***RANK FEATURES OF SMARTPHONE - Build a Python Application to Classify and Rank Dataset.***

*#import the packages*

import numpy as np # linear algebra

import pandas as pd # data processing

import os

import plotly.offline as pyo

import plotly.express as px

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

warnings.filterwarnings("ignore")

*#load the dataset*

df = pd.read\_csv("/content/train.csv")

*#display first 11rows*

df = pd.DataFrame(df)

df.head(11)

*#data frame description*

df.describe()

*#null check*

df.isnull()

*#countplot*

for i in df:

if(df[i].nunique()<30):

sns.countplot(x=df[i])

plt.show()

*#distplot*

plt.figure(figsize = (30,10))

plt.subplot(331)

sns.distplot(df['battery\_power'])

plt.subplot(332)

sns.distplot(df['clock\_speed'])

plt.subplot(333)

sns.distplot(df['int\_memory'])

plt.subplot(334)

sns.distplot(df['m\_dep'])

plt.subplot(335)

sns.distplot(df['mobile\_wt'])

plt.subplot(336)

sns.distplot(df['px\_height'])

plt.subplot(337)

sns.distplot(df['px\_width'])

plt.subplot(338)

sns.distplot(df['ram'])

plt.subplot(339)

sns.distplot(df['talk\_time'])

plt.show()

*#bluetooth*

df["is\_bluetooth"]=''

for i in range(len(df)):

if df['blue'][i]==0:

df['is\_bluetooth'][i]='No'

else:

df['is\_bluetooth'][i]='Yes'

px.pie(data\_frame = df, names = 'is\_bluetooth', title = 'Percentage of devices having bluetooth', hole= 0.2)

*#dualsim*

df["is\_DualSim"]=''

for i in range(len(df)):

if df['dual\_sim'][i]==0:

df['is\_DualSim'][i]='No'

else:

df['is\_DualSim'][i]='Yes'

px.pie(data\_frame = df, names = 'is\_DualSim', title = 'Percentage of devices having dual sim', hole= 0.2)

*#4G*

df["is\_4G"]=''

for i in range(len(df)):

if df['four\_g'][i]==0:

df['is\_4G'][i]='No'

else:

df['is\_4G'][i]='Yes'

px.pie(data\_frame = df, names = 'is\_4G', title = 'Percentage of devices having 4G connection', hole= 0.2)

*#3G*

df["is\_3G"]=''

for i in range(len(df)):

if df['three\_g'][i]==0:

df['is\_3G'][i]='No'

else:

df['is\_3G'][i]='Yes'

px.pie(data\_frame = df, names = 'is\_3G', title = 'Percentage of devices having 3G connection', hole= 0.2)

*#touchscreen*

df["is\_touchscreen"]=''

for i in range(len(df)):

if df['touch\_screen'][i]==0:

df['is\_touchscreen'][i]='No'

else:

df['is\_touchscreen'][i]='Yes'

px.pie(data\_frame = df, names = 'is\_touchscreen', title = 'Percentage of devices having touch screen', hole= 0.2)

*#wifi*

df["is\_wifi"]=''

for i in range(len(df)):

if df['wifi'][i]==0:

df['is\_wifi'][i]='No'

else:

df['is\_wifi'][i]='Yes'

px.pie(data\_frame = df, names = 'is\_wifi', title = 'Percentage of devices having Wifi', hole= 0.2)

*#processors*

df["cores"]=''

for i in range(len(df)):

if df['n\_cores'][i]==1:

df['cores'][i]='single core'

elif df['n\_cores'][i]==2:

df['cores'][i]='dual core'

elif df['n\_cores'][i]==3:

df['cores'][i]='triple core'

elif df['n\_cores'][i]==4:

df['cores'][i]='quad core'

elif df['n\_cores'][i]==5:

df['cores'][i]='penta core'

elif df['n\_cores'][i]==6:

df['cores'][i]='hexa core'

elif df['n\_cores'][i]==7:

df['cores'][i]='hepta core'

else:

df['cores'][i]='octa core'

px.pie(data\_frame = df, names = 'cores', title = 'Percentage of devices having different types of cores', hole= 0.2)

*#training set*

df1 = df.loc[:,['battery\_power','blue','dual\_sim','fc','four\_g','int\_memory','m\_dep','mobile\_wt','n\_cores','pc','px\_height','px\_width','ram','sc\_h','sc\_w','talk\_time','three\_g','wifi','price\_range']]

df1

df2 = df.loc[:,['touch\_screen','clock\_speed']]

df2

*#merge the datasets*

df3 = pd.concat([df1, df2])

df3

#*ranking according to price range*

df["rank\_by\_price"] = df["price\_range"].rank()

dt1 = df

dt1.head()

*#sorting*

dt1["rank\_by\_price"] = dt1["rank\_by\_price"].sort\_values()

dt1

dt1.sort\_values(by=["rank\_by\_price"])

dt1.head()

dt2 = pd.read\_csv(r"/content/train.csv")

RankedDataset1 = dt2.rank()

RankedDataset1.sort\_values(by="price\_range")

b = dt2

b["rank\_by\_price"] = b["price\_range"].rank()

b["rank\_by\_battery"] = b["battery\_power"].rank(ascending=False)

b["rank\_by\_blueooth"] = b["blue"].rank(ascending=False)

b["rank\_by\_clockspeed"] = b["clock\_speed"].rank(ascending=False)

b["rank\_by\_DualSIM"] = b["dual\_sim"].rank(ascending=False)

b["rank\_by\_fc"] = b["fc"].rank(ascending=False)

b["rank\_by\_4G"] = b["four\_g"].rank(ascending=False)

b["rank\_by\_InternalMemory"] = b["int\_memory"].rank(ascending=False)

b["rank\_by\_mdep"] = b["m\_dep"].rank(ascending=False)

b["rank\_by\_weight"] = b["mobile\_wt"].rank(ascending=True)

b["rank\_by\_ncores"] = b["n\_cores"].rank(ascending=False)

b["rank\_by\_pc"] = b["pc"].rank(ascending=False)

b["rank\_by\_height"] = b["px\_height"].rank(ascending=False)

b["rank\_by\_width"] = b["px\_width"].rank(ascending=False)

b["rank\_by\_ram"] = b["ram"].rank(ascending=False)

b["rank\_by\_sch"] = b["sc\_h"].rank(ascending=False)

b["rank\_by\_scw"] = b["sc\_w"].rank(ascending=False)

b["rank\_by\_talktime"] = b["talk\_time"].rank(ascending=False)

b["rank\_by\_3G"] = b["three\_g"].rank(ascending=False)

b["rank\_by\_touchscreen"] = b["touch\_screen"].rank(ascending=False)

b["rank\_by\_wifi"] = b["wifi"].rank(ascending=False)

b.head()

RankedDataset2 = b.iloc[:,21:]

RankedDataset2

*#logistic regression*

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

import sklearn

df = pd.read\_csv(r"/content/train.csv")

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

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

from sklearn.preprocessing import StandardScaler

sc\_x = StandardScaler()

x = sc\_x.fit\_transform(x)

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)

log=LogisticRegression()

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

print("Training score of LogisticRegression is: {}".format(log.score(x\_train,y\_train)\*100))

y\_predlog=log.predict(x\_test)

print(y\_predlog)

acrr=accuracy\_score(y\_test,y\_predlog)\*100

print("Accuracy of Logistic Regression Classifier is: {}%".format(acrr))

print("Confusion matrix Logistic Regression Classifier is: \n{}".format(confusion\_matrix(y\_test,y\_predlog)))

print("{}".format(classification\_report(y\_test,y\_predlog)))

*#svm model*

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

df = pd.read\_csv(r"/content/train.csv")

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

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

from sklearn.preprocessing import StandardScaler

sc\_x = StandardScaler()

x = sc\_x.fit\_transform(x)

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

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

from sklearn.svm import SVC

svm=SVC(kernel='rbf')

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

print("Training score of SVM is: {}".format(svm.score(x\_train,y\_train)\*100))

y\_pred\_svm=svm.predict(x\_test)

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

print("Accuracy of SVM is: {}".format(ac\_svm))

print("Confusion matrix of SVM is: {}".format(confusion\_matrix(y\_test,y\_pred\_svm)))

print("{}".format(classification\_report(y\_test,y\_pred\_svm)))

*#gaussiannb*

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

df = pd.read\_csv(r"/content/train.csv")

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

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

from sklearn.preprocessing import StandardScaler

sc\_x = StandardScaler()

x = sc\_x.fit\_transform(x)

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

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

from sklearn.naive\_bayes import GaussianNB

nb=GaussianNB()

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

print("Training score of GaussianNB is {}".format(nb.score(x\_train,y\_train)\*100))

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

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

print("Accuracy of Naive Bayes Classifier is: {}".format(ac\_nb))

print("Confusion matrix of Naive Bayes Classifier is:{}".format(confusion\_matrix(y\_test,y\_pred)))

print("{}".format(classification\_report(y\_test,y\_pred)))

*#decisiontree*

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

df = pd.read\_csv(r"/content/train.csv")

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

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

from sklearn.preprocessing import StandardScaler

sc\_x = StandardScaler()

x = sc\_x.fit\_transform(x)

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

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

from sklearn.tree import DecisionTreeClassifier

DT=DecisionTreeClassifier(criterion = "entropy")

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

print("Training score of DecisionTreeClassifier is: {}".format(DT.score(x\_train,y\_train)\*100))

y\_pred\_DT=DT.predict(x\_test)

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

print("Accuracy of Decision Tree Classifier is: {}".format(ac\_DT))

print("Confusion matrix of Decision Tree Classifier is: {}".format(confusion\_matrix(y\_test,y\_pred\_DT)))

print("{}".format(classification\_report(y\_test,y\_pred\_DT)))

*#randomforest model*

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

df = pd.read\_csv(r"/content/train.csv")

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

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

from sklearn.preprocessing import StandardScaler

sc\_x = StandardScaler()

x = sc\_x.fit\_transform(x)

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

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

from sklearn.ensemble import RandomForestClassifier

RF=RandomForestClassifier(n\_estimators=300)

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

print("Training score of Random Forest Classifier is: {}".format(RF.score(x\_train,y\_train)\*100))

y\_pred\_RF=RF.predict(x\_test)

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

print("Accuracy of Random Forest Classifier is: {}".format(ac\_RF))

print("Confusion matrix of Random Forest Classifier is: {}".format(confusion\_matrix(y\_test,y\_pred\_RF)))

print("{}".format(classification\_report(y\_test,y\_pred\_RF)))

*#knn*

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

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

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

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

#X = df.iloc[:, [2, 3]].values

#y = df.iloc[:, -1].values

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

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

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

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

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors = 7, metric = 'minkowski', p = 2)

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

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

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

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

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

print("Accuracy of KNN is: {}".format(ac))

print("Confusion matrix of KNN is: {}".format(confusion\_matrix(y\_test,y\_pred)))

print("{}".format(classification\_report(y\_test,y\_pred)))

*#Perceptron*

from numpy import mean

from numpy import std

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

from sklearn.datasets import make\_classification

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import RepeatedStratifiedKFold

from sklearn.linear\_model import Perceptron

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

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

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

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

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

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

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

X, y = make\_classification(n\_samples=1000, n\_features=10, n\_informative=10, n\_redundant=0, random\_state=1)

model = Perceptron()

cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)

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

scores = cross\_val\_score(model, X, y, scoring='accuracy', cv=cv, n\_jobs=-1)

print('Accuracy of Perceptron : %.3f (%.3f)' % (mean(scores)\*100, std(scores)))

print("Confusion matrix of Perceptron is: {}".format(confusion\_matrix(y\_test,y\_pred)))

print("{}".format(classification\_report(y\_test,y\_pred)))

*#classifiers*

classifiers=["Perceptron","LogisticRegression","GaussianNB","SVM","DecisionTreeClassifier","RandomForestClassifier","KNN"]

accuracy\_=[ac,acrr,ac\_nb,ac\_svm,ac\_DT,ac\_RF,ac]

df\_ac=pd.DataFrame({'model':classifiers,"accuracy":accuracy\_})

px.histogram(data\_frame=df\_ac,x="model",y="accuracy")

*#random forest regressor*

from sklearn.ensemble import RandomForestRegressor

train\_accuracy=[]

test\_accuracy=[]

for i in range(100,600,100):

c=RandomForestRegressor(n\_estimators=i)

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

train\_accuracy.append(c.score(x\_train,y\_train))

test\_accuracy.append(c.score(x\_test,y\_test))

frame=pd.DataFrame({"n\_neighbors":range(100,600,100),"train\_accuracy":train\_accuracy,"test\_accuracy":test\_accuracy})

frame

plt.figure(figsize=(10,7))

plt.plot(range(100,600,100),frame["train\_accuracy"],label="Train")

plt.plot(range(100,600,100),frame["test\_accuracy"],label="Test")

plt.title("Optimal Number of Estimators")

plt.xlabel("Estimators")

plt.ylabel("Accuracy")

plt.legend()

plt.show()

print(frame)

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_val\_score

kfold = KFold(n\_splits = 10)

xyz = []

accuracy = []

classifiers=["Perceptron","SVM","Logistic Regression","Decision Tree","Random Forest","GaussianNB","KNN"]

models=[Perceptron(),SVC(kernel='rbf'),LogisticRegression(),DecisionTreeClassifier(),RandomForestClassifier(n\_estimators=300,random\_state=0),GaussianNB(),KNeighborsClassifier()]

for i in models:

model = i

cv\_result=cross\_val\_score(model,x\_train,y\_train,cv=kfold,scoring="accuracy")

cv\_result = cv\_result

xyz.append(cv\_result.mean())

accuracy.append(cv\_result)

cv\_models\_datafeame= pd.DataFrame(xyz, index = classifiers)

cv\_models\_datafeame.columns = ['CV Mean']

cv\_models\_datafeame

cv\_models\_datafeame.sort\_values(['CV Mean'], ascending =[0])

box = pd.DataFrame(accuracy, index = [classifiers])

boxT = box.T

plt.figure(figsize = (10,7))

ax = sns.boxplot(data = boxT, orient = "h", palette = "Set2", width = 0.6)

ax.set\_yticklabels(classifiers)

ax.set\_title('Cross Validation accuracy with different classifiers')

ax.set\_xlabel('Accuracy')

plt.show()

*#This work is done by RAJESHWARI K department of Information Technology doing III year in Mepco Schlenk Engineering College, Sivakasi.*