# Run this program on your local python

# interpreter, provided you have installed

# the required libraries.

# Importing the required packages

import numpy as np

import pandas as pd

from sklearn.metrics import confusion\_matrix

from sklearn.cross\_validation import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

from sklearn.metrics import classification\_report

# Function importing Dataset

def importdata():

balance\_data = pd.read\_csv(

'https://archive.ics.uci.edu/ml/machine-learning-databases/blood-transfusion/transfusion.data',

sep= ',', header = 0 )

# Printing the dataswet shape

print ("Dataset Lenght: ", len(balance\_data))

print ("Dataset Shape: ", balance\_data.shape)

# Printing the dataset obseravtions

print ("Dataset: ",balance\_data.head())

return balance\_data

# Function to split the dataset

def splitdataset(balance\_data):

# Seperating the target variable

X = balance\_data.values[:, 0:4]

Y = balance\_data.values[:, 4]

# Spliting the dataset into train and test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, Y, test\_size = 0.3, random\_state = 100)

return X, Y, X\_train, X\_test, y\_train, y\_test

# Function to perform training with giniIndex.

def train\_using\_gini(X\_train, X\_test, y\_train):

# Creating the classifier object

clf\_gini = DecisionTreeClassifier(criterion= "gini",

random\_state = 100,max\_depth=3, min\_samples\_leaf=5)

# Performing training

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

return clf\_gini

# Function to perform training with entropy.

def tarin\_using\_entropy(X\_train, X\_test, y\_train):

# Decision tree with entropy

clf\_entropy = DecisionTreeClassifier(

criterion = "entropy", random\_state = 100,

max\_depth = 3, min\_samples\_leaf = 5)

# Performing training

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

return clf\_entropy

# Function to make predictions

def prediction(X\_test, clf\_object):

# Predicton on test with giniIndex

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

print("Predicted values:")

print(y\_pred)

return y\_pred

# Function to calculate accuracy

def cal\_accuracy(y\_test, y\_pred):

print("Confusion Matrix: ",

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

print ("Accuracy : ",

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

print("Report : ",

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

# Driver code

def main():

# Building Phase

data = importdata()

X, Y, X\_train, X\_test, y\_train, y\_test = splitdataset(data)

clf\_gini = train\_using\_gini(X\_train, X\_test, y\_train)

clf\_entropy = tarin\_using\_entropy(X\_train, X\_test, y\_train)

# Operational Phase

print("Results Using Gini Index:")

# Prediction using gini

y\_pred\_gini = prediction(X\_test, clf\_gini)

cal\_accuracy(y\_test, y\_pred\_gini)

print("Results Using Entropy:")

# Prediction using entropy

y\_pred\_entropy = prediction(X\_test, clf\_entropy)

cal\_accuracy(y\_test, y\_pred\_entropy)

# Calling main function

if \_\_name\_\_=="\_\_main\_\_":

main()

import numpy as np

import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import confusion\_matrix

from sklearn.cross\_validation import train\_test\_split

from sklearn.metrics import accuracy\_score

from sklearn.metrics import classification\_report

balance\_data = pd.read\_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/blood-transfusion/transfusion.data',

sep= ',', header = 0 )

print ("Dataset Lenght: ", len(balance\_data))

print ("Dataset Shape: ", balance\_data.shape)

print ("Dataset: ",balance\_data.head())

X = balance\_data.values[:, 0:4]

Y = balance\_data.values[:, 4]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, Y, test\_size = 0.3, random\_state = 100)

mlp = MLPClassifier(hidden\_layer\_sizes=(13,13,13),max\_iter=1750)

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

predictions = mlp.predict(X\_test)

print(confusion\_matrix(y\_test,predictions))

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

import numpy as np

import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.neural\_network import MLPClassifier

from sklearn import datasets, svm

from sklearn.metrics import confusion\_matrix

from sklearn.cross\_validation import train\_test\_split

from sklearn.metrics import accuracy\_score

from sklearn.metrics import classification\_report

balance\_data = pd.read\_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/glass/glass.data',

sep= ',', header = 0 )

print ("Dataset Lenght: ", len(balance\_data))

print ("Dataset Shape: ", balance\_data.shape)

print ("Dataset: ",balance\_data.head())

X = balance\_data.values[:, 1:9]

Y = balance\_data.values[:, 10]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size = 0.3, random\_state = 100)

C = 1.0 # SVM regularization parameter

# SVC with linear kernel

svc = svm.SVC(kernel='linear', C=C).fit(X, Y)

predicted= svc.predict(X\_test)

cnf\_matrix = confusion\_matrix(y\_test, predicted)

print("Confusion matrix for SVC")

print(cnf\_matrix)

# LinearSVC (linear kernel)

#lin\_svc = svm.LinearSVC(C=C).fit(X, Y)

#predicted= lin\_svc.predict(X\_test)

#cnf\_matrix = confusion\_matrix(y\_test, predicted)

#print("Confusion matrix for LINEAR kernel")

#print(cnf\_matrix)

# SVC with RBF kernel

rbf\_svc = svm.SVC(kernel='rbf', gamma=0.7, C=C).fit(X, Y)

predicted= rbf\_svc.predict(X\_test)

cnf\_matrix = confusion\_matrix(y\_test, predicted)

print("Confusion matrix for RBF kernel")

print(cnf\_matrix)

# SVC with polynomial (degree 3) kernel

poly\_svc = svm.SVC(kernel='poly', degree=3, C=C).fit(X, Y)

predicted= poly\_svc.predict(X\_test)

cnf\_matrix = confusion\_matrix(y\_test, predicted)

print("Confusion matrix for POLYNOMIAL kernel")

print(cnf\_matrix)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)  
from sklearn.linear\_model import LogisticRegression  
from sklearn import metrics  
logreg = LogisticRegression()  
logreg.fit(X\_train, y\_train)

y\_pred = logreg.predict(X\_test)  
print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(logreg.score(X\_test, y\_test)))

from sklearn import model\_selection  
from sklearn.model\_selection import cross\_val\_score  
kfold = model\_selection.KFold(n\_splits=10, random\_state=7)  
modelCV = LogisticRegression()  
scoring = 'accuracy'  
results = model\_selection.cross\_val\_score(modelCV, X\_train, y\_train, cfrom sklearn.metrics import confusion\_matrix  
confusion\_matrix = confusion\_matrix(y\_test, y\_pred)  
print(confusion\_matrix)

from sklearn.metrics import confusion\_matrix  
confusion\_matrix = confusion\_matrix(y\_test, y\_pred)  
print(confusion\_matrix)

v=kfold, scoring=scoring)  
print("10-fold cross validation average accuracy: %.3f" % (results.mean()))

from sklearn.metrics import classification\_report  
print(classification\_report(y\_test, y\_pred))

from sklearn.ensemble import AdaBoostClassifier #[For Classification](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html#sklearn.ensemble.AdaBoostClassifier)

from sklearn.ensemble import AdaBoostRegressor #[For Regression](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html#sklearn.ensemble.AdaBoostRegressor)

from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()

clf = AdaBoostClassifier(n\_estimators=100, base\_estimator=dt,learning\_rate=1)

#Above I have used decision tree as a base estimator, you can use any ML learner as base estimator if it ac# cepts sample weight

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

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

from sklearn.cross\_validation import train\_test\_split

import pandas as pd

import numpy as np

from sklearn.tree import DecisionTreeClassifier

from sklearn import datasets,svm

from sklearn.neural\_network import MLPClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn import datasets

dataset=pd.read\_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/blood-transfusion/transfusion.data',sep=",",header=0)

dataset = datasets.load\_iris()

X=dataset.data[:,0:2]

Y=dataset.target

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.3,random\_state=100)

#classtree=DecisionTreeClassifier(criterion="gini",max\_depth=2)

#classtree=MLPClassifier(hidden\_layer\_sizes=(10,10,12,12,),max\_iter=1000)

classtree=LogisticRegression(C=10)

#print("called")

classtree.fit(X,Y)

#print("fited")

pre=classtree.predict(X\_test)

#print("predicted")

print(accuracy\_score(Y\_test,pre))

print(classification\_report(Y\_test,pre))

print(confusion\_matrix(Y\_test,pre))

from sklearn.ensemble import AdaBoostClassifier

dataset=pd.read\_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/blood-transfusion/transfusion.data',sep=",",header=0)

dataset = datasets.load\_iris()

X=dataset.data[:,0:2]

Y=dataset.target

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.3,random\_state=100)

#classtree=DecisionTreeClassifier(criterion="gini",max\_depth=2)

#classtree=MLPClassifier(hidden\_layer\_sizes=(10,10,12,12,),max\_iter=1000)

classtree=DecisionTreeClassifier(criterion= "gini",random\_state = 100,max\_depth=3, min\_samples\_leaf=5)

classtree=AdaBoostClassifier(n\_estimators=100, base\_estimator=classtree,learning\_rate=1)

#print("called")

classtree.fit(X,Y)

#print("fited")

pre=classtree.predict(X\_test)

#print("predicted")

print(accuracy\_score(Y\_test,pre))

print(classification\_report(Y\_test,pre))

print(confusion\_matrix(Y\_test,pre))