**NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY**



**Machine Learning**

**COCSC17**

**Submitted By:**

**Laksh Sachdeva**

**2022UCS1572**

**CSE-1**

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| **S.No.** | **EXPERIMENT** |
| 1 | **Linear Regression** |
| 2 | **Logistic Regression** |
| 3 | **SVM** |
| 4 | **Random forest** |
| 5 | **Decision tree** |
| 6 | **Adaboost** |
| 7 | **ANN** |
| 8 | **CNN** |
| 9 | **KNN** |
| 10 | **RNN** |
| 11 | **K-Means clustering** |

1. **Implementation of Linear Regression**

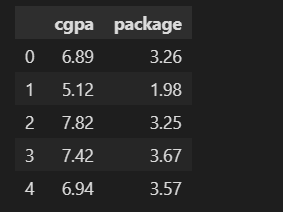
import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

df = pd.read\_csv("placement (1).csv")

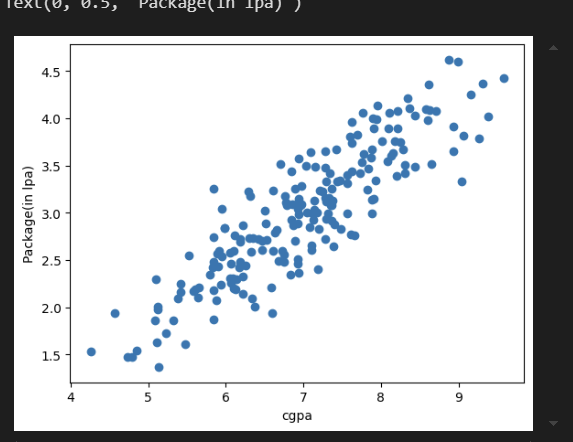
df.head()



plt.scatter(df['cgpa'] , df['package'])

plt.xlabel("cgpa")

plt.ylabel("Package(in lpa)")



X = df.iloc[: , 0:1]

y = df.iloc[: , 1]

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=42)

lr = LinearRegression()

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

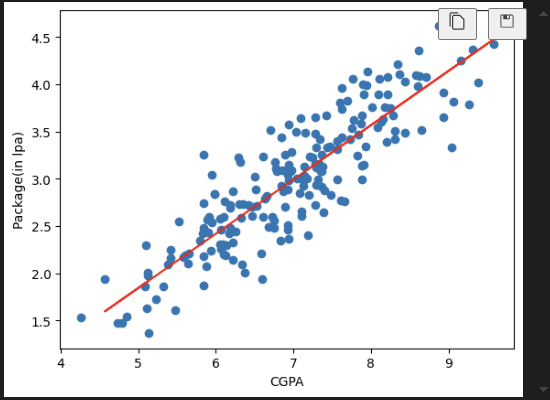
lr.predict(X\_train)

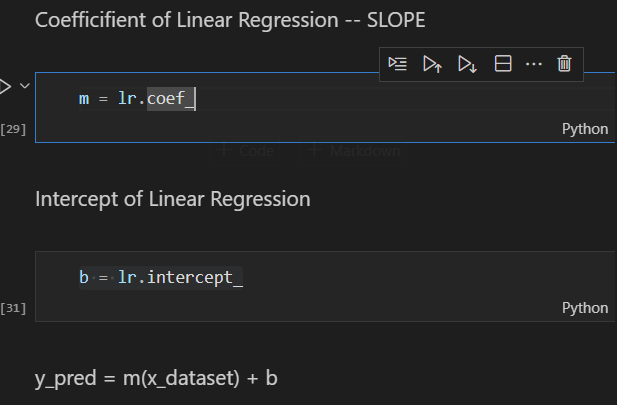
plt.scatter(df['cgpa'] , df['package'])

plt.plot(X\_train , lr.predict(X\_train) , color = "red")

plt.xlabel("CGPA")

plt.ylabel("Package(in lpa)")



lr.predict([[8.58]])



1. **Implementation of Logistic Regression**

**The dataset has two columns - age (age of the person/customer) and bought\_insurance (whether the customer bought insurance or not). If bought\_insurance = 1, the customer bought insurance and if bought\_insurance = 0, the customer did not buy the insurance.**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

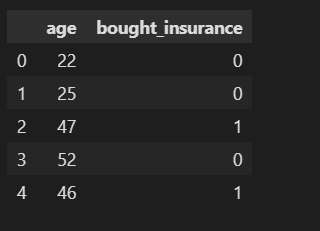
import seaborn as sns

df = pd.read\_csv("insurance\_data.csv")

df.head()

df = pd.read\_csv("insurance\_data.csv")

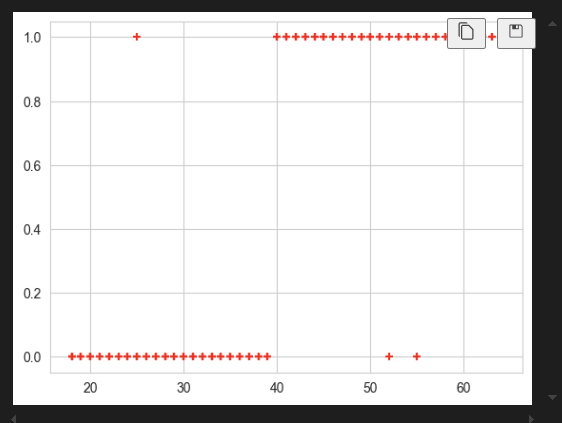
df.head()



print("Classification classes in the dataset : " , np.unique(df['bought\_insurance']))

Classification classes in the dataset : [0 1]

plt.scatter(df['age'],df['bought\_insurance'],marker='+',color='red')



X = df[['age']]     # input variable

y = df['bought\_insurance']    # output variable

Shape: (196, 1) Dimension: 2

Shape: (196,) Dimension: 1

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 = 42)

from sklearn.linear\_model import LogisticRegression

log\_r = LogisticRegression()

log\_r.fit(X\_train , y\_train)

y\_pred = log\_r.predict(X\_test)

coeff = log\_r.coef\_

b = log\_r.intercept\_

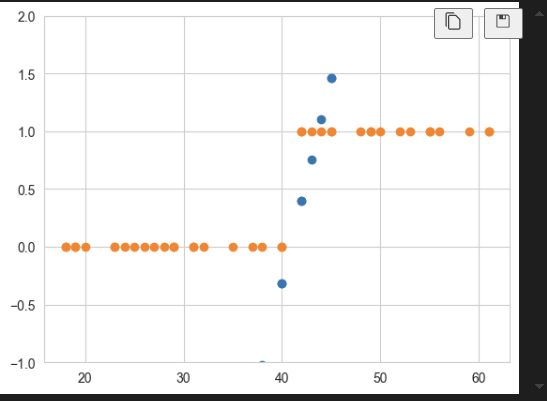
x\_in = X\_test

y\_val = coeff\*x\_in + b

plt.scatter(x\_in , y\_val)

plt.scatter(X\_test , y\_pred)

plt.ylim(-1,2)



print("Predict when age is 63" , log\_r.predict([[63]]))

print("Predict when age is 40" , log\_r.predict([[40]]))



from sklearn.metrics import confusion\_matrix

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

tn, fp, fn, tp = confusion\_matrix(y\_test, y\_pred).ravel()

total\_samples = len(y\_test)

tn\_percent = (tn / total\_samples) \* 100

fp\_percent = (fp / total\_samples) \* 100

fn\_percent = (fn / total\_samples) \* 100

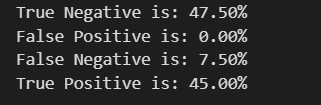
tp\_percent = (tp / total\_samples) \* 100

print(f"True Negative is: {tn\_percent:.2f}%")

print(f"False Positive is: {fp\_percent:.2f}%")

print(f"False Negative is: {fn\_percent:.2f}%")

print(f"True Positive is: {tp\_percent:.2f}%")



from sklearn.metrics import accuracy\_score

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

print("Accuracy of model is: " , acc ,"%")



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

print("Precision " , precision\_score(y\_test,y\_pred,average=None))



1. **Implementation of SVM**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

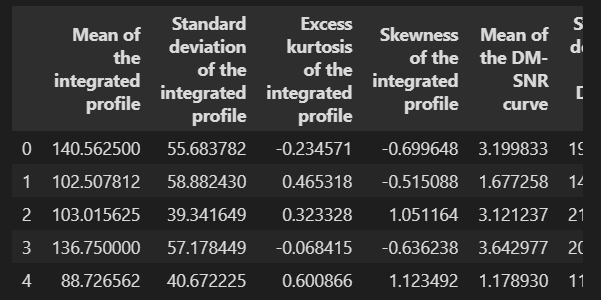
import seaborn as sns

%matplotlib inline

df = pd.read\_csv('dataset.csv')

df.shape



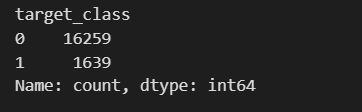
df.head()

We can see that there are 9 variables in the dataset. 8 are continuous variables and 1 is discrete variable. The discrete variable is target\_class variable. It is also the target variable.

df.columns = ['IP Mean', 'IP Sd', 'IP Kurtosis', 'IP Skewness',

              'DM-SNR Mean', 'DM-SNR Sd', 'DM-SNR Kurtosis', 'DM-SNR Skewness', 'target\_class']

df['target\_class'].value\_counts()



**There are 9 numerical variables in the dataset.**

**8 are continuous variables and 1 is discrete variable.**

**The discrete variable is target\_class variable. It is also the target variable.**

**There are no missing values in the dataset.**

# draw boxplots to visualize outliers

plt.figure(figsize=(24,20))

plt.subplot(4, 2, 1)

fig = df.boxplot(column='IP Mean')

fig.set\_title('')

fig.set\_ylabel('IP Mean')

plt.subplot(4, 2, 2)

fig = df.boxplot(column='IP Sd')

fig.set\_title('')

fig.set\_ylabel('IP Sd')

plt.subplot(4, 2, 3)

fig = df.boxplot(column='IP Kurtosis')

fig.set\_title('')

fig.set\_ylabel('IP Kurtosis')

plt.subplot(4, 2, 4)

fig = df.boxplot(column='IP Skewness')

fig.set\_title('')

fig.set\_ylabel('IP Skewness')

plt.subplot(4, 2, 5)

fig = df.boxplot(column='DM-SNR Mean')

fig.set\_title('')

fig.set\_ylabel('DM-SNR Mean')

plt.subplot(4, 2, 6)

fig = df.boxplot(column='DM-SNR Sd')

fig.set\_title('')

fig.set\_ylabel('DM-SNR Sd')

plt.subplot(4, 2, 7)

fig = df.boxplot(column='DM-SNR Kurtosis')

fig.set\_title('')

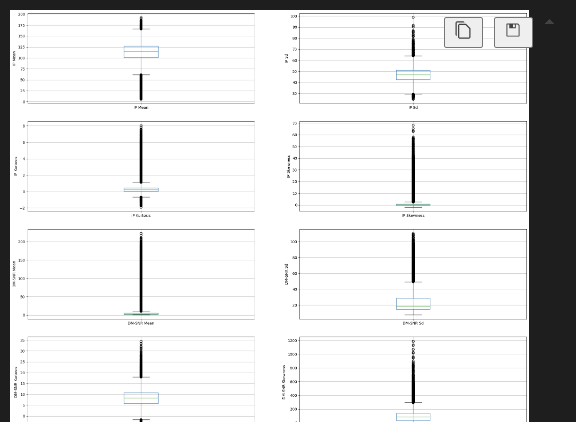
fig.set\_ylabel('DM-SNR Kurtosis')

plt.subplot(4, 2, 8)

fig = df.boxplot(column='DM-SNR Skewness')

fig.set\_title('')

fig.set\_ylabel('DM-SNR Skewness')



**here are 2 variants of SVMs. They are hard-margin variant of SVM and soft-margin variant of SVM.**

**The hard-margin variant of SVM does not deal with outliers. In this case, we want to find the hyperplane with maximum margin such that every training point is correctly classified with margin at least 1. This technique does not handle outliers well.**

**Another version of SVM is called soft-margin variant of SVM. In this case, we can have a few points incorrectly classified or classified with a margin less than 1. But for every such point, we have to pay a penalty in the form of C parameter, which controls the outliers. Low C implies we are allowing more outliers and high C implies less outliers.**

**The message is that since the dataset contains outliers, so the value of C should be high while training the model.**

X = df.drop(['target\_class'], axis=1)

y = df['target\_class']

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 = 42)

# check the shape of X\_train and X\_test

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

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

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

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

svc = SVC()

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

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

print('Model accuracy score with default hyperparameters: {0:0.4f}'. format(accuracy\_score(y\_test, y\_pred)))



from sklearn.metrics import confusion\_matrix

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

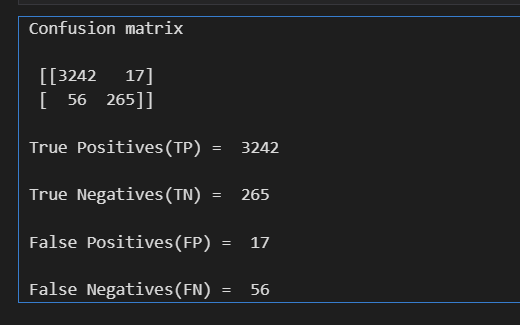
print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

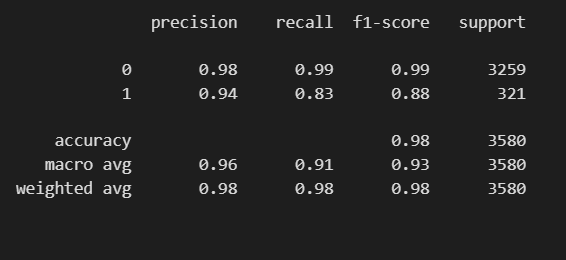
print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])



from sklearn.metrics import classification\_report

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



1. **Implementation of Random Forest**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

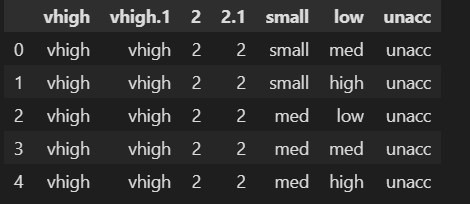
%matplotlib inline

df = pd.read\_csv('car\_evaluation (1).csv')

df.shape



df.head()



col\_names = ['buying', 'maint', 'doors', 'persons', 'lug\_boot', 'safety', 'class']

df.columns = col\_names

df.head()

 col\_names = ['buying', 'maint', 'doors', 'persons', 'lug\_boot', 'safety', 'class']

for col in col\_names:

    print(df[col].value\_counts())

Summary of Model

There are 7 variables in the dataset. All the variables are of categorical data type.

These are given by buying, maint, doors, persons, lug\_boot, safety and class.

class is the target variable.

X = df.drop(['class'], axis=1)

y = df['class']

# split data into training and testing sets

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

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

X\_train.shape, X\_test.shape



import category\_encoders as ce

encoder = ce.OrdinalEncoder(cols=['buying', 'maint', 'doors', 'persons', 'lug\_boot', 'safety'])

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

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

X\_train.head()



# import Random Forest classifier

from sklearn.ensemble import RandomForestClassifier

# instantiate the classifier

rfc = RandomForestClassifier(random\_state=0)

# fit the model

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

# Predict the Test set results

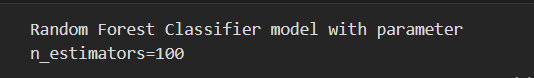
y\_pred = rfc.predict(X\_test)

# Check accuracy score

from sklearn.metrics import accuracy\_score

print('Model accuracy score with 10 decision-trees : {0:0.4f}'.





rfc\_100 = RandomForestClassifier(n\_estimators=100, random\_state=0)

# fit the model to the training set

rfc\_100.fit(X\_train, y\_train)

# Predict on the test set results

y\_pred\_100 = rfc\_100.predict(X\_test)

# Check accuracy score

print('Model accuracy score with 100 decision-trees : {0:0.4f}'. format(accuracy\_score(y\_test, y\_pred\_100)))



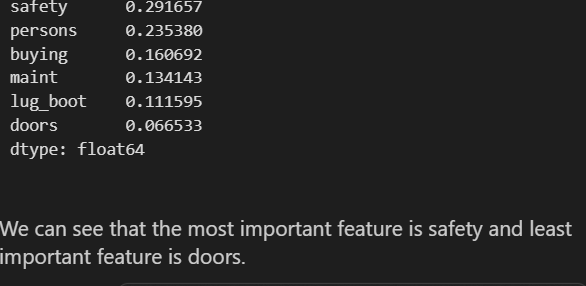
clf = RandomForestClassifier(n\_estimators=100, random\_state=0)

# fit the model to the training set

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

feature\_scores = pd.Series(clf.feature\_importances\_, index=X\_train.columns).sort\_values(ascending=False)

feature\_scores

 sns.barplot(x=feature\_scores, y=feature\_scores.index)

# Add labels to the graph

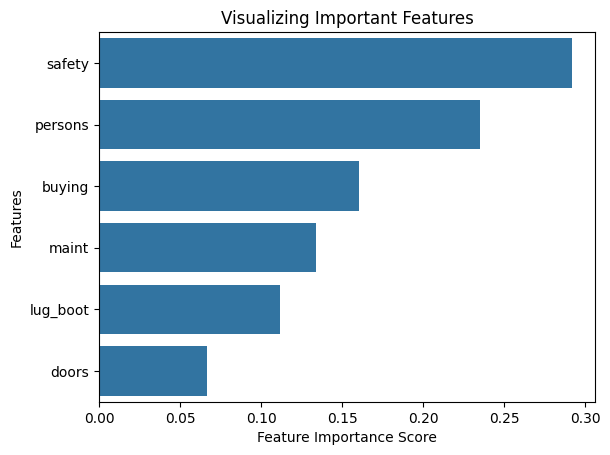
plt.xlabel('Feature Importance Score')

plt.ylabel('Features')

# Add title to the graph

plt.title("Visualizing Important Features")

plt.show()



# declare feature vector and target variable

X = df.drop(['class', 'doors'], axis=1)

y = df['class']

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

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

encoder = ce.OrdinalEncoder(cols=['buying', 'maint', 'persons', 'lug\_boot', 'safety'])

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

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

clf = RandomForestClassifier(random\_state=0)

# fit the model to the training set

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

# Predict on the test set results

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

# Check accuracy score

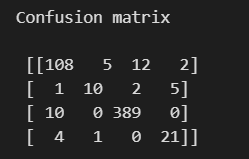
print('Model accuracy score with doors variable removed : {0:0.4f}'. format(accuracy\_score(y\_test, y\_pred)))



from sklearn.metrics import confusion\_matrix

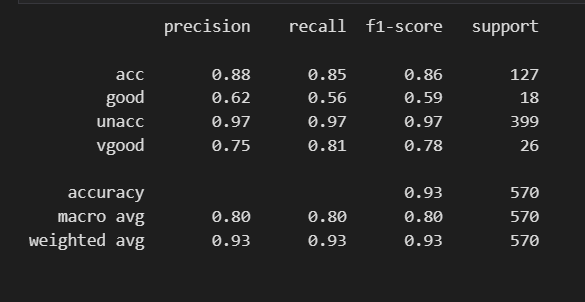
cm = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix\n\n', cm)



from sklearn.metrics import classification\_report

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



1. **Implementation of Decision Trees**

#Loading Libraries

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

import sklearn.datasets as datasets

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, roc\_auc\_score, roc\_curve

from sklearn.tree import plot\_tree

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import GridSearchCV, RandomizedSearchCV

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

df = pd.read\_csv('iris.csv')

df.head()

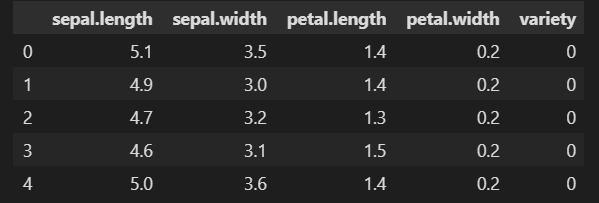


from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

df['variety'] = encoder.fit\_transform(df['variety'])

df.head()



# Splitting the data into train and test sets

X = df.drop("variety",axis=1)

y = df["variety"]

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.3, random\_state= 1)

# Defining an object for DTC and fitting for whole dataset

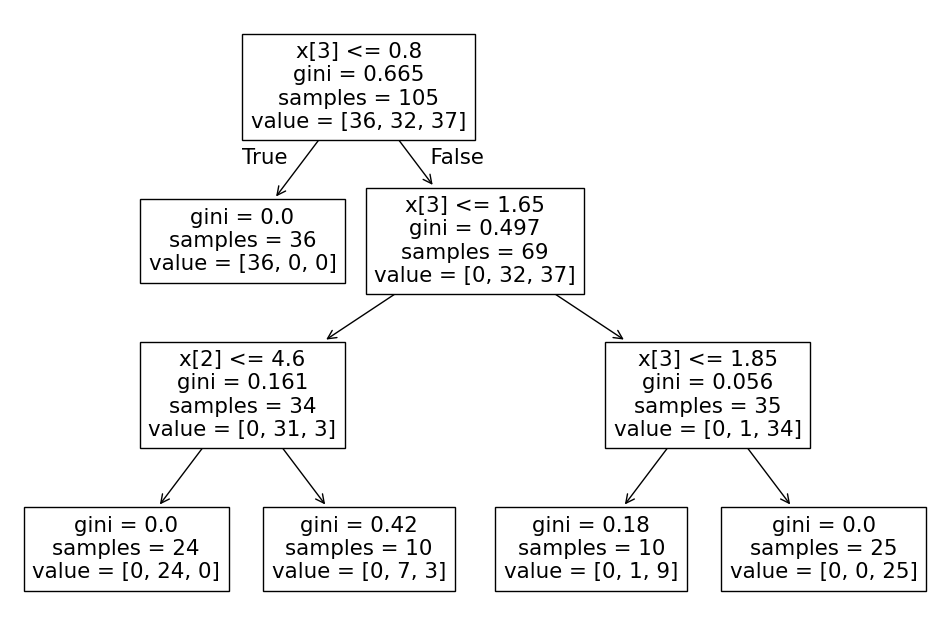
dt = DecisionTreeClassifier(criterion="gini" , max\_depth=3, min\_samples\_leaf=10, random\_state=1 )

dt.fit(X, y)

plt.figure(figsize=(12,8))

from sklearn import tree

tree.plot\_tree(dt.fit(X\_train, y\_train))



from sklearn.metrics import accuracy\_score ,

classification\_report , confusion\_matrix , recall\_score,precision\_score,roc\_curve , roc\_auc\_score

y\_pred\_train = dt.predict(X\_train)

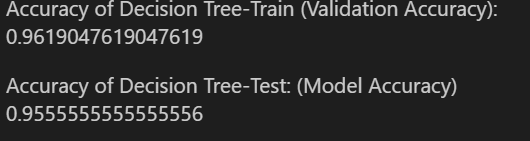
y\_pred = dt.predict(X\_test)

print('Accuracy of Decision Tree-Train  (Validation Accuracy): ', accuracy\_score(y\_pred\_train, y\_train))

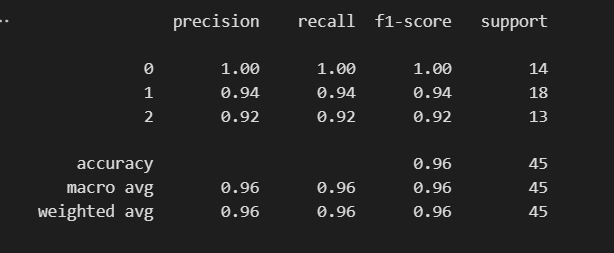
print('Accuracy of Decision Tree-Test: (Model Accuracy) ', accuracy\_score(y\_pred, y\_test))

Accuracy of Decision Tree-Train (Validation Accuracy): 0.9619047619047619

Accuracy of Decision Tree-Test: (Model Accuracy) 0.9555555555555556

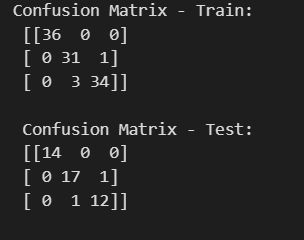


print(classification\_report(y\_test,y\_pred))



print('Confusion Matrix - Train:','\n',confusion\_matrix(y\_train,y\_pred\_train))

print('\n','Confusion Matrix - Test:','\n',confusion\_matrix(y\_test,y\_pred))



1. **Implementation of Adaboost**

import pandas as pd

import numpy as np

from mlxtend.plotting import plot\_decision\_regions

df = pd.DataFrame()

df['X1'] = [1,2,3,4,5,6,6,7,9,9]

df['X2'] = [5,3,6,8,1,9,5,8,9,2]

df['label'] = [1,1,0,1,0,1,0,1,0,0]

import seaborn as sns

sns.scatterplot(x=df['X1'],y=df['X2'],hue=df['label'])

df['weights'] = 1/df.shape[0]

df

from sklearn.tree import DecisionTreeClassifier

dt1 = DecisionTreeClassifier(max\_depth=1)

X = df.iloc[:,0:2].values

y = df.iloc[:,2].values

# Step 2 - Train 1st model

dt1.fit(X,y)

DecisionTreeClassifier(ccp\_alpha=0.0, class\_weight=None, criterion='gini',

                       max\_depth=1, max\_features=None, max\_leaf\_nodes=None,

                       min\_impurity\_decrease=0.0, min\_impurity\_split=None,

                       min\_samples\_leaf=1, min\_samples\_split=2,

                       min\_weight\_fraction\_leaf=0.0, presort='deprecated',

                       random\_state=None, splitter='best')

from sklearn.tree import plot\_tree

plot\_tree(dt1)

plot\_decision\_regions(X, y, clf=dt1, legend=2)

df['y\_pred'] = dt1.predict(X)

def calculate\_model\_weight(error):

  return 0.5\*np.log((1-error)/(error))

# Step 3 - calculate model weight

alpha1 = calculate\_model\_weight(0.3)

alpha1

# Step 4 - Update weights

def update\_row\_weights(row,alpha=0.423):

  if row['label'] == row['y\_pred']:

    return row['weights'] \* np.exp(-alpha)

  else:

    return row['weights'] \* np.exp(alpha)

df['updated\_weights'] = df.apply(update\_row\_weights,axis=1)

df['updated\_weights'].sum()

df['nomalized\_weights'] = df['updated\_weights']/df['updated\_weights'].sum()

df

df['nomalized\_weights'].sum()

df['cumsum\_upper'] = np.cumsum(df['nomalized\_weights'])

df['cumsum\_lower'] = df['cumsum\_upper'] - df['nomalized\_weights']

df[['X1','X2','label','weights','y\_pred','updated\_weights','cumsum\_lower','cumsum\_upper']]

def create\_new\_dataset(df):

  indices = []

  for i in range(df.shape[0]):

    a = np.random.random()

    for index,row in df.iterrows():

      if row['cumsum\_upper'] > a and a > row['cumsum\_lower']:

        indices.append(index)

  return indices

index\_values = create\_new\_dataset(df)

second\_df = df.iloc[index\_values,[0,1,2,3]]

dt2 = DecisionTreeClassifier(max\_depth=1)

X = second\_df.iloc[:,0:2].values

y = second\_df.iloc[:,2].values

dt2.fit(X,y)

DecisionTreeClassifier(ccp\_alpha=0.0, class\_weight=None, criterion='gini',

                       max\_depth=1, max\_features=None, max\_leaf\_nodes=None,

                       min\_impurity\_decrease=0.0, min\_impurity\_split=None,

                       min\_samples\_leaf=1, min\_samples\_split=2,

                       min\_weight\_fraction\_leaf=0.0, presort='deprecated',

                       random\_state=None, splitter='best')

plot\_tree(dt2)

plot\_decision\_regions(X, y, clf=dt2, legend=2)

second\_df['y\_pred'] = dt2.predict(X)

alpha2 = calculate\_model\_weight(0.1)

# Step 4 - Update weights

def update\_row\_weights(row,alpha=1.09):

  if row['label'] == row['y\_pred']:

    return row['weights'] \* np.exp(-alpha)

  else:

    return row['weights'] \* np.exp(alpha)

second\_df['updated\_weights'] = second\_df.apply(update\_row\_weights,axis=1)

second\_df['nomalized\_weights'] = second\_df['updated\_weights']/second\_df['updated\_weights'].sum()

second\_df['nomalized\_weights'].sum()

second\_df['cumsum\_upper'] = np.cumsum(second\_df['nomalized\_weights'])

second\_df['cumsum\_lower'] = second\_df['cumsum\_upper'] - second\_df['nomalized\_weights']

second\_df[['X1','X2','label','weights','y\_pred','nomalized\_weights','cumsum\_lower','cumsum\_upper']]

index\_values = create\_new\_dataset(second\_df)

third\_df = second\_df.iloc[index\_values,[0,1,2,3]]

dt3 = DecisionTreeClassifier(max\_depth=1)

X = second\_df.iloc[:,0:2].values

y = second\_df.iloc[:,2].values

dt3.fit(X,y)

DecisionTreeClassifier(ccp\_alpha=0.0, class\_weight=None, criterion='gini',

                       max\_depth=1, max\_features=None, max\_leaf\_nodes=None,

                       min\_impurity\_decrease=0.0, min\_impurity\_split=None,

                       min\_samples\_leaf=1, min\_samples\_split=2,

                       min\_weight\_fraction\_leaf=0.0, presort='deprecated',

                       random\_state=None, splitter='best')

plot\_decision\_regions(X, y, clf=dt3, legend=2)

third\_df['y\_pred'] = dt3.predict(X)

third\_df

alpha3 = calculate\_model\_weight(0.7)

alpha3

print(alpha1,alpha2,alpha3)

query = np.array([1,5]).reshape(1,2)

dt1.predict(query)

dt2.predict(query)

dt3.predict(query)

alpha1\*1 + alpha2\*(1) + alpha3\*(1)

np.sign(1.09)

query = np.array([9,9]).reshape(1,2)

dt1.predict(query)

dt2.predict(query)

dt3.predict(query)

alpha1\*(1) + alpha2\*(-1) + alpha3\*(-1)

np.sign(-0.25)

1. **Implementation of ANN**

#import libraries

import numpy as np

import pandas as pd

import tensorflow as tf

#importing dataset

dataset = pd.read\_csv('Churn\_Modelling.csv')

#partial view of dataset from top

dataset.head()

#partial view of dataset from bottom

dataset.tail()

#dimention of the dataset

dataset.shape

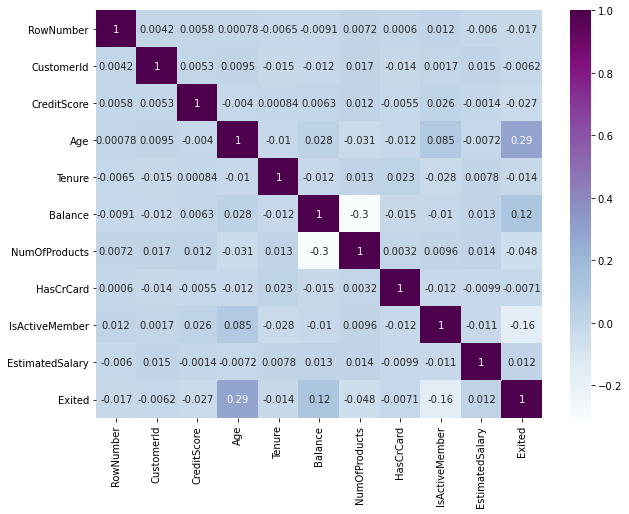
#finding correlation between the features

corr\_var=dataset.corr()

print(corr\_var)

plt.figure(figsize=(10,7.5))

sns.heatmap(corr\_var, annot=True, cmap='BuPu')



#as there is no importance in cust id, row no and sur name for modelling we are not included here in independent feature

X = dataset.iloc[:, 3:-1].values

#target value

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

#as we have two columns as categorical terms we go for encoding we need to convert to numericals

#Categorical encoding

#gender will have some correlation with other feature so we go for label encoding

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

#gender column in index 2

X[:, 2] = le.fit\_transform(X[:, 2])

#country name wont be that much correlation added it has more than 2 names so go for one hot encoding

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import OneHotEncoder

#country name is present in 1st index value

ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])], remainder='passthrough')

X = np.array(ct.fit\_transform(X))

#training and testing split

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3, random\_state = 0)

#feature scaling is an important and mandatory for ann process before modelling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

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

#ANN - initializing

ann = tf.keras.models.Sequential()

#input layer

# 6 features

ann.add(tf.keras.layers.Dense(units=6, activation='relu'))

#hidden layer

ann.add(tf.keras.layers.Dense(units=6, activation='relu'))

#output layer

#as target value is binary - AF

ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

#compiling

#loss - target is binary

ann.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

#training set

ann.fit(X\_train, y\_train, batch\_size = 32, epochs = 50)

#test result - prediction

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

#insted of values we ll get 0 or 1

y\_pred = (y\_pred > 0.5)

#actual vs prediicted outputs

print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))

#accuracy and confusion matrix

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

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

print(cm)

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

Use our ANN model to predict if the customer with the following informations will leave the bank:

Geography: France

Credit Score: 750

Gender: Female

Age: 48 years old

Tenure: 5 years

Balance: $ 62500

Number of Products: 3

Does this customer have a credit card ? Yes

Is this customer an Active Member: Yes

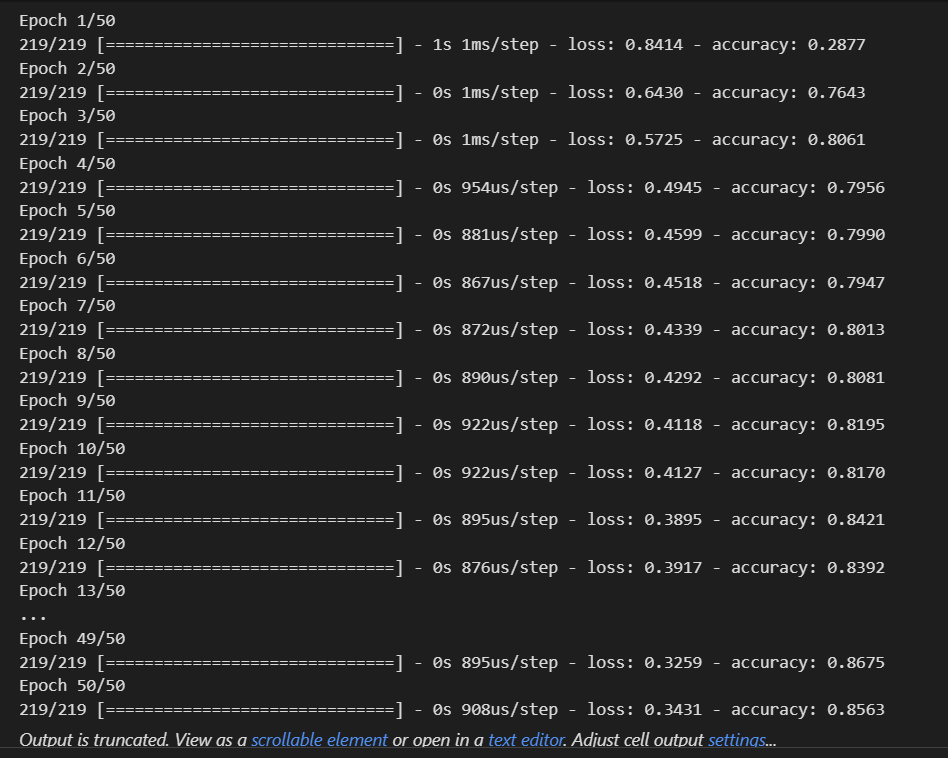
Estimated Salary: $ 80000

So, should we say goodbye to that customer ?

#for predicting single sample

print(ann.predict(sc.transform([[1, 0, 0, 750, 0, 48,53, 62500, 3, 1, 1, 80000]])) > 0.5)

****

****

1. **Implementation of CNN**

import mnist

import numpy as np

from conv import Conv3x3

from maxpool import MaxPool2

from softmax import Softmax

# We only use the first 1k examples of each set in the interest of time.

# Feel free to change this if you want.

train\_images = mnist.train\_images()[:1000]

train\_labels = mnist.train\_labels()[:1000]

test\_images = mnist.test\_images()[:1000]

test\_labels = mnist.test\_labels()[:1000]

conv = Conv3x3(8)                  # 28x28x1 -> 26x26x8

pool = MaxPool2()                  # 26x26x8 -> 13x13x8

softmax = Softmax(13 \* 13 \* 8, 10) # 13x13x8 -> 10

def forward(image, label):

  '''

  Completes a forward pass of the CNN and calculates the accuracy and

  cross-entropy loss.

  - image is a 2d numpy array

  - label is a digit

  '''

  # We transform the image from [0, 255] to [-0.5, 0.5] to make it easier

  # to work with. This is standard practice.

  out = conv.forward((image / 255) - 0.5)

  out = pool.forward(out)

  out = softmax.forward(out)

  # Calculate cross-entropy loss and accuracy. np.log() is the natural log.

  loss = -np.log(out[label])

  acc = 1 if np.argmax(out) == label else 0

  return out, loss, acc

def train(im, label, lr=.005):

  '''

  Completes a full training step on the given image and label.

  Returns the cross-entropy loss and accuracy.

  - image is a 2d numpy array

  - label is a digit

  - lr is the learning rate

  '''

  # Forward

  out, loss, acc = forward(im, label)

  # Calculate initial gradient

  gradient = np.zeros(10)

  gradient[label] = -1 / out[label]

  # Backprop

  gradient = softmax.backprop(gradient, lr)

  gradient = pool.backprop(gradient)

  gradient = conv.backprop(gradient, lr)

  return loss, acc

print('MNIST CNN initialized!')

# Train the CNN for 3 epochs

for epoch in range(3):

  print('--- Epoch %d ---' % (epoch + 1))

  # Shuffle the training data

  permutation = np.random.permutation(len(train\_images))

  train\_images = train\_images[permutation]

  train\_labels = train\_labels[permutation]

  # Train!

  loss = 0

  num\_correct = 0

  for i, (im, label) in enumerate(zip(train\_images, train\_labels)):

    if i % 100 == 99:

      print(

        '[Step %d] Past 100 steps: Average Loss %.3f | Accuracy: %d%%' %

        (i + 1, loss / 100, num\_correct)

      )

      loss = 0

      num\_correct = 0

    l, acc = train(im, label)

    loss += l

    num\_correct += acc

# Test the CNN

print('\n--- Testing the CNN ---')

loss = 0

num\_correct = 0

for im, label in zip(test\_images, test\_labels):

  \_, l, acc = forward(im, label)

  loss += l

  num\_correct += acc

num\_tests = len(test\_images)

print('Test Loss:', loss / num\_tests)

print('Test Accuracy:', num\_correct / num\_tests)

1. **Implementation of KNN**

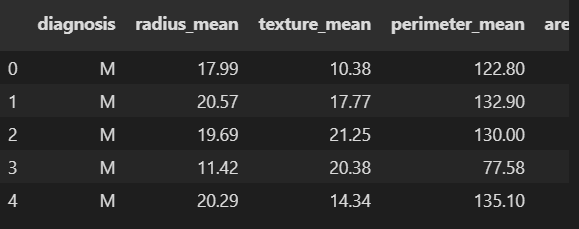
import numpy as np

import pandas as pd

df = pd.read\_csv('data (1).csv')

df.drop(columns=['id','Unnamed: 32'],inplace=True)

df.head()



df.shape



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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.iloc[:,1:], df.iloc[:,0],test\_size=0.2, random\_state=2)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

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

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

from sklearn.neighbors import KNeighborsClassifier

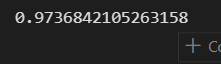
knn = KNeighborsClassifier(n\_neighbors=5)

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

from sklearn.metrics import accuracy\_score

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

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



scores = []

for i in range(1,100):

    knn = KNeighborsClassifier(n\_neighbors=i)

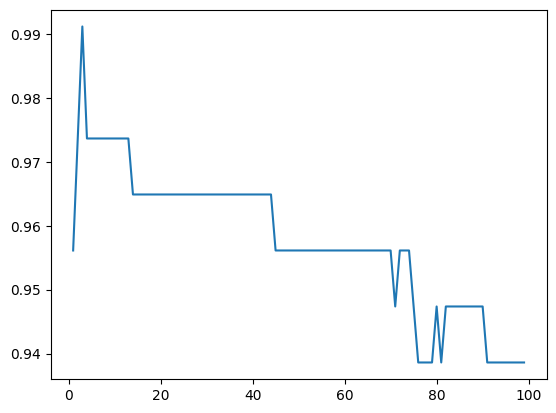
    knn.fit(X\_train,y\_train)

    y\_pred = knn.predict(X\_test)

    scores.append(accuracy\_score(y\_test, y\_pred))

import matplotlib.pyplot as plt

plt.plot(range(1,100),scores)

****

1. **Implementation of K-Means clustering**

from sklearn.datasets import make\_blobs

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

import pandas as pd

#centroids = [(-5,-5),(5,5),(-2.5,2.5),(2.5,-2.5)]

#cluster\_std = [1,1,1,1]

#X,y = make\_blobs(n\_samples=100,cluster\_std=cluster\_std,centers=centroids,n\_features=2,random\_state=2)

#plt.scatter(X[:,0],X[:,1])

df = pd.read\_csv('student\_clustering.csv')

X = df.iloc[:,:].values

km = KMeans(n\_clusters=4,max\_iter=500)

y\_means = km.fit\_predict(X)

plt.scatter(X[y\_means == 0,0],X[y\_means == 0,1],color='red')

plt.scatter(X[y\_means == 1,0],X[y\_means == 1,1],color='blue')

plt.scatter(X[y\_means == 2,0],X[y\_means == 2,1],color='green')

plt.scatter(X[y\_means == 3,0],X[y\_means == 3,1],color='yellow')

plt.show()

