NAIVE BAYES CLASSIFIER

```
import pandas as pd
import numpy as np
-
data = pd.read_csv('/home/exam/Downloads/covid(For Naive Bayes Program) (1).csv')
-
data
-
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
-
pc_encoded=le.fit_transform(data['pc'].values)
wbc_encoded=le.fit_transform(data['wbc'].values)
mc_encoded=le.fit_transform(data['mc'].values)
ast_encoded=le.fit_transform(data['ast'].values)
bc_encoded=le.fit_transform(data['bc'].values)
ldh_encoded=le.fit_transform(data['ldh'].values)
Y=le.fit_transform(data['diagnosis'].values)
X=np.array(list(zip(pc_encoded,wbc_encoded,mc_encoded,ast_encoded,bc_encoded,ldh_encoded))
)
-
X
-
Y
-
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
model = MultinomialNB()
-
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y)
```

```
-
model.fit(X_train, Y_train)
y_pred = model.predict(X_test)
-
print("Accuracy:",accuracy_score(Y_test, y_pred))
-
print("\nReport")
print(classification_report(Y_test,y_pred))
-
lr_probs = model.predict_proba(xtest)   X_test
-
type(lr_probs)
-
lr_probs
-
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
-
lr_probs
-                    from sklearn.metrics import roc_curve
lr_fpr, lr_tpr, _=roc_curve(ytest, lr_probs)   Y_test
-
lr_fpr
-
lr-tpr
-
from matplotlib import pyplot
pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Naive Bayes Classifier')
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
# show the legend
```

```
pyplot.legend()
# show the plot
pyplot.show()
```

SVM

```
from sklearn.svm import SVC
from sklearn import svm
import numpy as np
-
X=np.array([[3,4],[1,4],[2,3],[6,-1],[7,-1],[5,-3]])
y=np.array([-1,-1,-1,1,1,1])
l=SVC(C=1e5,kernel="linear")
l.fit(X,y)
-

print("w=",l.coef_)
print("b=",l.intercept_)
print("Indices of support vectors=",l.support_)
print("Support vectors=",l.support_vectors_)
print("No. of support vectors from each class=",l.n_support_)
print("coefficient of support vectors in decision function=",np.abs(l.dual_coef_))
-
import pandas as pd
data=pd.read_csv("/home/exam/Downloads/B2_heart.csv")
data.head()
x=data.drop("target",axis=1)
y=data.target
-
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)
linear=svm.SVC(kernel="linear")
```

```
linear.fit(x_train,y_train)
print(linear.support_vectors_)
print(linear.n_support_)
y_pred=linear.predict(x_test)
-
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test,y_pred))
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test,y_pred))
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
-
model1=SVC(kernel="sigmoid")
model2=SVC(kernel="poly")
model3=SVC(kernel="rbf")
model1.fit(x_train,y_train)
model2.fit(x_train,y_train)
model3.fit(x_train,y_train)
y_pred1=model1.predict(x_test)
y_pred2=model2.predict(x_test)
y_pred3=model3.predict(x_test)
-
print("prediction by model1",accuracy_score(y_test,y_pred1))
print("prediction by model2",accuracy_score(y_test,y_pred2))
print("prediction by model3",accuracy_score(y_test,y_pred3))
-
print("prediction by model1",confusion_matrix(y_test,y_pred1))
print("prediction by model2",confusion_matrix(y_test,y_pred2))
print("prediction by model3",confusion_matrix(y_test,y_pred3))
-
print("prediction by model1",classification_report(y_test,y_pred1))
print("prediction by model2",classification_report(y_test,y_pred2))
```

print("prediction by model3",classification_report(y_test,y_pred3))

RANDOM FOREST CLASSIFIER

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
data = pd.read_csv('/home/exam/Downloads/B3-pima.csv')
-
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler, MinMaxScaler
#import pandas_profiling
from matplotlib import rcParams
import warnings
data.columns
-
data.sample(5)
-
X=data.drop("Outcome",axis=1)
y=data["Outcome"]
scaler=StandardScaler()
X_scaled=scaler.fit_transform(X)
X_train,X_test,Y_train,Y_test=train_test_split(X_scaled,y,stratify=y,test_size=0.10,random_state=34)
classifier = RandomForestClassifier(n_estimators=100)
classifier.fit(X_train,Y_train)
-
y_pred = classifier.predict(X_test)
```

```
print("Accuracy:",accuracy_score(Y_test,y_pred))
-
feature_importances_df = pd.DataFrame(
   {"feature":list(X.columns),"importance":classifier.feature_importances_}
).sort_values("importance",ascending=False)

feature_importances_df
-
from sklearn.tree import DecisionTreeClassifier
clf=DecisionTreeClassifier()
clf.fit(X_train,Y_train)
Y_pred = clf.predict(X_test)
from sklearn.metrics import accuracy_score
print("Accuracy-DecisionTree :",accuracy_score(Y_test,Y_pred))
-
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Assuming you have already trained a Decision Tree classifier (dt_classifier)
plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True, rounded=True)
plt.show()
```