# akilus-sayadat-037-iris-dataset

August 22, 2023

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[ ]:
```
from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(iris.data, columns = iris.feature_names)
column_names = list(df.columns.values)
df.head()
```

[ ]:
```
X = df.iloc[:][:]
y = iris["target"]
dict_bnb = {}
dict_mnb = {}
dict_gnb = {}
dict_dtr = {}
RocAucbnb = {}
RocAucmnb = {}
RocAucgnb = {}
RocAucdtr = {}
print(X, y)
```

[ ]:
```
def plot(y_test, y_pred):
  from sklearn.metrics import confusion_matrix
  import seaborn as sns

  print("Confusion Matrix : ")
  cf_matrix = confusion_matrix(y_test, y_pred)
  group_counts = ["{0:0.0f}".format(value) for value in
                  cf_matrix.flatten()]
  group_percentages = ["{0:.2%}".format(value) for value in
                       cf_matrix.flatten()/np.sum(cf_matrix)]
  labels = [f"{v1}\n{v2}" for v1, v2 in
            zip(group_counts,group_percentages)]
  labels = np.asarray(labels).reshape(3,3)
  plt.figure(figsize=(6, 4))
```

```
  sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues', xticklabels = iris.
↪target_names, yticklabels=iris.target_names)
  plt.xlabel('Predicted')
  plt.ylabel('Actual')
  plt.title('Confusion Matrix')
  plt.show()
  print("*********************************************")
```

[ ]:
```
def reports(y_test, y_pred):
  from sklearn.metrics import classification_report
  plot(y_test, y_pred)
  print("*********************************************")
  print("Classification Evaluation : ")
  print(classification_report(y_test, y_pred, zero_division = 0))
```

### 0.0.1 Classification using BernoulliNB Naive Bayes

[ ]:
```
def FBouBernoulli(split, alpha_value = 1.0, binarize_value = 0.0,␣
↪fit_prior_value = False):
  from sklearn.naive_bayes import BernoulliNB
  from sklearn.metrics import accuracy_score
  from sklearn.model_selection import train_test_split
  from sklearn.preprocessing import StandardScaler
  #scaler = StandardScaler()
  X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = split,␣
↪random_state=44)
  #scaler.fit_transform(X_train)
  #scaler.transform(X_test)
  classifier = BernoulliNB(alpha = alpha_value, binarize = binarize_value,␣
↪fit_prior = fit_prior_value)
  classifier.fit(X_train, y_train)
  y_pred = classifier.predict(X_test)
  print("Train-test split: " + str(split))
  print("value: alpha: "+str(alpha_value) + " binarize: " + str(binarize_value)␣
↪+ " fit_prior: " +str(fit_prior_value))
  print("*********************************************")
  accuracy = accuracy_score(y_test, y_pred)
  if str(split) in dict_bnb:
    if dict_bnb[str(split)][0] < accuracy:
      dict_bnb[str(split)] = [accuracy, y_test, y_pred]
    if str(split) == '0.3' and accuracy > dict_bnb[str(split)][0]:
      RocAucbnb['max'] = {'y_test': y_test, 'y_pred': y_pred}
  else:
    dict_bnb[str(split)] = [accuracy, y_test, y_pred]
    if str(split) == '0.3':
      RocAucbnb['max'] = {'y_test': y_test, 'y_pred': y_pred}
```

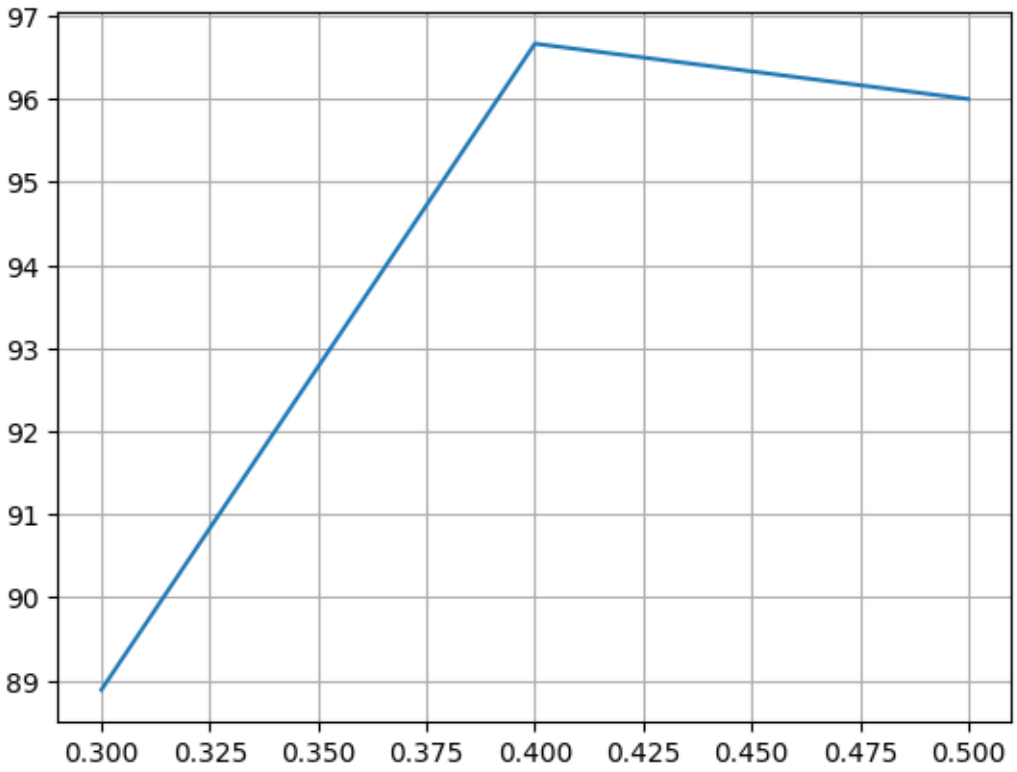
```
  reports(y_test, y_pred)
```

[ ]:
```
## Train-Test split 0.3
FBouBernoulli(0.3)
FBouBernoulli(0.3, 1.0)
FBouBernoulli(0.3, 1.0, 1.8)
FBouBernoulli(0.3, 1.0, 1.8, True)
```

[ ]:
```
## Train-Test split 0.4
FBouBernoulli(0.4)
FBouBernoulli(0.4, 1.0)
FBouBernoulli(0.4, 1.0, 1.7)
FBouBernoulli(0.4, 1.0, 1.7, True)
```

[ ]:
```
## Train-Test split 0.5
FBouBernoulli(0.5)
FBouBernoulli(0.5, 1.0)
FBouBernoulli(0.5, 1.0, 1.75)
FBouBernoulli(0.5, 1.0, 1.75, True)
```

[46]:
```
keys = dict_bnb.keys()
y_points = []
for key in keys:
  y_points.append(dict_bnb[key][0]*100)
x_points = [float(key) for key in dict_bnb.keys()]

plt.plot(x_points, y_points)
plt.grid(True)
plt.show()
```




### 0.1 Classification using Multinomial Naive Bayes

[ ]:
```
def FMultinomial(split, alpha_value = 1.0):
  from sklearn.naive_bayes import MultinomialNB
  from sklearn.metrics import accuracy_score
  from sklearn.model_selection import train_test_split
  X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = split)
  classifier = MultinomialNB(alpha = alpha_value)
  classifier.fit(X_train, y_train)
  y_pred = classifier.predict(X_test)
  print("Train-test split: " + str(split))
  print("value: alpha: "+str(alpha_value))
  print("*********************************************")
  accuracy = accuracy_score(y_test, y_pred)
  if str(split) in dict_mnb:
    if dict_mnb[str(split)][0] < accuracy:
      dict_mnb[str(split)] = [accuracy, y_test, y_pred]
    if str(split) == '0.3' and accuracy > dict_mnb[str(split)][0]:
      RocAucmnb['max'] = {'y_test': y_test, 'y_pred': y_pred}
  else:
    dict_mnb[str(split)] = [accuracy, y_test, y_pred]
```

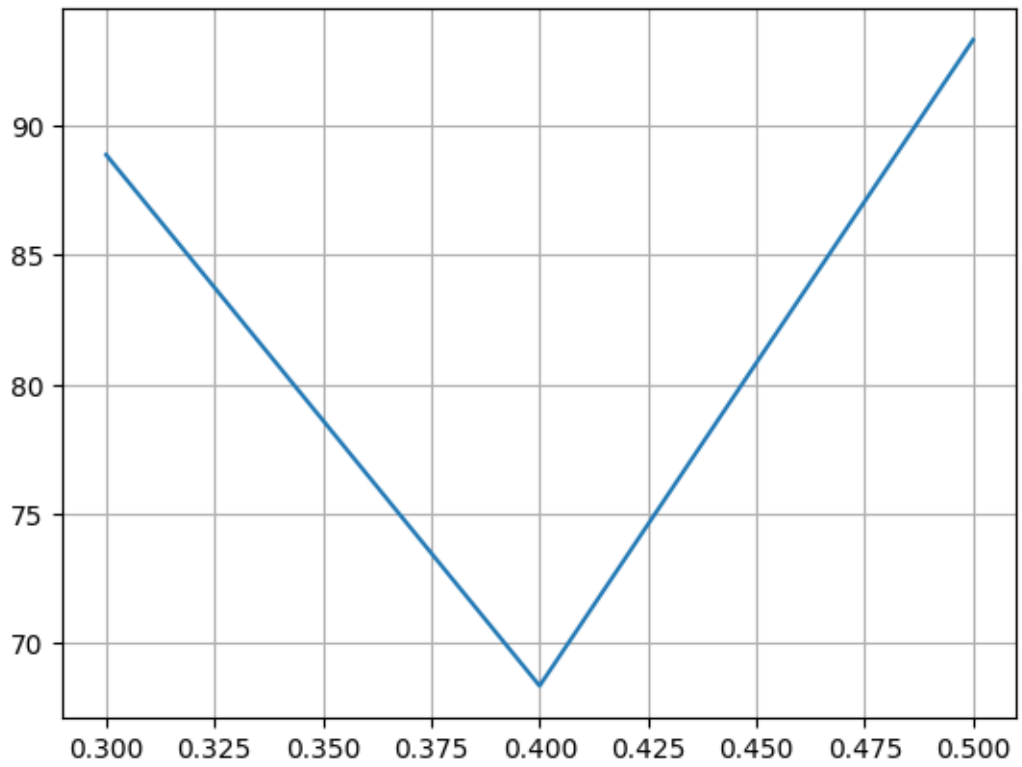
```
    if str(split) == '0.3':
      RocAucmnb['max'] = {'y_test': y_test, 'y_pred': y_pred}
  reports(y_test, y_pred)
```

[ ]:
```
## Train-Test split 0.3
FMultinomial(0.3)
FMultinomial(0.3, 1.6)
```

[ ]:
```
## Train-Test split 0.4
FMultinomial(0.4)
FMultinomial(0.4, 1.4)
```

[ ]:
```
## Train-Test split 0.5
FMultinomial(0.5)
FMultinomial(0.5, 1.5)
```

[45]:
```
keys = dict_mnb.keys()
y_points = []
for key in keys:
  y_points.append(dict_mnb[key][0]*100)
x_points = [float(key) for key in dict_mnb.keys()]

plt.plot(x_points, y_points)
plt.grid(True)
plt.show()
```




### 0.1.1 Classification using Guassian Naive Bayes

[ ]:
```
def FGaussian(split):
  from sklearn.naive_bayes import GaussianNB
  from sklearn.model_selection import train_test_split
  from sklearn.metrics import accuracy_score
  from sklearn.preprocessing import StandardScaler
  scaler = StandardScaler()
  X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = split,␣
 ↪random_state=44)
  scaler.fit_transform(X_train)
  scaler.transform(X_test)
  classifier = GaussianNB()
  classifier.fit(X_train, y_train)
  y_pred = classifier.predict(X_test)
  print("Train-test split: " + str(split))
  print("**********************************************")
  reports(y_test, y_pred)
  accuracy = accuracy_score(y_test, y_pred)
  if str(split) in dict_gnb:
    if dict_gnb[str(split)][0] < accuracy:
      dict_gnb[str(split)] = [accuracy, y_test, y_pred]
```

```
    if str(split) == '0.3' and accuracy > dict_gnb[str(split)][0]:
      RocAucgnb['max'] = {'y_test': y_test, 'y_pred': y_pred}
  else:
    dict_gnb[str(split)] = [accuracy, y_test, y_pred]
    if str(split) == '0.3':
      RocAucgnb['max'] = {'y_test': y_test, 'y_pred': y_pred}
```

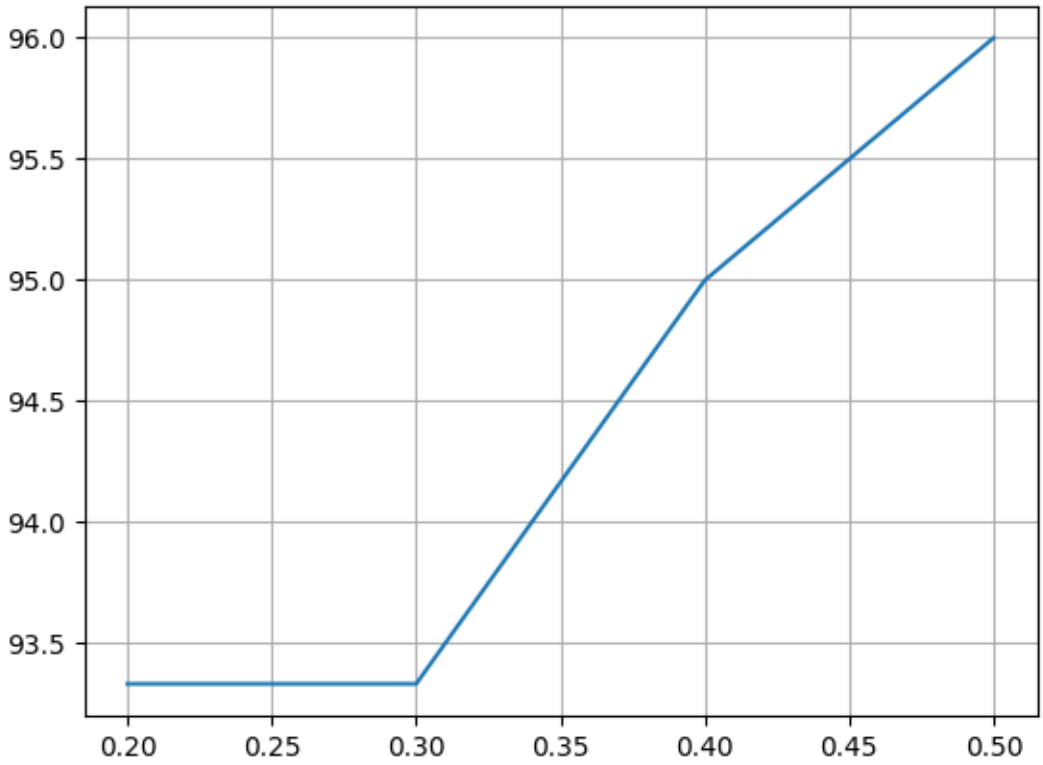
[ ]:
```
## Train-Test split 0.2
FGaussian(0.2)
```

[ ]:
```
## Train-Test split 0.3
FGaussian(0.3)
```

[ ]:
```
## Train-Test split 0.4
FGaussian(0.4)
```

[ ]:
```
## Train-Test split 0.5
FGaussian(0.5)
```

[44]:
```
keys = dict_gnb.keys()
y_points = []
for key in keys:
  y_points.append(dict_gnb[key][0]*100)
x_points = [float(key) for key in dict_gnb.keys()]

plt.plot(x_points, y_points)
plt.grid(True)
plt.show()
```




### 0.1.2 Classification using Decision Tree

[ ]:
```
def decision_tree(split, criterion_value):
  from sklearn.model_selection import train_test_split
  from sklearn.tree import DecisionTreeClassifier
  from sklearn import tree
  from sklearn.metrics import accuracy_score
  from sklearn.preprocessing import StandardScaler
  scaler = StandardScaler()
  X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = split,␣
 ↪random_state=44)
  scaler.fit_transform(X_train)
  scaler.transform(X_test)

  classifier = DecisionTreeClassifier(criterion = criterion_value)
  classifier.fit(X_train, y_train)
  y_pred = classifier.predict(X_test)
  print("Train-test split: " + str(split))
  print("Value: Entropy: " + criterion_value)
  print("*********************************************")
  reports(y_test, y_pred)
  accuracy = accuracy_score(y_test, y_pred)
```

```
  if str(split) in dict_dtr:
    if dict_dtr[str(split)][0] < accuracy:
      dict_dtr[str(split)] = [accuracy, y_test, y_pred]
    if str(split) == '0.3' and accuracy > dict_dtr[str(split)][0]:
      RocAucdtr['max'] = {'y_test': y_test, 'y_pred': y_pred}
  else:
    dict_dtr[str(split)] = [accuracy, y_test, y_pred]
    if str(split) == '0.3':
      RocAucdtr['max'] = {'y_test': y_test, 'y_pred': y_pred}

  reports(y_test, y_pred)
  fig = plt.figure(figsize=(12,8))
  _ = tree.plot_tree(classifier,
                     feature_names=column_names,
                     class_names=['outcome1', 'outcome2', 'output3'],
                     filled=True)
```

[ ]:
```
decision_tree(0.2, 'entropy')
```

[ ]:
```
decision_tree(0.2, 'gini')
```

[ ]:
```
decision_tree(0.3, 'entropy')
```

[ ]:
```
decision_tree(0.3, 'gini')
```

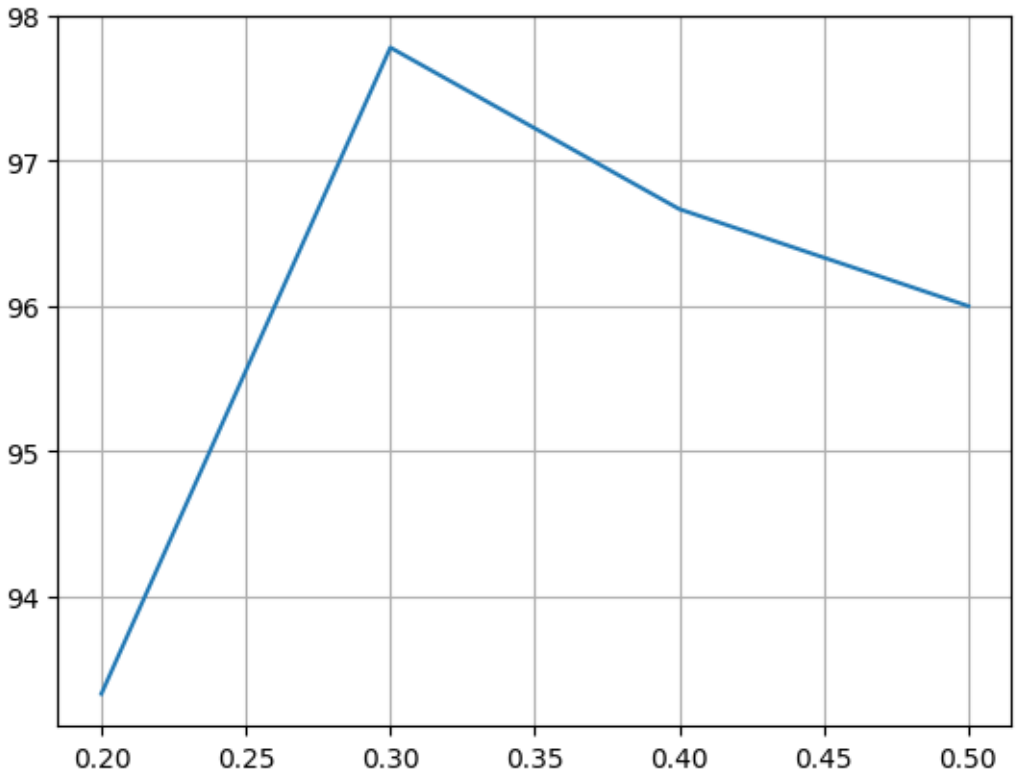
[ ]:
```
decision_tree(0.4, 'entropy')
```

[ ]:
```
decision_tree(0.4, 'gini')
```

[ ]:
```
decision_tree(0.5, 'entropy')
```

[ ]:
```
decision_tree(0.5, 'gini')
```

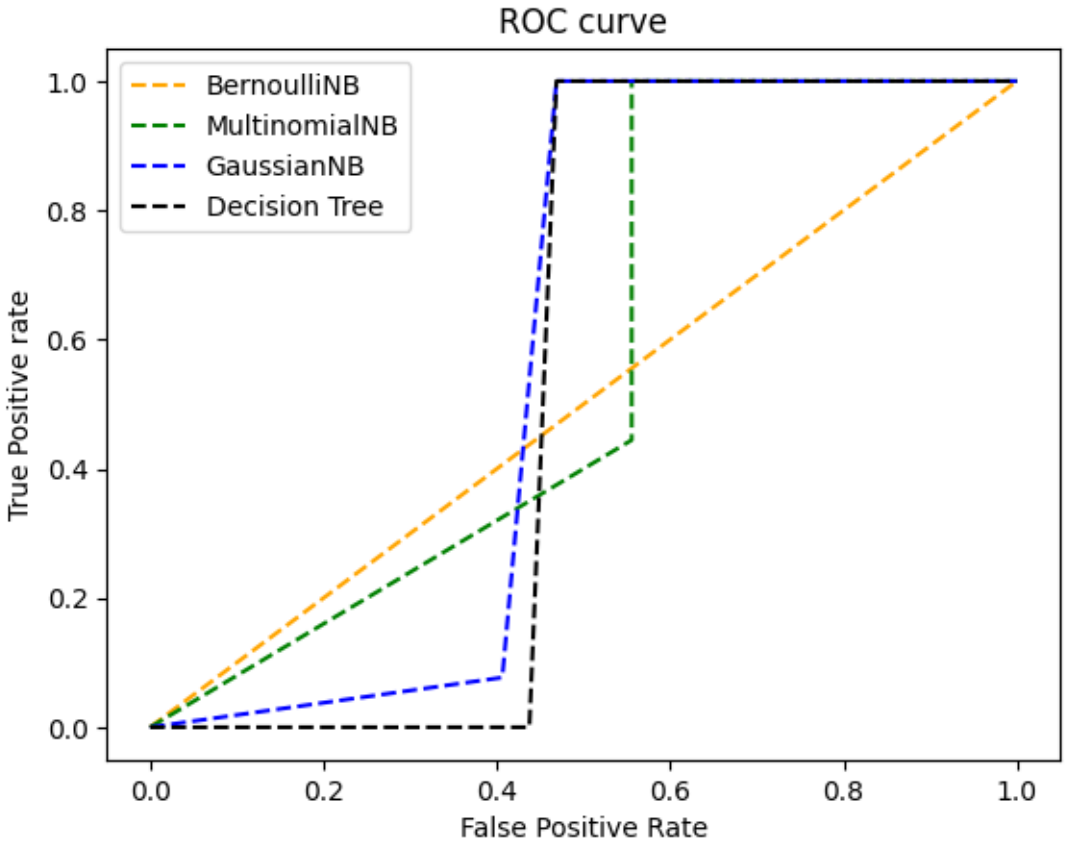
[43]:
```
keys = dict_dtr.keys()
y_points = []
for key in keys:
  y_points.append(dict_dtr[key][0]*100)
x_points = [float(key) for key in dict_dtr.keys()]


plt.plot(x_points, y_points)
plt.grid(True)
plt.show()
```




### 0.1.3 ROC curve and ROC_AUC score for all the classifier having maximum accuracy when train test split 70-30.

[38]:
```
from sklearn import metrics
def auc_roc():
    fpr1, tpr1, _1 = metrics.roc_curve(RocAucbnb['max']['y_test'],␣
 ↪RocAucbnb['max']['y_pred'], pos_label=1)
    fpr4, tpr4, _3 = metrics.roc_curve(RocAucmnb['max']['y_test'],␣
 ↪RocAucmnb['max']['y_pred'], pos_label=1)
    fpr2, tpr2, _2 = metrics.roc_curve(RocAucgnb['max']['y_test'],␣
 ↪RocAucgnb['max']['y_pred'], pos_label=1)
    fpr3, tpr3, _3 = metrics.roc_curve(RocAucdtr['max']['y_test'],␣
 ↪RocAucdtr['max']['y_pred'], pos_label=1)
    plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='BernoulliNB')
    plt.plot(fpr4, tpr4, linestyle='--',color='green', label='MultinomialNB')
    plt.plot(fpr2, tpr2, linestyle='--', color='blue', label= 'GaussianNB')
    plt.plot(fpr3, tpr3, linestyle='--', color='black', label= 'Decision Tree')
    plt.title('ROC curve')
    # x label
    plt.xlabel('False Positive Rate')
    # y label
```

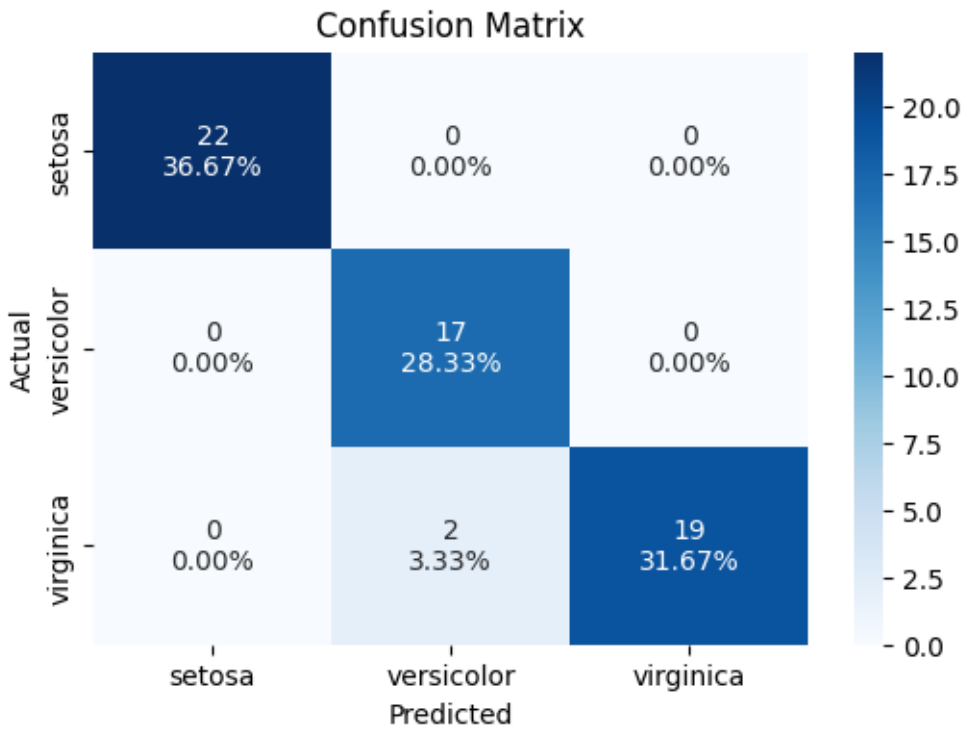
```
    plt.ylabel('True Positive rate')

    plt.legend(loc='best')
    plt.savefig('ROC',dpi=300)
    plt.show()
auc_roc()
```




**Best result for BernoulliNB Naive Bayes**

[39]:
```
maxi=0
y_test = []
y_pred = []
for key in dict_bnb:
  if maxi < 100*dict_bnb[key][0]:
    maxi = 100*dict_bnb[key][0]
    y_test = dict_bnb[key][1]
    y_pred = dict_bnb[key][2]
reports(y_test, y_pred)
```

```
Confusion Matrix :
```



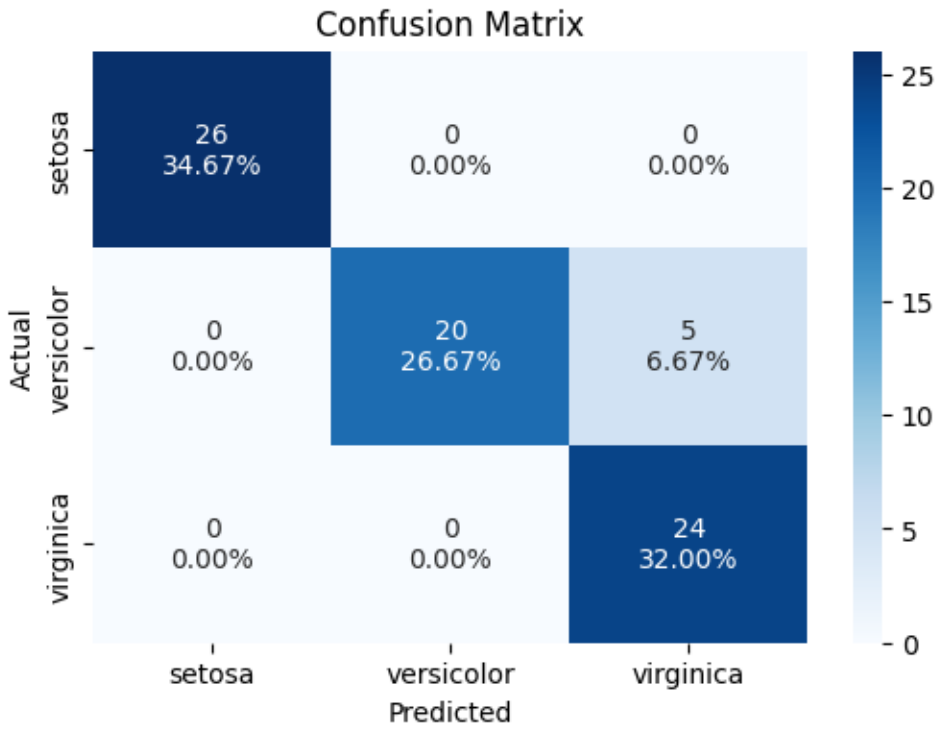

```
**********************************************
**********************************************
Classification Evaluation :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        22
           1       0.89      1.00      0.94        17
           2       1.00      0.90      0.95        21

    accuracy                           0.97        60
   macro avg       0.96      0.97      0.96        60
weighted avg       0.97      0.97      0.97        60
```

**Best result for Multinomial Naive Bayes**

[40]:
```
maxi=0
y_test = []
y_pred = []
for key in dict_mnb:
  if maxi < 100*dict_mnb[key][0]:
    maxi = 100*dict_mnb[key][0]
    y_test = dict_mnb[key][1]
    y_pred = dict_mnb[key][2]
```

```
reports(y_test, y_pred)
```

```
Confusion Matrix :
```



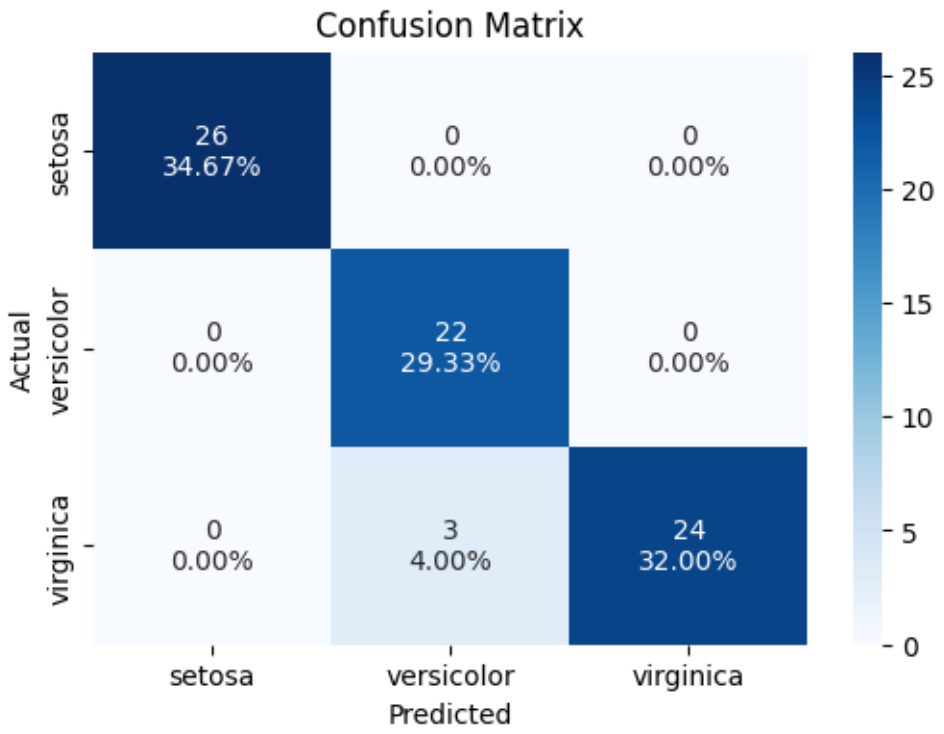

```
**********************************************
**********************************************
Classification Evaluation :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        26
           1       1.00      0.80      0.89        25
           2       0.83      1.00      0.91        24

    accuracy                           0.93        75
   macro avg       0.94      0.93      0.93        75
weighted avg       0.94      0.93      0.93        75
```

### Best result for Gaussian Naive Bayes

[41]:
```
maxi=0
y_test = []
y_pred = []
for key in dict_gnb:
```

```
  if maxi < 100*dict_gnb[key][0]:
    maxi = 100*dict_gnb[key][0]
    y_test = dict_gnb[key][1]
    y_pred = dict_gnb[key][2]
reports(y_test, y_pred)
```

Confusion Matrix :

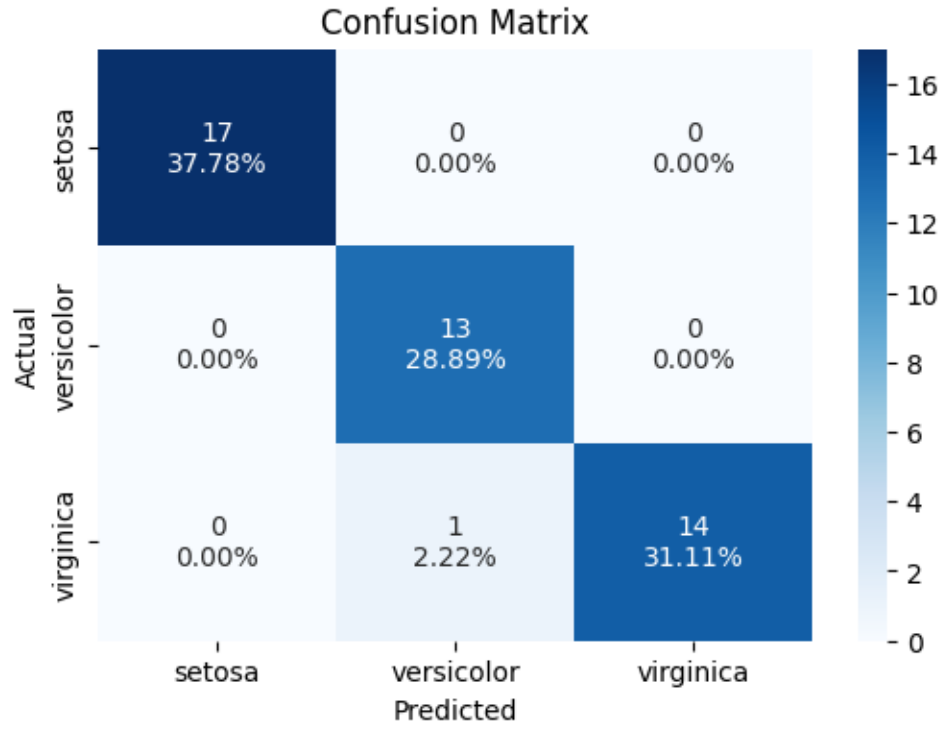



```
**********************************************
**********************************************
Classification Evaluation :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        26
           1       0.88      1.00      0.94        22
           2       1.00      0.89      0.94        27

    accuracy                           0.96        75
   macro avg       0.96      0.96      0.96        75
weighted avg       0.96      0.96      0.96        75
```

**Best result for Decision Tree Classifier**

[42]:
```
maxi=0
y_test = []
y_pred = []
for key in dict_dtr:
  if maxi < 100*dict_dtr[key][0]:
    maxi = 100*dict_dtr[key][0]
    y_test = dict_dtr[key][1]
    y_pred = dict_dtr[key][2]
reports(y_test, y_pred)
```

```
Confusion Matrix :
```




```
**********************************************
**********************************************
Classification Evaluation :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        17
           1       0.93      1.00      0.96        13
           2       1.00      0.93      0.97        15

    accuracy                           0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

# sayadat-037-breast-cancer-dataset

August 22, 2023

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

### 0.0.1 Loading dataset from github

[ ]:
```
from sklearn.datasets import load_breast_cancer
breast_cancer = load_breast_cancer()
df = pd.DataFrame(breast_cancer.data, columns = breast_cancer.feature_names)
column_names = list(df.columns.values)
df.head()
```

[ ]:
```
X = df.iloc[:][:]
y = breast_cancer["target"]
dict_bnb = {}
dict_mnb = {}
dict_gnb = {}
dict_dtr = {}
RocAucbnb = {}
RocAucmnb = {}
RocAucgnb = {}
RocAucdtr = {}
```

[ ]:
```
def plot(y_test, y_pred):
  from sklearn.metrics import confusion_matrix
  import seaborn as sns

  print("Confusion Matrix : ")
  cf_matrix = confusion_matrix(y_test, y_pred)
  group_names = ['True Pos','False Pos','False Neg','True neg']
  group_counts = ["{0:0.0f}".format(value) for value in
                  cf_matrix.flatten()]
  group_percentages = ["{0:.2%}".format(value) for value in
                       cf_matrix.flatten()/np.sum(cf_matrix)]
  labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
            zip(group_names,group_counts,group_percentages)]
  labels = np.asarray(labels).reshape(2,2)
```

```
  plt.figure(figsize=(6, 4))
  sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues',␣
↪xticklabels=['Benign', 'Malignant'], yticklabels=['Benign', 'Malignant'])
  plt.xlabel('Predicted')
  plt.ylabel('Actual')
  plt.title('Confusion Matrix')
  plt.show()
  print("**********************************************")
```

[ ]:
```
def reports(y_test, y_pred):
  from sklearn.metrics import classification_report
  plot(y_test, y_pred)
  print("**********************************************")
  print("Classification Evaluation : ")
  print(classification_report(y_test, y_pred, zero_division = 0))
```

### 0.0.2 Classification using BernoulliNB Naive Bayes

[ ]:
```
def FBouBernoulli(split, alpha_value = 1.0, binarize_value = 0.0,␣
↪fit_prior_value = False):
  from sklearn.naive_bayes import BernoulliNB
  from sklearn.metrics import accuracy_score
  from sklearn.model_selection import train_test_split
  from sklearn.preprocessing import StandardScaler
  #scaler = StandardScaler()
  X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = split,␣
↪random_state=44)
  #scaler.fit_transform(X_train)
  #scaler.transform(X_test)
  classifier = BernoulliNB(alpha = alpha_value, binarize = binarize_value,␣
↪fit_prior = fit_prior_value)
  classifier.fit(X_train, y_train)
  y_pred = classifier.predict(X_test)
  print("Train-test split: " + str(split))
  print("value: alpha: "+str(alpha_value) + " binarize: " + str(binarize_value)␣
↪+ " fit_prior: " +str(fit_prior_value))
  print("**********************************************")
  accuracy = accuracy_score(y_test, y_pred)
  if str(split) in dict_bnb:
    if dict_bnb[str(split)][0] < accuracy:
      dict_bnb[str(split)] = [accuracy, y_test, y_pred]
    if str(split) == '0.3' and accuracy > dict_bnb[str(split)][0]:
      RocAucbnb['max'] = {'y_test': y_test, 'y_pred': y_pred}
  else:
    dict_bnb[str(split)] = [accuracy, y_test, y_pred]
    if str(split) == '0.3':
```

```
      RocAucbnb['max'] = {'y_test': y_test, 'y_pred': y_pred}
  reports(y_test, y_pred)
```

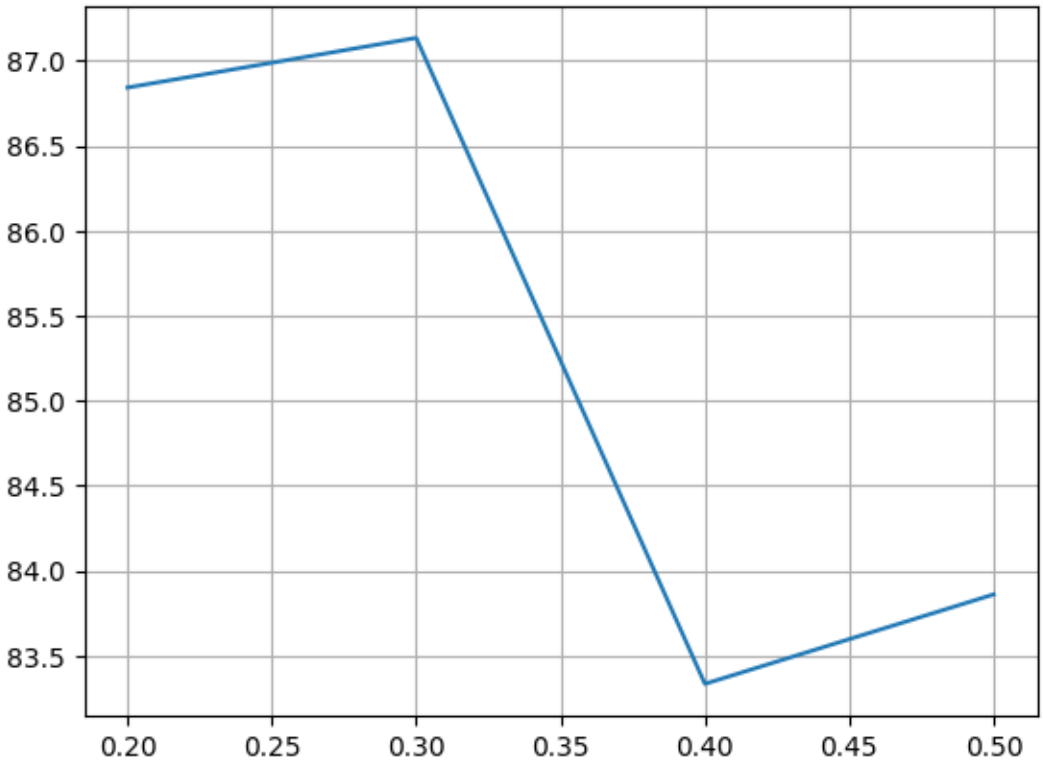
[ ]:
```
## Train-Test split 0.2
FBouBernoulli(0.2)
FBouBernoulli(0.2, 2.8)
FBouBernoulli(0.2, 2.0, 2.8)
FBouBernoulli(0.2, 3.0, 3.3, True)
```

[ ]:
```
## Train-Test split 0.3
FBouBernoulli(0.3)
FBouBernoulli(0.3, 1.0, 2.8)
FBouBernoulli(0.3, 1.0, 2.8, True)
```

[ ]:
```
## Train-Test split 0.4
FBouBernoulli(0.4)
FBouBernoulli(0.4, 1.0, 2.8)
FBouBernoulli(0.4, 1.0, 2.8, True)
```

[ ]:
```
## Train-Test split 0.5
FBouBernoulli(0.5)
FBouBernoulli(0.5, 1.0, 2.9)
FBouBernoulli(0.5, 1.0, 2.9, True)
```

[48]:
```
keys = dict_bnb.keys()
y_points = []
for key in keys:
  y_points.append(dict_bnb[key][0]*100)
x_points = [float(key) for key in dict_bnb.keys()]

plt.plot(x_points, y_points)
plt.grid(True)
plt.show()
```




### 0.1 Classification using Multinomial Naive Bayes

[ ]:
```
def FMultinomial(split, alpha_value = 1.0):
  from sklearn.naive_bayes import MultinomialNB
  from sklearn.metrics import accuracy_score
  from sklearn.model_selection import train_test_split
  X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = split)
  classifier = MultinomialNB(alpha = alpha_value)
  classifier.fit(X_train, y_train)
  y_pred = classifier.predict(X_test)
  print("Train-test split: " + str(split))
  print("value: alpha: "+str(alpha_value))
  print("**********************************************")
  accuracy = accuracy_score(y_test, y_pred)
  if str(split) in dict_mnb:
    if dict_mnb[str(split)][0] < accuracy:
      dict_mnb[str(split)] = [accuracy, y_test, y_pred]
    if str(split) == '0.3' and accuracy > dict_mnb[str(split)][0]:
      RocAucmnb['max'] = {'y_test': y_test, 'y_pred': y_pred}
  else:
    dict_mnb[str(split)] = [accuracy, y_test, y_pred]
    if str(split) == '0.3':
```

```
      RocAucmnb['max'] = {'y_test': y_test, 'y_pred': y_pred}
  reports(y_test, y_pred)
```

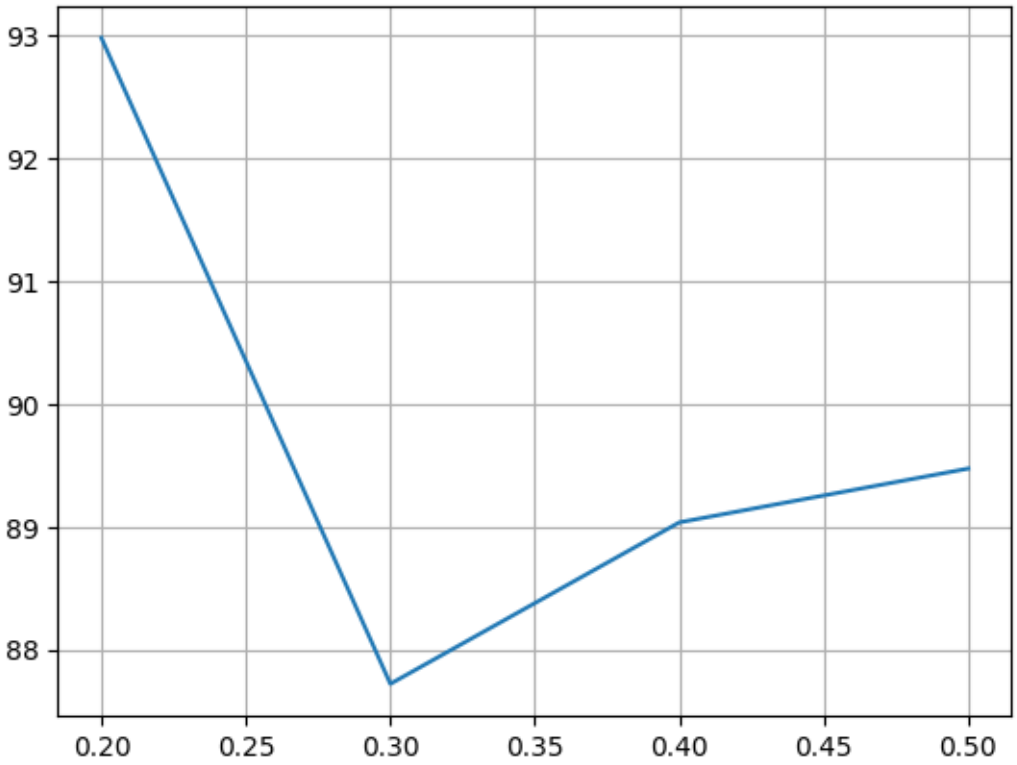
[ ]:
```
## Train-Test split 0.2
FMultinomial(0.2)
FMultinomial(0.2, 1.8)
```

[ ]:
```
## Train-Test split 0.3
FMultinomial(0.3)
FMultinomial(0.3, 2.5)
```

[ ]:
```
## Train-Test split 0.4
FMultinomial(0.4)
FMultinomial(0.4, 2.1)
```

[ ]:
```
## Train-Test split 0.5
FMultinomial(0.5)
FMultinomial(0.5, 1.8)
```

[47]:
```
keys = dict_mnb.keys()
y_points = []
for key in keys:
  y_points.append(dict_mnb[key][0]*100)
x_points = [float(key) for key in dict_mnb.keys()]

plt.plot(x_points, y_points)
plt.grid(True)
plt.show()
```




### 0.1.1 Classification using Guassian Naive Bayes

[ ]:
```
def FGaussian(split):
  from sklearn.naive_bayes import GaussianNB
  from sklearn.model_selection import train_test_split
  from sklearn.metrics import accuracy_score
  from sklearn.preprocessing import StandardScaler
  scaler = StandardScaler()
  X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = split,␣
↪random_state=44)
  scaler.fit_transform(X_train)
  scaler.transform(X_test)
  classifier = GaussianNB()
  classifier.fit(X_train, y_train)
  y_pred = classifier.predict(X_test)
  print("Train-test split: " + str(split))
  print("**********************************************")
  reports(y_test, y_pred)
  accuracy = accuracy_score(y_test, y_pred)
  if str(split) in dict_gnb:
    if dict_gnb[str(split)][0] < accuracy:
      dict_gnb[str(split)] = [accuracy, y_test, y_pred]
```

```
    if str(split) == '0.3' and accuracy > dict_gnb[str(split)][0]:
      RocAucgnb['max'] = {'y_test': y_test, 'y_pred': y_pred}
  else:
    dict_gnb[str(split)] = [accuracy, y_test, y_pred]
    if str(split) == '0.3':
      RocAucgnb['max'] = {'y_test': y_test, 'y_pred': y_pred}
  reports(y_test, y_pred)
```

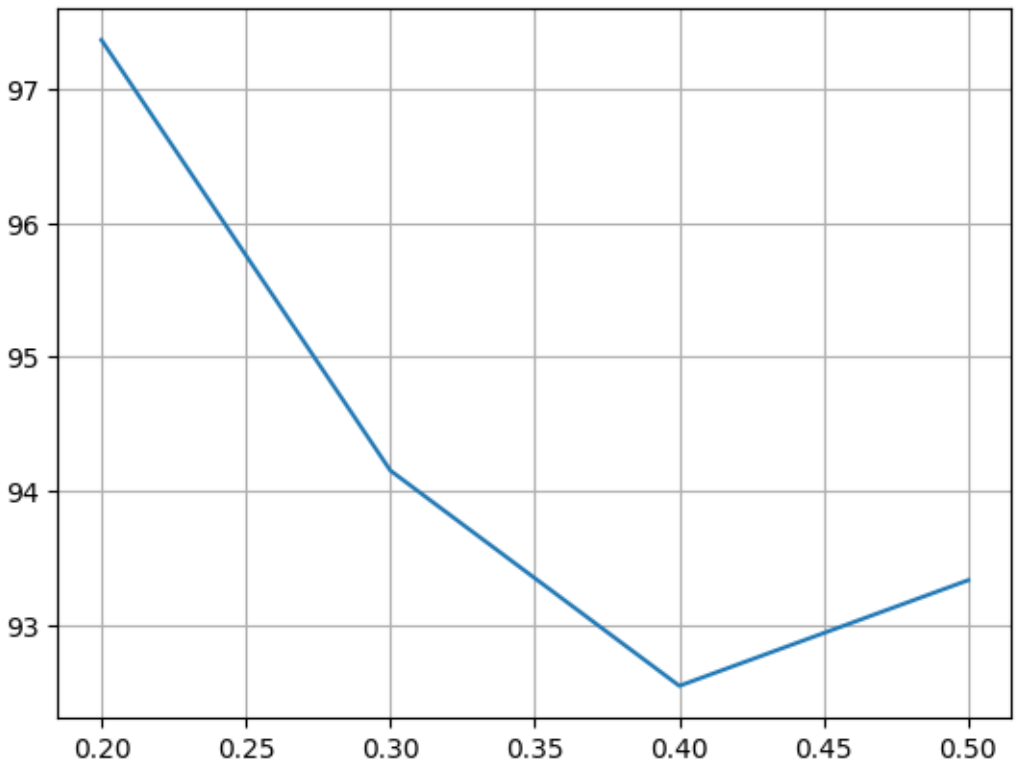
[ ]:
```
## Train-Test split 0.2
FGaussian(0.2)
```

[ ]:
```
## Train-Test split 0.3
FGaussian(0.3)
```

[ ]:
```
## Train-Test split 0.4
FGaussian(0.4)
```

[ ]:
```
## Train-Test split 0.5
FGaussian(0.5)
```

[46]:
```
keys = dict_gnb.keys()
y_points = []
for key in keys:
  y_points.append(dict_gnb[key][0]*100)
x_points = [float(key) for key in dict_gnb.keys()]

plt.plot(x_points, y_points)
plt.grid(True)
plt.show()
```




### 0.1.2 Classification using Decision Tree

[ ]:
```
def decision_tree(split, criterion_value):
  from sklearn.model_selection import train_test_split
  from sklearn.tree import DecisionTreeClassifier
  from sklearn import tree
  from sklearn.metrics import accuracy_score
  from sklearn.preprocessing import StandardScaler
  scaler = StandardScaler()
  X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = split,␣
 ↪random_state=44)
  scaler.fit_transform(X_train)
  scaler.transform(X_test)

  classifier = DecisionTreeClassifier(criterion = criterion_value)
  classifier.fit(X_train, y_train)
  y_pred = classifier.predict(X_test)
  print("Train-test split: " + str(split))
  print("Value: Entropy: " + criterion_value)
  print("*********************************************")
  reports(y_test, y_pred)
  accuracy = accuracy_score(y_test, y_pred)
```

```
  if str(split) in dict_dtr:
    if dict_dtr[str(split)][0] < accuracy:
      dict_dtr[str(split)] = [accuracy, y_test, y_pred]
    if str(split) == '0.3' and accuracy > dict_dtr[str(split)][0]:
      RocAucdtr['max'] = {'y_test': y_test, 'y_pred': y_pred}
  else:
    dict_dtr[str(split)] = [accuracy, y_test, y_pred]
    if str(split) == '0.3':
      RocAucdtr['max'] = {'y_test': y_test, 'y_pred': y_pred}

  reports(y_test, y_pred)
  fig = plt.figure(figsize=(12,8))
  _ = tree.plot_tree(classifier,
                     feature_names=column_names,
                     class_names=['outcome1', 'outcome2'],
                     filled=True)
```

[ ]:
```
decision_tree(0.2, 'entropy')
```

[ ]:
```
decision_tree(0.2, 'gini')
```

[ ]:
```
decision_tree(0.3, 'entropy')
```

[ ]:
```
decision_tree(0.3, 'gini')
```

[ ]:
```
decision_tree(0.4, 'entropy')
```

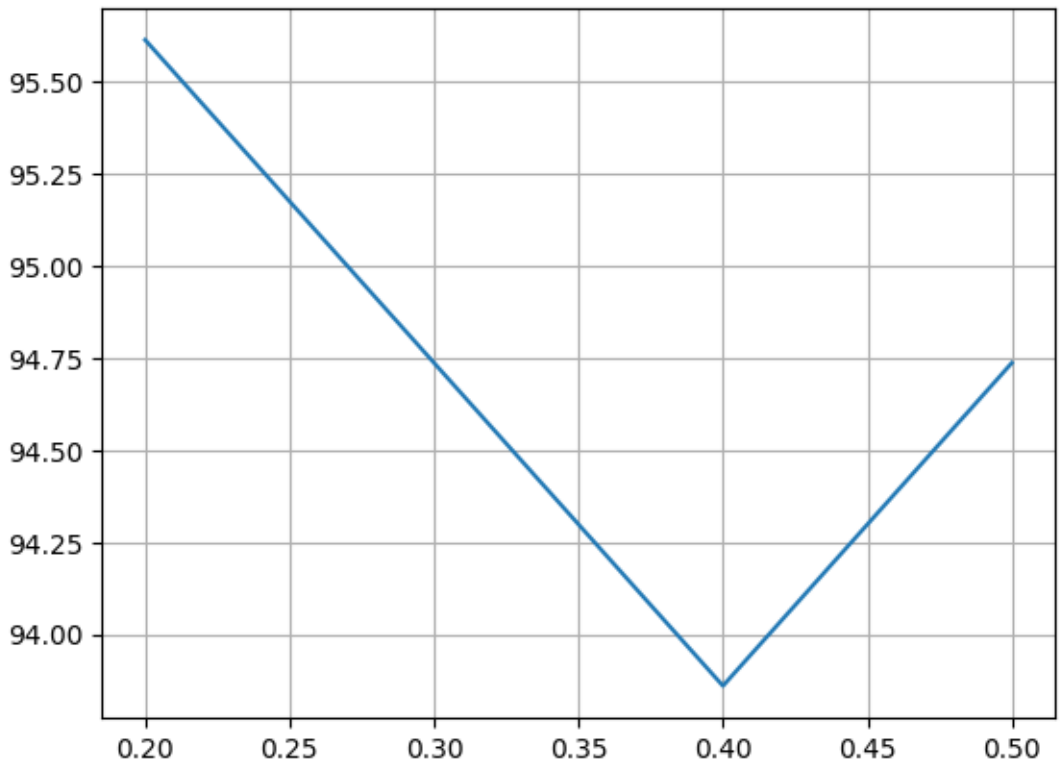
[ ]:
```
decision_tree(0.4, 'gini')
```

[ ]:
```
decision_tree(0.5, 'entropy')
```

[ ]:
```
decision_tree(0.5, 'gini')
```

[45]:
```
keys = dict_dtr.keys()
y_points = []
for key in keys:
  y_points.append(dict_dtr[key][0]*100)
x_points = [float(key) for key in dict_dtr.keys()]
print(y_points, x_points)

plt.plot(x_points, y_points)
plt.grid(True)
plt.show()
```

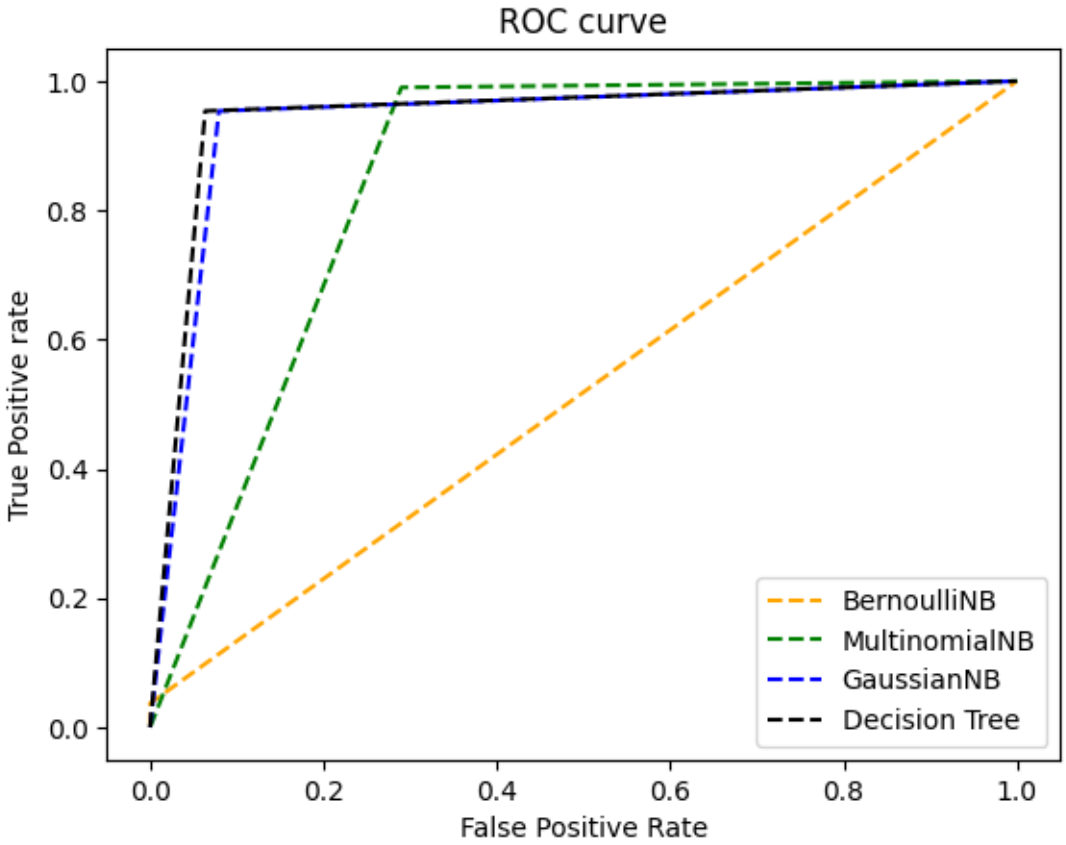
```
[95.6140350877193, 94.73684210526315, 93.85964912280701, 94.73684210526315]
[0.2, 0.3, 0.4, 0.5]
```




### 0.1.3 ROC curve and ROC_AUC score for all the classifier having maximum accuracy when train test split 70-30.

[40]:
```
from sklearn import metrics
def auc_roc():
    fpr1, tpr1, _1 = metrics.roc_curve(RocAucbnb['max']['y_test'],␣
 ↪RocAucbnb['max']['y_pred'], pos_label=1)
    fpr4, tpr4, _3 = metrics.roc_curve(RocAucmnb['max']['y_test'],␣
 ↪RocAucmnb['max']['y_pred'], pos_label=1)
    fpr2, tpr2, _2 = metrics.roc_curve(RocAucgnb['max']['y_test'],␣
 ↪RocAucgnb['max']['y_pred'], pos_label=1)
    fpr3, tpr3, _3 = metrics.roc_curve(RocAucdtr['max']['y_test'],␣
 ↪RocAucdtr['max']['y_pred'], pos_label=1)
    plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='BernoulliNB')
    plt.plot(fpr4, tpr4, linestyle='--',color='green', label='MultinomialNB')
    plt.plot(fpr2, tpr2, linestyle='--', color='blue', label= 'GaussianNB')
    plt.plot(fpr3, tpr3, linestyle='--', color='black', label= 'Decision Tree')
    plt.title('ROC curve')
    # x label
    plt.xlabel('False Positive Rate')
    # y label
```

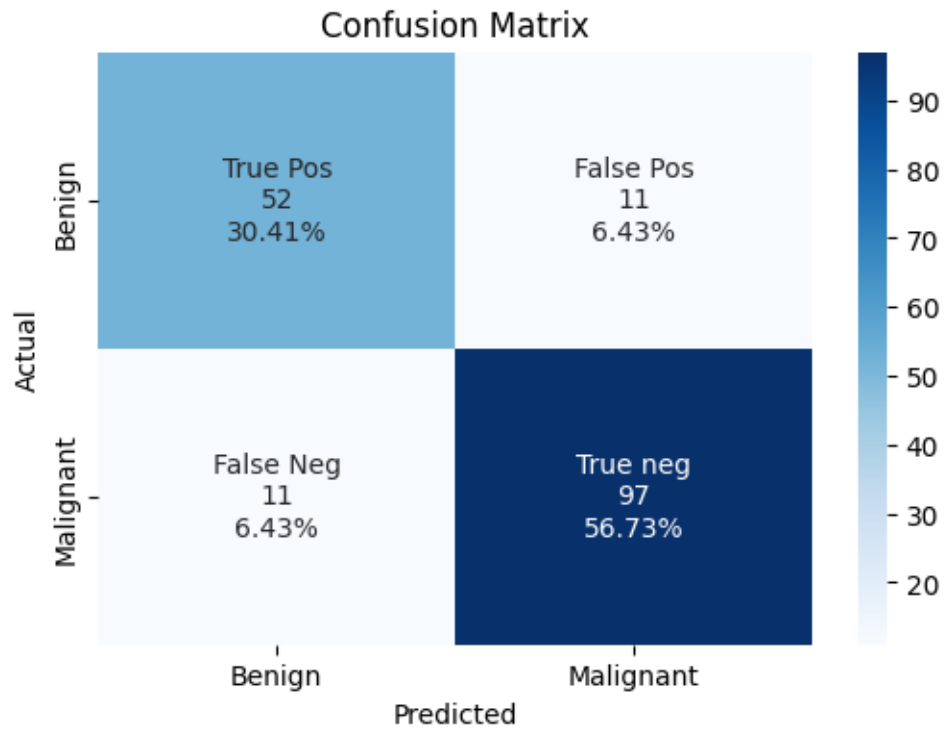
```
    plt.ylabel('True Positive rate')

    plt.legend(loc='best')
    plt.savefig('ROC',dpi=300)
    plt.show()
auc_roc()
```




**Best result for BernoulliNB Naive Bayes**

[41]:
```
maxi=0
y_test = []
y_pred = []
for key in dict_bnb:
  if maxi < 100*dict_bnb[key][0]:
    maxi = 100*dict_bnb[key][0]
    y_test = dict_bnb[key][1]
    y_pred = dict_bnb[key][2]
reports(y_test, y_pred)
```

```
Confusion Matrix :
```

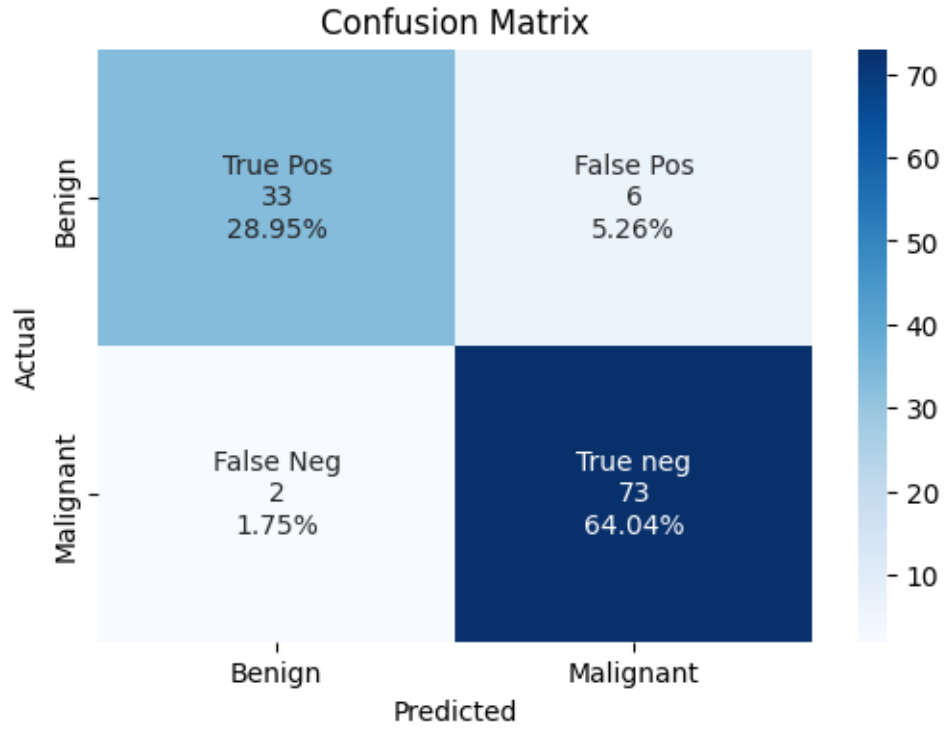



```
**********************************************
**********************************************
Classification Evaluation :
              precision    recall  f1-score   support

           0       0.83      0.83      0.83        63
           1       0.90      0.90      0.90       108

    accuracy                           0.87       171
   macro avg       0.86      0.86      0.86       171
weighted avg       0.87      0.87      0.87       171
```

**Best result for Multinomial Naive Bayes**

[42]:
```
maxi=0
y_test = []
y_pred = []
for key in dict_mnb:
  if maxi < 100*dict_mnb[key][0]:
    maxi = 100*dict_mnb[key][0]
    y_test = dict_mnb[key][1]
    y_pred = dict_mnb[key][2]
reports(y_test, y_pred)
```

```
Confusion Matrix :
```



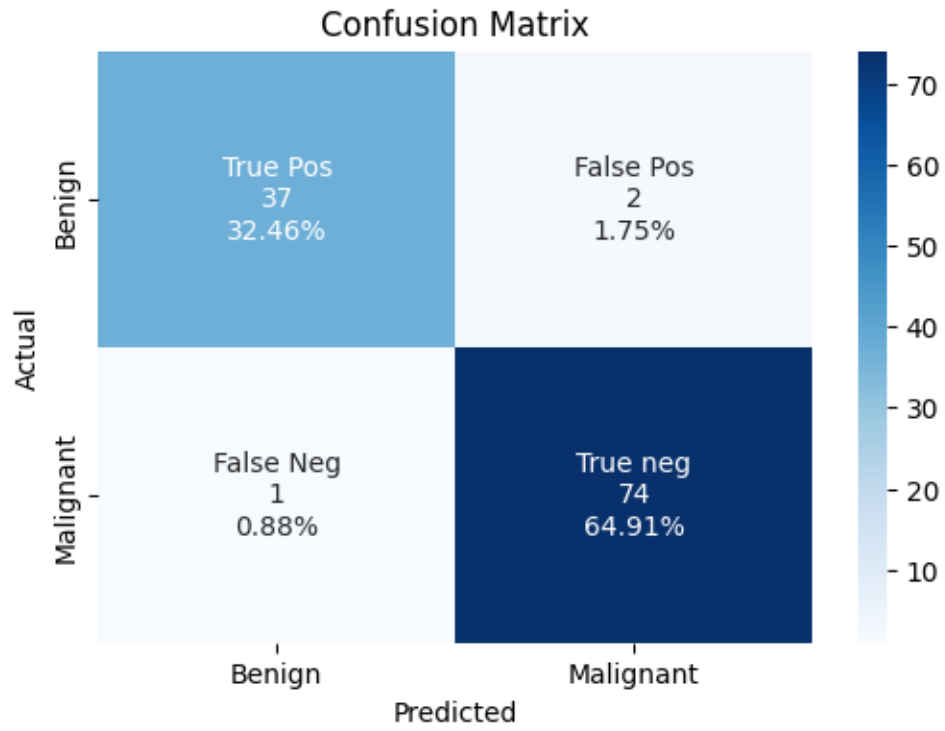

```
**********************************************
**********************************************
Classification Evaluation :
              precision    recall  f1-score   support

           0       0.94      0.85      0.89        39
           1       0.92      0.97      0.95        75

    accuracy                           0.93       114
   macro avg       0.93      0.91      0.92       114
weighted avg       0.93      0.93      0.93       114
```

### Best result for Gaussian Naive Bayes

[43]:
```
maxi=0
y_test = []
y_pred = []
for key in dict_dtr:
  if maxi < 100*dict_gnb[key][0]:
    maxi = 100*dict_gnb[key][0]
    y_test = dict_gnb[key][1]
```

```
    y_pred = dict_gnb[key][2]
reports(y_test, y_pred)
```

```
Confusion Matrix :
```



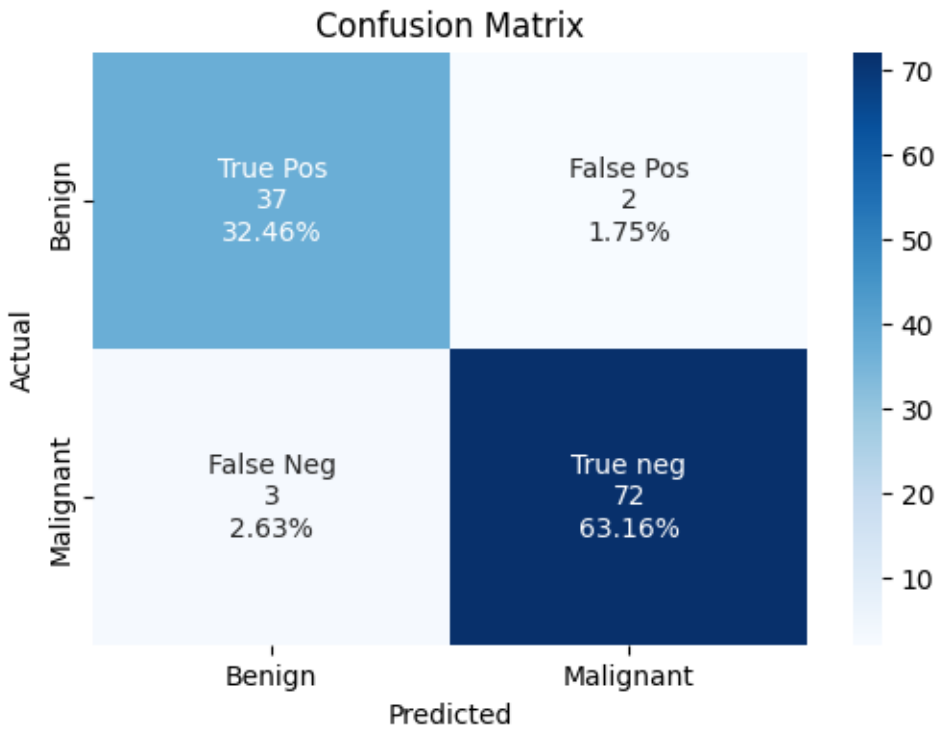

```
**********************************************
**********************************************
Classification Evaluation :
              precision    recall  f1-score   support

           0       0.97      0.95      0.96        39
           1       0.97      0.99      0.98        75

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```

### Best result for Decision Tree Classifier

[44]:
```
maxi=0
y_test = []
y_pred = []
for key in dict_dtr:
```

```
  if maxi < 100*dict_dtr[key][0]:
    maxi = 100*dict_dtr[key][0]
    y_test = dict_dtr[key][1]
    y_pred = dict_dtr[key][2]
reports(y_test, y_pred)
```

Confusion Matrix :




```
**********************************************
**********************************************
Classification Evaluation :
              precision    recall  f1-score   support

           0       0.93      0.95      0.94        39
           1       0.97      0.96      0.97        75

    accuracy                           0.96       114
   macro avg       0.95      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

# kilus-sayadat-037-diabetes-dataset

August 22, 2023

[ ]:

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

### 0.0.1 Loading dataset from github

[ ]:
```
url = 'https://raw.githubusercontent.com/Aqeel-0/test/build/diabetes.csv'
df = pd.read_csv(url)
df.head()
column_names = list(df.columns.values)
```

[ ]:
```
X = df.iloc[:,:-1]
y = df["Outcome"]
dict_bnb = {}
dict_mnb = {}
dict_gnb = {}
dict_dtr = {}
RocAucbnb = {}
RocAucmnb = {}
RocAucgnb = {}
RocAucdtr = {}
y.info(), X.info()
```

[ ]:
```
def plot(y_test, y_pred):
  from sklearn.metrics import confusion_matrix
  import seaborn as sns

  print("Confusion Matrix : ")
  cf_matrix = confusion_matrix(y_test, y_pred)
  group_names = ['True Pos','False Pos','False Neg','True neg']
  group_counts = ["{0:0.0f}".format(value) for value in
                  cf_matrix.flatten()]
  group_percentages = ["{0:.2%}".format(value) for value in
                       cf_matrix.flatten()/np.sum(cf_matrix)]
  labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
```

```
            zip(group_names,group_counts,group_percentages)]
  labels = np.asarray(labels).reshape(2,2)
  plt.figure(figsize=(6, 4))
  sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues',␣
↪xticklabels=['Benign', 'Malignant'], yticklabels=['Benign', 'Malignant'])
  plt.xlabel('Predicted')
  plt.ylabel('Actual')
  plt.title('Confusion Matrix')
  plt.show()
  print("*********************************************")
```

[ ]:
```
def reports(y_test, y_pred):
  from sklearn.metrics import classification_report
  plot(y_test, y_pred)
  print("*********************************************")
  print("Classification Evaluation : ")
  print(classification_report(y_test, y_pred, zero_division = 0))
```

### 0.0.2 Classification using BernoulliNB Naive Bayes

[ ]:
```
def FBouBernoulli(split, alpha_value = 1.0, binarize_value = 0.0,␣
↪fit_prior_value = False):
  from sklearn.naive_bayes import BernoulliNB
  from sklearn.metrics import accuracy_score
  from sklearn.model_selection import train_test_split
  from sklearn.preprocessing import StandardScaler
  #scaler = StandardScaler()
  X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = split,␣
↪random_state=44)
  #scaler.fit_transform(X_train)
  #scaler.transform(X_test)
  classifier = BernoulliNB(alpha = alpha_value, binarize = binarize_value,␣
↪fit_prior = fit_prior_value)
  classifier.fit(X_train, y_train)
  y_pred = classifier.predict(X_test)
  print("Train-test split: " + str(split))
  print("value: alpha: "+str(alpha_value) + " binarize: " + str(binarize_value)␣
↪+ " fit_prior: " +str(fit_prior_value))
  print("*********************************************")
  accuracy = accuracy_score(y_test, y_pred)
  if str(split) in dict_bnb:
    if dict_bnb[str(split)][0] < accuracy:
      dict_bnb[str(split)] = [accuracy, y_test, y_pred]
    if str(split) == '0.3' and accuracy > dict_bnb[str(split)][0]:
      RocAucbnb['max'] = {'y_test': y_test, 'y_pred': y_pred}
  else:
```

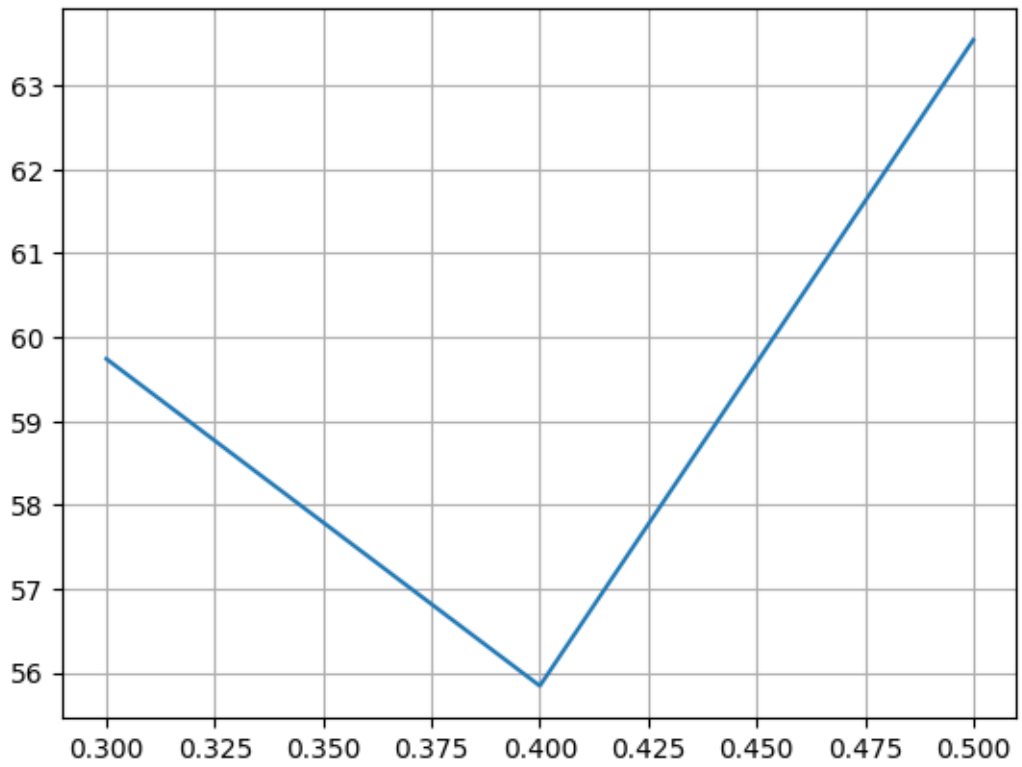
```
    dict_bnb[str(split)] = [accuracy, y_test, y_pred]
    if str(split) == '0.3':
      RocAucbnb['max'] = {'y_test': y_test, 'y_pred': y_pred}
  reports(y_test, y_pred)
```

[ ]:
```
## Train-Test split 0.3
FBouBernoulli(0.3)
FBouBernoulli(0.3, 1.0, 1.5)
FBouBernoulli(0.3, 1.0, 1.5, True)
```

[ ]:
```
## Train-Test split 0.4
FBouBernoulli(0.4)
```

[ ]:
```
## Train-Test split 0.5
FBouBernoulli(0.5)
FBouBernoulli(0.5, 1.0, 7.9)
FBouBernoulli(0.5, 1.0, 7.9, True)
```

[87]:
```
keys = dict_bnb.keys()
y_points = []
for key in keys:
  y_points.append(dict_bnb[key][0]*100)
x_points = [float(key) for key in dict_bnb.keys()]

plt.plot(x_points, y_points)
plt.grid(True)
plt.show()
```




### 0.1 Classification using Multinomial Naive Bayes

[ ]:
```
def FMultinomial(split, alpha_value = 1.0):
  from sklearn.naive_bayes import MultinomialNB
  from sklearn.metrics import accuracy_score
  from sklearn.model_selection import train_test_split
  X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = split)
  classifier = MultinomialNB(alpha = alpha_value)
  classifier.fit(X_train, y_train)
  y_pred = classifier.predict(X_test)
  print("Train-test split: " + str(split))
  print("value: alpha: "+str(alpha_value))
  print("**********************************************")
  accuracy = accuracy_score(y_test, y_pred)
  if str(split) in dict_mnb:
    if dict_mnb[str(split)][0] < accuracy:
      dict_mnb[str(split)] = [accuracy, y_test, y_pred]
    if str(split) == '0.3' and accuracy > dict_mnb[str(split)][0]:
      RocAucmnb['max'] = {'y_test': y_test, 'y_pred': y_pred}
  else:
    dict_mnb[str(split)] = [accuracy, y_test, y_pred]
    if str(split) == '0.3':
```

```
      RocAucmnb['max'] = {'y_test': y_test, 'y_pred': y_pred}
  reports(y_test, y_pred)
```

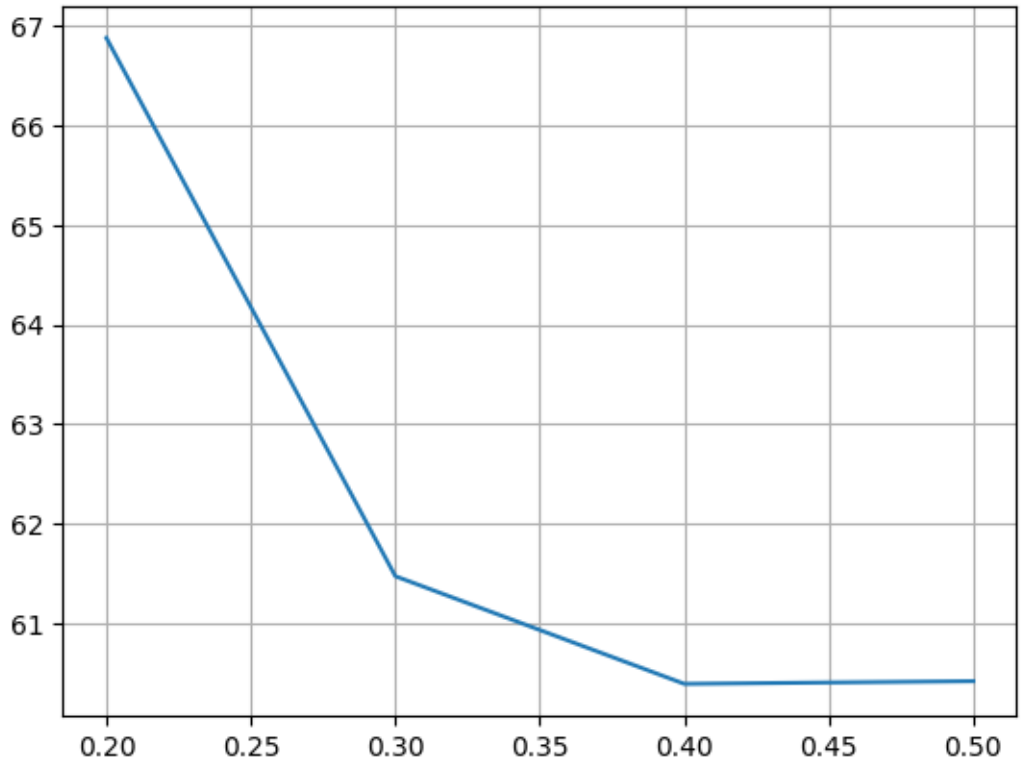
[ ]:
```
## Train-Test split 0.2
FMultinomial(0.2)
FMultinomial(0.2, 1.8)
```

[ ]:
```
## Train-Test split 0.3
FMultinomial(0.3)
FMultinomial(0.3, 2.9)
```

[ ]:
```
## Train-Test split 0.4
FMultinomial(0.4)
FMultinomial(0.4, 1.1)
```

[ ]:
```
## Train-Test split 0.5
FMultinomial(0.5)
FMultinomial(0.5, 4.8)
```

[86]:
```
keys = dict_mnb.keys()
y_points = []
for key in keys:
  y_points.append(dict_mnb[key][0]*100)
x_points = [float(key) for key in dict_mnb.keys()]

plt.plot(x_points, y_points)
plt.grid(True)
plt.show()
```




### 0.1.1 Classification using Guassian Naive Bayes

[ ]:
```
def FGaussian(split):
  from sklearn.naive_bayes import GaussianNB
  from sklearn.model_selection import train_test_split
  from sklearn.metrics import accuracy_score
  from sklearn.preprocessing import StandardScaler
  scaler = StandardScaler()
  X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = split,␣
 ↪random_state=44)
  scaler.fit_transform(X_train)
  scaler.transform(X_test)
  classifier = GaussianNB()
  classifier.fit(X_train, y_train)
  y_pred = classifier.predict(X_test)
  print("Train-test split: " + str(split))
  print("**********************************************")
  reports(y_test, y_pred)
  accuracy = accuracy_score(y_test, y_pred)
  if str(split) in dict_gnb:
    if dict_gnb[str(split)][0] < accuracy:
      dict_gnb[str(split)] = [accuracy, y_test, y_pred]
```

```
    if str(split) == '0.3' and accuracy > dict_gnb[str(split)][0]:
      RocAucgnb['max'] = {'y_test': y_test, 'y_pred': y_pred}
  else:
    dict_gnb[str(split)] = [accuracy, y_test, y_pred]
    if str(split) == '0.3':
      RocAucgnb['max'] = {'y_test': y_test, 'y_pred': y_pred}
```

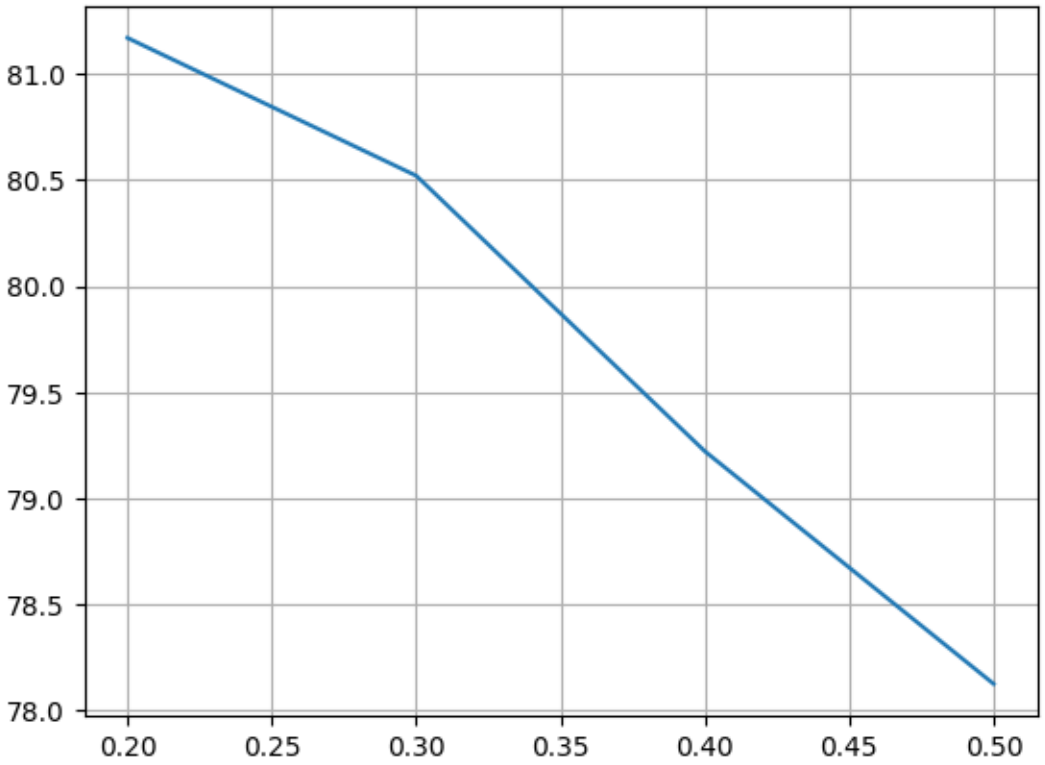
[ ]:
```
## Train-Test split 0.2
FGaussian(0.2)
```

[ ]:
```
## Train-Test split 0.3
FGaussian(0.3)
```

[ ]:
```
## Train-Test split 0.4
FGaussian(0.4)
```

[ ]:
```
## Train-Test split 0.5
FGaussian(0.5)
```

[85]:
```
keys = dict_gnb.keys()
y_points = []
for key in keys:
  y_points.append(dict_gnb[key][0]*100)
x_points = [float(key) for key in dict_gnb.keys()]

plt.plot(x_points, y_points)
plt.grid(True)
plt.show()
```




### 0.1.2 Classification using Decision Tree

[ ]:
```
def decision_tree(split, criterion_value):
  from sklearn.model_selection import train_test_split
  from sklearn.tree import DecisionTreeClassifier
  from sklearn import tree
  from sklearn.metrics import accuracy_score
  from sklearn.preprocessing import StandardScaler
  scaler = StandardScaler()
  X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = split,␣
 ↪random_state=44)
  scaler.fit_transform(X_train)
  scaler.transform(X_test)

  classifier = DecisionTreeClassifier(criterion = criterion_value)
  classifier.fit(X_train, y_train)
  y_pred = classifier.predict(X_test)
  print("Train-test split: " + str(split))
  print("Value: Entropy: " + criterion_value)
  print("*********************************************")
  reports(y_test, y_pred)
  accuracy = accuracy_score(y_test, y_pred)
```

```
  if str(split) in dict_dtr:
    if dict_dtr[str(split)][0] < accuracy:
      dict_dtr[str(split)] = [accuracy, y_test, y_pred]
    if str(split) == '0.3' and accuracy > dict_dtr[str(split)][0]:
      RocAucdtr['max'] = {'y_test': y_test, 'y_pred': y_pred}
  else:
    dict_dtr[str(split)] = [accuracy, y_test, y_pred]
    if str(split) == '0.3':
      RocAucdtr['max'] = {'y_test': y_test, 'y_pred': y_pred}

  reports(y_test, y_pred)
  fig = plt.figure(figsize=(12,8))
  _ = tree.plot_tree(classifier,
                     feature_names=column_names,
                     class_names=['outcome1', 'outcome2'],
                     filled=True)
```

[ ]:
```
decision_tree(0.2, 'entropy')
```

[ ]:
```
decision_tree(0.2, 'gini')
```

[ ]:
```
decision_tree(0.3, 'entropy')
```

[ ]:
```
decision_tree(0.3, 'gini')
```

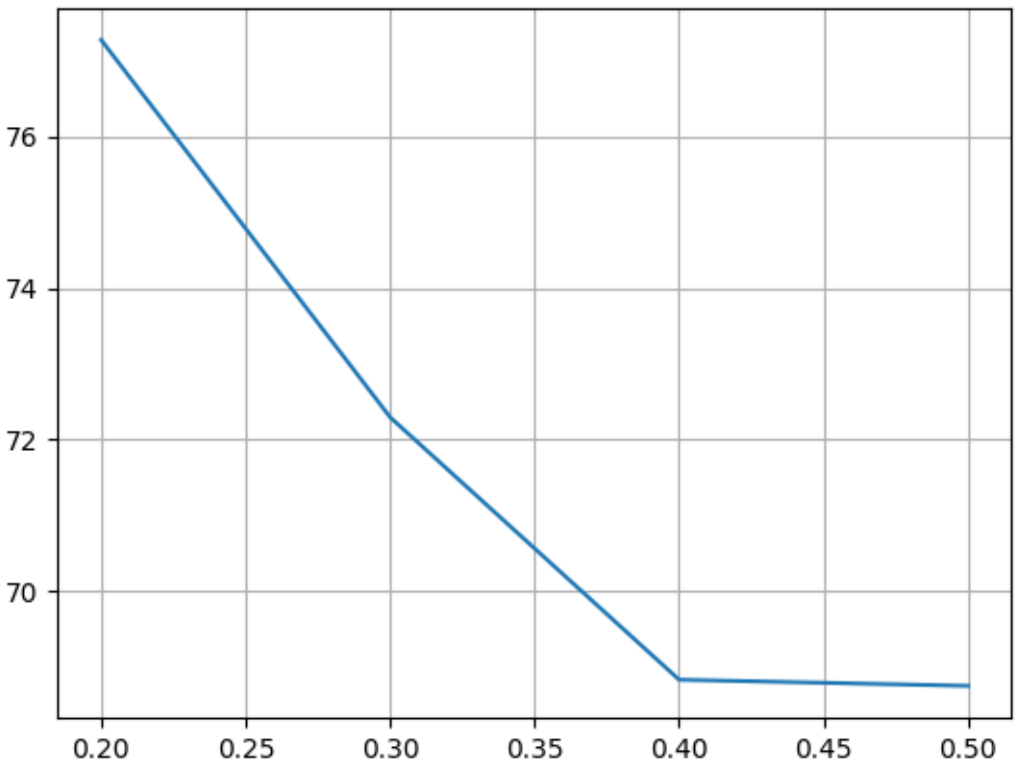
[ ]:
```
decision_tree(0.4, 'entropy')
```

[ ]:
```
decision_tree(0.4, 'gini')
```

[ ]:
```
decision_tree(0.5, 'entropy')
```

[ ]:
```
decision_tree(0.5, 'gini')
```

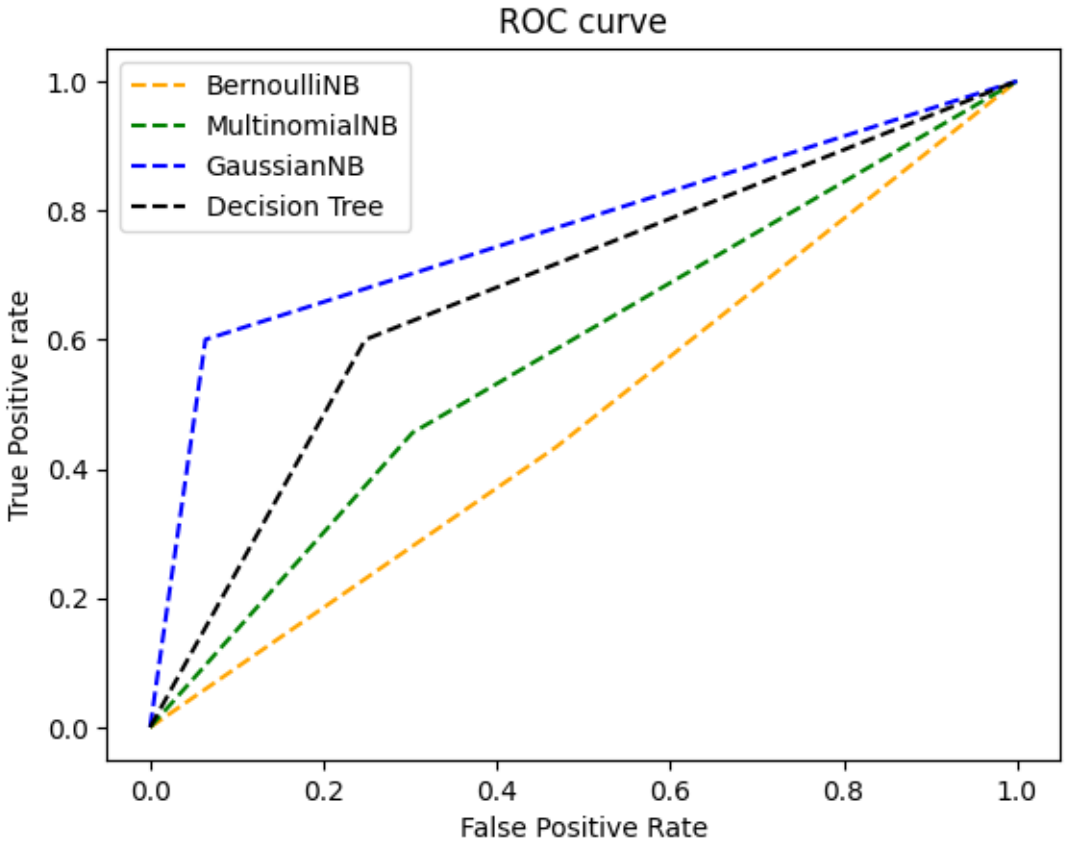
[84]:
```
keys = dict_dtr.keys()
y_points = []
for key in keys:
  y_points.append(dict_dtr[key][0]*100)
x_points = [float(key) for key in dict_dtr.keys()]


plt.plot(x_points, y_points)
plt.grid(True)
plt.show()
```




### 0.1.3 ROC curve and ROC_AUC score for all the classifier having maximum accuracy when train test split 70-30.

[79]:
```
from sklearn import metrics
def auc_roc():
    fpr1, tpr1, _1 = metrics.roc_curve(RocAucbnb['max']['y_test'],␣
 ↪RocAucbnb['max']['y_pred'], pos_label=1)
    fpr4, tpr4, _3 = metrics.roc_curve(RocAucmnb['max']['y_test'],␣
 ↪RocAucmnb['max']['y_pred'], pos_label=1)
    fpr2, tpr2, _2 = metrics.roc_curve(RocAucgnb['max']['y_test'],␣
 ↪RocAucgnb['max']['y_pred'], pos_label=1)
    fpr3, tpr3, _3 = metrics.roc_curve(RocAucdtr['max']['y_test'],␣
 ↪RocAucdtr['max']['y_pred'], pos_label=1)
    plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='BernoulliNB')
    plt.plot(fpr4, tpr4, linestyle='--',color='green', label='MultinomialNB')
    plt.plot(fpr2, tpr2, linestyle='--', color='blue', label= 'GaussianNB')
    plt.plot(fpr3, tpr3, linestyle='--', color='black', label= 'Decision Tree')
    plt.title('ROC curve')
    # x label
    plt.xlabel('False Positive Rate')
    # y label
```

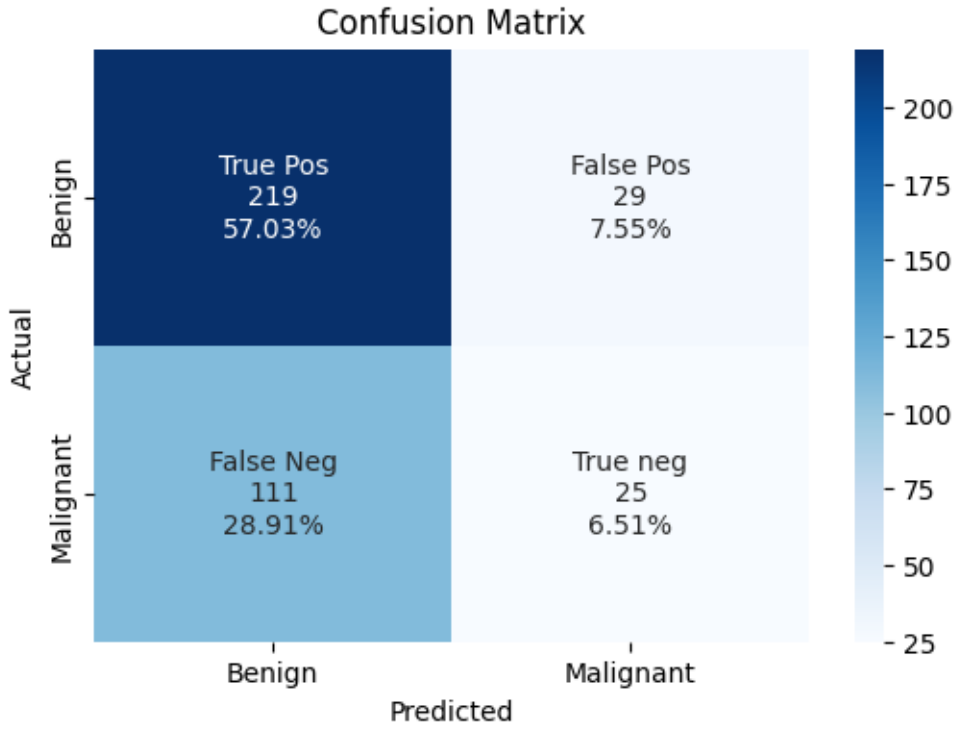
```
    plt.ylabel('True Positive rate')

    plt.legend(loc='best')
    plt.savefig('ROC',dpi=300)
    plt.show()
auc_roc()
```




**Best result for BernoulliNB Naive Bayes**

[80]:
```
# print( dict_dtr)
maxi=0
y_test = []
y_pred = []
for key in dict_bnb:
  if maxi < 100*dict_bnb[key][0]:
    maxi = 100*dict_bnb[key][0]
    y_test = dict_bnb[key][1]
    y_pred = dict_bnb[key][2]
reports(y_test, y_pred)
```

Confusion Matrix :



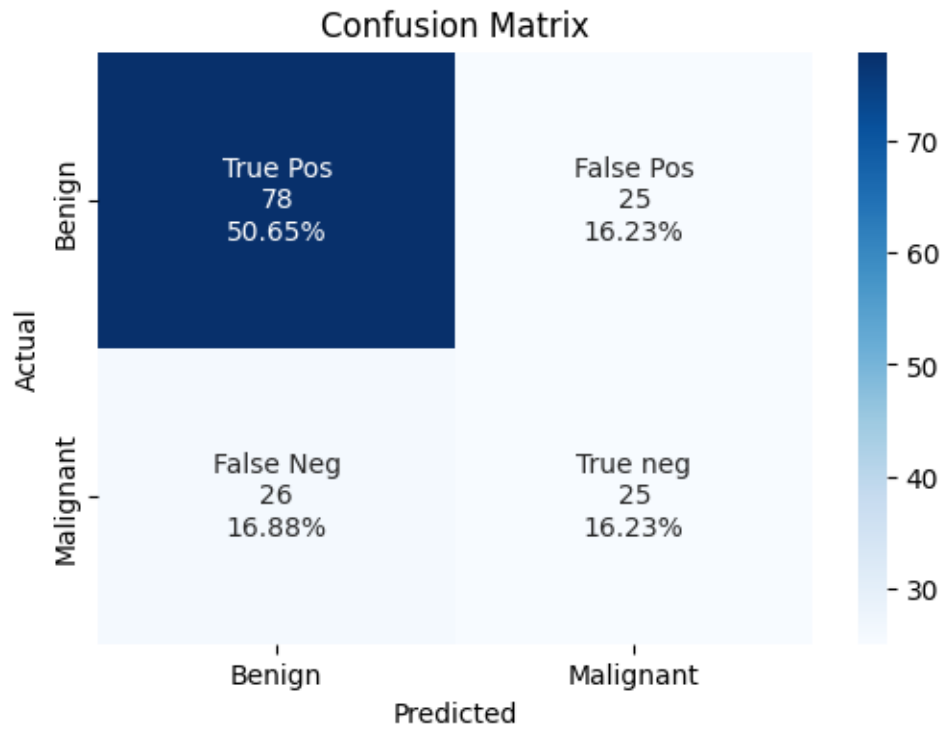

```
*********************************************
*********************************************
Classification Evaluation :
              precision    recall  f1-score   support

           0       0.66      0.88      0.76       248
           1       0.46      0.18      0.26       136

    accuracy                           0.64       384
   macro avg       0.56      0.53      0.51       384
weighted avg       0.59      0.64      0.58       384
```

**Best result for Multinomial Naive Bayes**

[81]:
```
# print( dict_dtr)
maxi=0
y_test = []
y_pred = []
for key in dict_mnb:
  if maxi < 100*dict_mnb[key][0]:
    maxi = 100*dict_mnb[key][0]
```

```
    y_test = dict_mnb[key][1]
    y_pred = dict_mnb[key][2]
reports(y_test, y_pred)
```

```
Confusion Matrix :
```



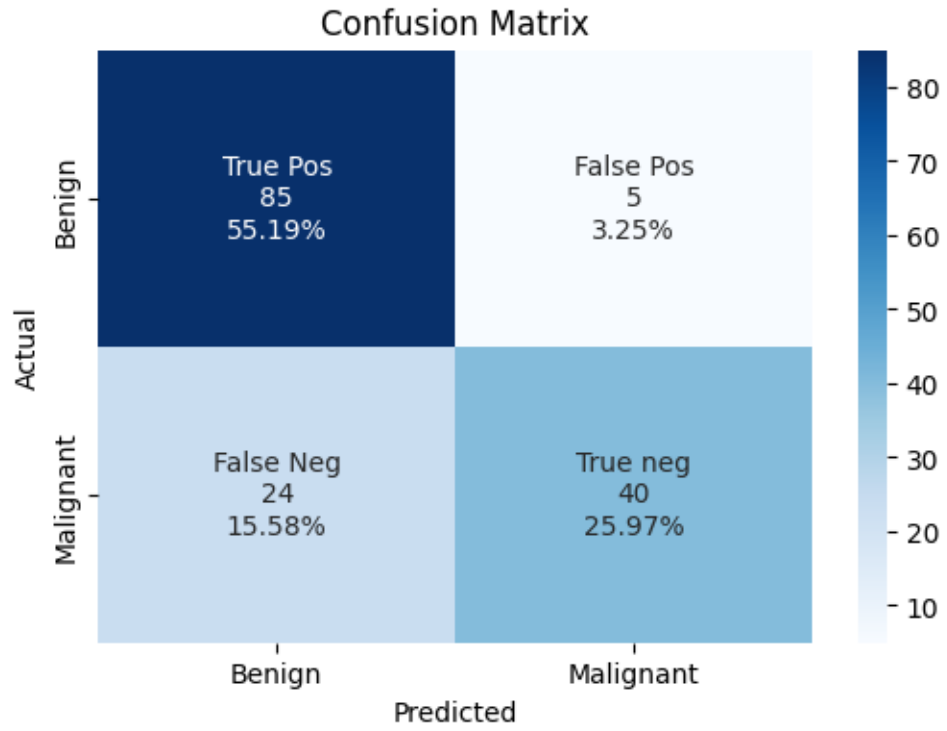

```
**********************************************
**********************************************
Classification Evaluation :
              precision    recall  f1-score   support

           0       0.75      0.76      0.75       103
           1       0.50      0.49      0.50        51

    accuracy                           0.67       154
   macro avg       0.62      0.62      0.62       154
weighted avg       0.67      0.67      0.67       154
```

**Best result for Gaussian Naive Bayes**

[82]:
```
# print( dict_dtr)
maxi=0
y_test = []
```

```
y_pred = []
for key in dict_gnb:
  if maxi < 100*dict_gnb[key][0]:
    maxi = 100*dict_gnb[key][0]
    y_test = dict_gnb[key][1]
    y_pred = dict_gnb[key][2]
reports(y_test, y_pred)
```

Confusion Matrix :

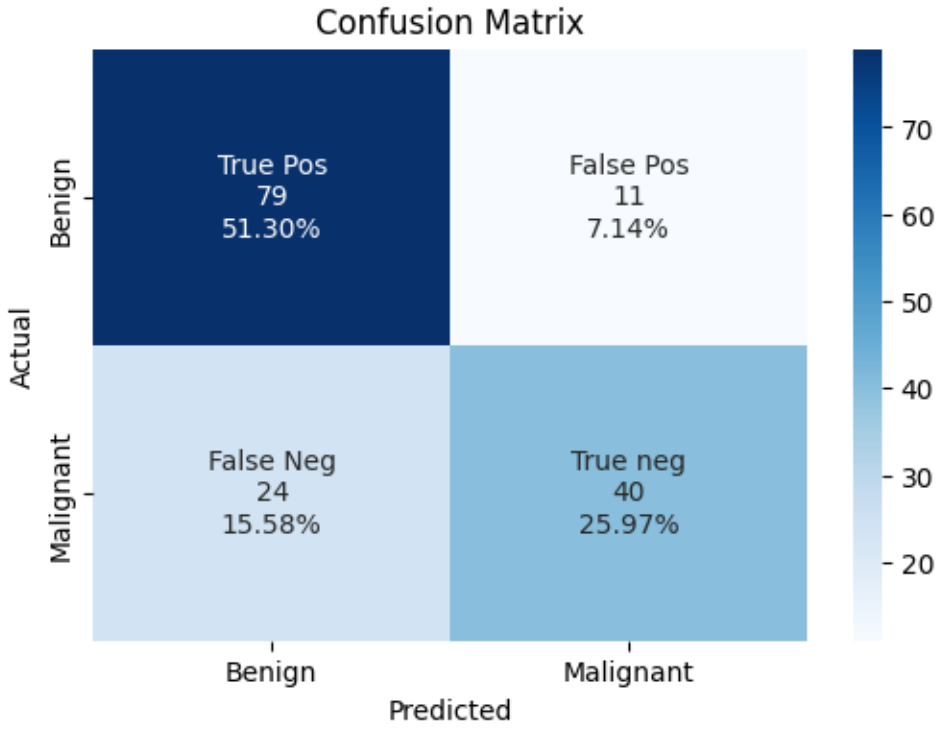



```
*********************************************
*********************************************
Classification Evaluation :
              precision    recall  f1-score   support

           0       0.78      0.94      0.85        90
           1       0.89      0.62      0.73        64

    accuracy                           0.81       154
   macro avg       0.83      0.78      0.79       154
weighted avg       0.83      0.81      0.80       154
```

**Best result for Decision Tree Classifier**

[83]:
```
# print( dict_dtr)
maxi=0
y_test = []
y_pred = []
for key in dict_dtr:
  if maxi < 100*dict_dtr[key][0]:
    maxi = 100*dict_dtr[key][0]
    y_test = dict_dtr[key][1]
    y_pred = dict_dtr[key][2]
reports(y_test, y_pred)
```

```
Confusion Matrix :
```




```
**********************************************
**********************************************
Classification Evaluation :
              precision    recall  f1-score   support

           0       0.77      0.88      0.82        90
           1       0.78      0.62      0.70        64

    accuracy                           0.77       154
   macro avg       0.78      0.75      0.76       154
weighted avg       0.77      0.77      0.77       154
```