read\_csv("D:/GEO/BE COURSES/LAB/DATASET/mobile\_prices.csv")

print(data.head())

plt.figure(figsize=(12, 10))

sns.heatmap(data.corr(), annot=True, cmap="coolwarm", linecolor='white', linewidths=1)

#data preparation

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

y = data.iloc[:, -1].values

x = StandardScaler().fit\_transform(x)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.20, random\_state=0)

# Logistic Regression algorithm provided by Scikit-learn:

from sklearn.linear\_model import LogisticRegression

lreg = LogisticRegression()

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

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

#accuracy of the model:

accuracy = accuracy\_score(y\_test, y\_pred) \* 100

print("Accuracy of the Logistic Regression Model: ",accuracy)

#predictions made by the model:

print(y\_pred)

#Let’s have a look at the number of mobile phones classified for each price range:

(unique, counts) = np.unique(y\_pred, return\_counts=True)

price\_range = np.asarray((unique, counts)).T

print(price\_range)

1. **Future Sales Prediction using Python**

Future Sales Prediction using Python

import pandas as pd

import numpy as np

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

from sklearn.linear\_model import LinearRegression

import plotly.io as io

io.renderers.default='browser'

data = pd.read\_csv("D:/GEO/BE COURSES/LAB/DATASET/futuresale prediction.csv")

print(data.head())

print(data.sample(5))

print(data.isnull().sum())

import plotly.express as px

import plotly.graph\_objects as go

figure = px.scatter(data\_frame = data, x="Sales",

y="TV", size="TV", trendline="ols")

figure.show()

figure = px.scatter(data\_frame = data, x="Sales",

y="Newspaper", size="Newspaper", trendline="ols")

figure.show()

figure = px.scatter(data\_frame = data, x="Sales",

y="Radio", size="Radio", trendline="ols")

figure.show()

correlation = data.corr()

print(correlation["Sales"].sort\_values(ascending=False))

x = np.array(data.drop(["Sales"], 1))

y = np.array(data["Sales"])

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y,

test\_size=0.2,

random\_state=42)

model = LinearRegression()

model.fit(xtrain, ytrain)

print(model.score(xtest, ytest))

features = [[TV, Radio, Newspaper]]

features = np.array([[230.1, 37.8, 69.2]])

print(model.predict(features))