In [355]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score
from sklearn.metrics import plot_roc_curve, plot_precision_recall_curve
from sklearn.metrics import classification_report,confusion_matrix
from sklearn import metrics
```

DATA PREPROCESSING

In [356]:

```
header=["loc","v(g)","ev(g)","iv(g)","n","v","l","d","i","e","b","t","lOCode","lOComment","
data=pd.read_csv("D://Downloads/Software/Software Dataset/promise1_useful.txt",names=header
data.head()
```

Out[356]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComment	lOB
0	3	1	1	1	1	0.00	0.00	0.00	0.00	0.00	...	0	12	
1	4	1	1	1	5	11.61	0.50	2.00	5.80	23.22	...	2	0	
2	9	2	1	1	15	51.89	0.23	4.38	11.86	227.03	...	0	8	
3	16	1	1	1	88	408.66	0.07	15.32	26.67	6261.24	...	0	4	
4	5	1	1	1	5	11.61	0.67	1.50	7.74	17.41	...	1	1	

5 rows × 22 columns

In [357]:

```
data=pd.DataFrame(data)

data.defects=data.defects.replace(True,1)
data.defects=data.defects.replace(False,0)
```

In [358]:

```
arr=np.array(data.defects)
print(np.where(arr==1))  #use shuffle as 1s and 0s are together
```

```
(array([51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
       68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
       85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98],
      dtype=int64),)
```

In [359]:

```
temp=np.array(data['defects'])
z=0
o=0
for i in temp:
    if(i==1):
        o+=1
    else:
        z+=1
print("ones: %d, zeroes: %d" %(o,z))
```

```
ones: 48, zeroes: 51
```

In [360]:

```
for i in range(16,len(header)-1):
    data[header[i]]=pd.to_numeric(data[header[i]], errors='coerce').astype('float32')
```

In [361]:

```
data=data.dropna(axis=0,how='any')
```

In [362]:

```
defects=data.loc[:,'defects']
data=data.drop('defects',axis=1)
```

In [363]:

```
from sklearn.preprocessing import Normalizer
transformer=Normalizer().fit(data)
x_scaled=transformer.transform(data)
data = pd.DataFrame(x_scaled,columns = ["loc","v(g)","ev(g)","iv(g)","n","v","l","d","i","e
data.head()
```

Out[363]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	
0	0.200446	0.066815	0.066815	0.066815	0.066815	0.000000	0.000000	0.000000	0.000000	0
1	0.140936	0.035234	0.035234	0.035234	0.176171	0.409068	0.017617	0.070468	0.204358	0
2	0.038330	0.008518	0.004259	0.004259	0.063883	0.220992	0.000980	0.018654	0.050510	0
3	0.002546	0.000159	0.000159	0.000159	0.014001	0.065019	0.000011	0.002437	0.004243	0
4	0.206293	0.041259	0.041259	0.041259	0.206293	0.479013	0.027643	0.061888	0.319342	0

5 rows × 21 columns

In [364]:

```
data['defects']=defects
#data=data.drop('lOCodeAndComment',axis=1)
#data=data.drop('lOBlank',axis=1)
#data=data.drop('lOComment',axis=1)
data=data.dropna(axis=0,how='any')
data.head()
```

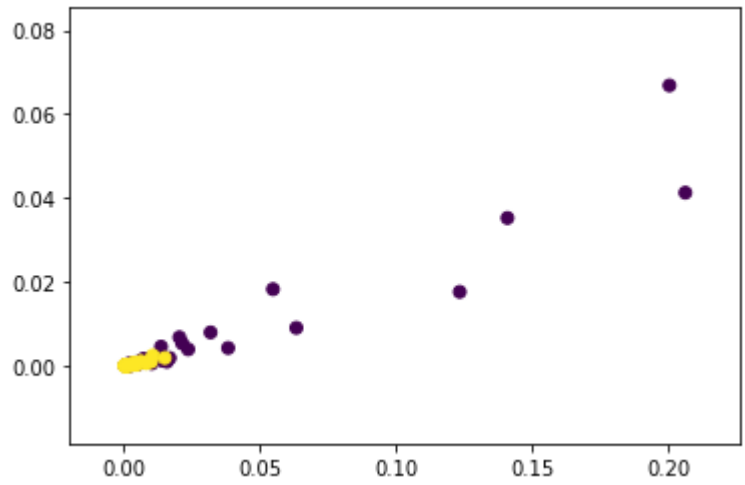
Out[364]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	
0	0.200446	0.066815	0.066815	0.066815	0.066815	0.000000	0.000000	0.000000	0.000000	0
1	0.140936	0.035234	0.035234	0.035234	0.176171	0.409068	0.017617	0.070468	0.204358	0
2	0.038330	0.008518	0.004259	0.004259	0.063883	0.220992	0.000980	0.018654	0.050510	0
3	0.002546	0.000159	0.000159	0.000159	0.014001	0.065019	0.000011	0.002437	0.004243	0
4	0.206293	0.041259	0.041259	0.041259	0.206293	0.479013	0.027643	0.061888	0.319342	0

5 rows × 22 columns

In [365]:

```
x=data['loc']
y=data['iv(g)']
z=data['defects']
plt.scatter(x,y,c=z)
plt.show()
```



In [366]:

```
x=data.drop('defects',axis=1).values
y=data[["defects"]].values
```

In [367]:

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y)
```

In [368]:

```
print(xtrain.shape, ytrain.shape, xtest.shape, ytest.shape)
```

```
(74, 21) (74, 1) (25, 21) (25, 1)
```

In [369]:

```
ytrain, ytest=ytrain.flatten(), ytest.flatten()
```

In [370]:

```
z=0
o=0
for i in ytrain:
    if(i==1):
        o+=1
    else:
        z+=1
print("ones: %d, zeroes: %d" %(o,z))
```

```
ones: 35, zeroes: 39
```

In [371]:

```
for i in range(0,len(ytrain)):
    if(ytrain[i]==1):
        print(xtrain[i])
        print("\n")
```

```
[2.67317510e-03 3.81882157e-04 5.45545939e-05 2.72772970e-04
 1.24930020e-02 7.31838966e-02 3.81882157e-06 7.41942477e-04
 5.37908296e-03 9.95639887e-01 2.45495673e-05 5.53134483e-02
 3.27327563e-04 2.18218376e-04 1.63663782e-04 0.00000000e+00
 8.18318909e-04 2.34584754e-03 8.23774368e-03 4.25525832e-03
 7.09209721e-04]


[3.95782003e-03 8.99504552e-04 8.99504552e-04 5.39702731e-04
 1.51116765e-02 7.93057183e-02 1.43920728e-05 2.25775643e-03
 6.31991898e-03 9.95089503e-01 2.69851366e-05 5.52835498e-02
 0.00000000e+00 3.05831548e-03 1.61910819e-03 0.00000000e+00
 3.05831548e-03 3.77791912e-03 9.53474825e-03 5.57692822e-03
 1.61910819e-03]


[5.87269483e-04 9.03491512e-05 4.06571180e-05 4.51745756e-05
 2.74209674e-03 1.93462830e-02 9.03491512e-08 2.33100810e-04
 3.74948977e-04 9.98267365e-01 6.45996431e-06 5.54592905e-02
```

BASE PREDICTIORS

1- SVM

In [372]:

```
from sklearn.svm import SVC
```

In [373]:

```
svm_model=SVC()
svm_model.fit(xtrain,ytrain)
```

Out[373]:

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [374]:

```
predsvm=svm_model.predict(xtest)
svm_model.score(xtest,ytest)*100
```

Out[374]:

```
48.0
```

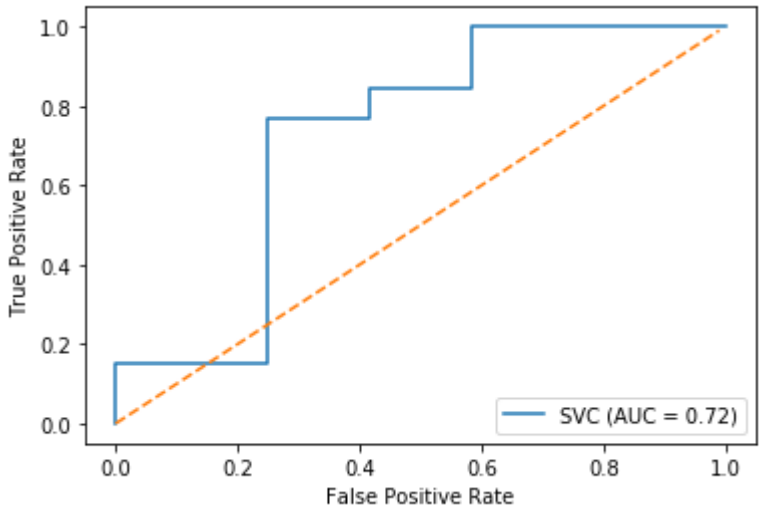
In [375]:

```
accuracy=confusion_matrix(ytest,predsvm)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predsvm, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predsvm))
print("\n")
print("   P    N")
print(confusion_matrix(ytest,predsvm))
print("\n")
print(classification_report(ytest,predsvm))
```

```
Accuracy:  48.0
Probability of detection of defect(Recall, pd):  0.0
Probability of false alarm(pf):  0.52
Probability of correct detection(Precision):  nan


F1-score or FM:  0.0
AUC value:  0.5


   P    N
[[12  0]
 [13  0]]


              precision    recall  f1-score   support

         0.0       0.48      1.00      0.65        12
         1.0       0.00      0.00      0.00        13

    accuracy                           0.48        25
   macro avg       0.24      0.50      0.32        25
weighted avg       0.23      0.48      0.31        25


C:\ProgramData\Anaconda3\envs\myenv\lib\site-packages\ipykernel_launcher.py:
10: RuntimeWarning: invalid value encountered in longlong_scalars
  # Remove the CWD from sys.path while we load stuff.
C:\ProgramData\Anaconda3\envs\myenv\lib\site-packages\sklearn\metrics\_class
ification.py:1272: UndefinedMetricWarning: Precision and F-score are ill-def
ined and being set to 0.0 in labels with no predicted samples. Use `zero_div
ision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

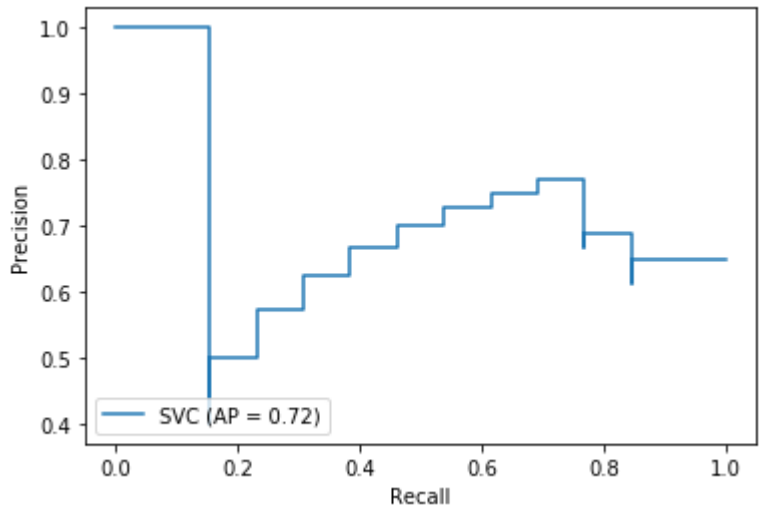
In [376]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(svm_model, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [377]:

```
disp = plot_precision_recall_curve(svm_model, xtest, ytest)
plt.show()
```



2- KNN

In [378]:

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=9)
```

In [379]:

```
knn.fit(xtrain,ytrain)
```

Out[379]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=9, p=2,
                     weights='uniform')
```

In [380]:

```
predknn=knn.predict(xtest)
knn.score(xtest,ytest)*100
```

Out[380]:

```
64.0
```

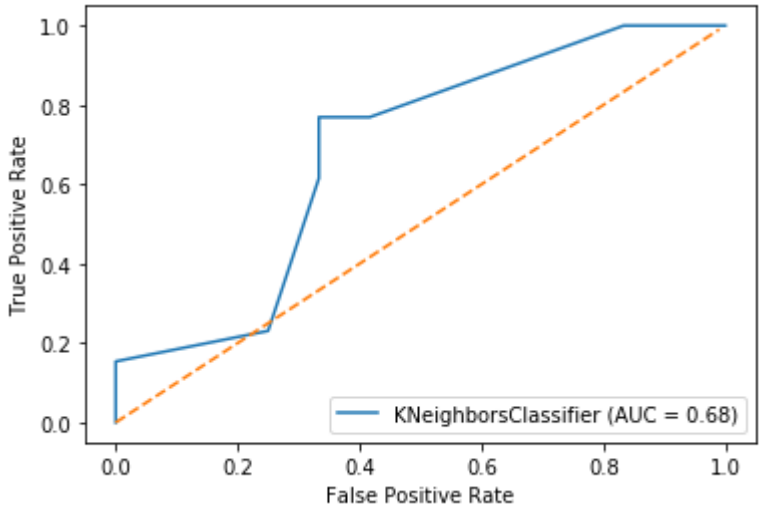
In [381]:

```
accuracy=confusion_matrix(ytest,predknn)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predknn, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predknn))
print("\n")
print(confusion_matrix(ytest,predknn))
print("\n")
print(classification_report(ytest,predknn))
```

```
Accuracy:  64.0
Probability of detection of defect(Recall, pd):  0.6153846153846154
Probability of false alarm(pf):  0.38461538461538464
Probability of correct detection(Precision):  0.6666666666666666


F1-score or FM:  0.64
AUC value:  0.6410256410256411


[[8 4]
 [5 8]]


              precision    recall  f1-score   support

         0.0       0.62      0.67      0.64        12
         1.0       0.67      0.62      0.64        13

    accuracy                           0.64        25
   macro avg       0.64      0.64      0.64        25
weighted avg       0.64      0.64      0.64        25
```

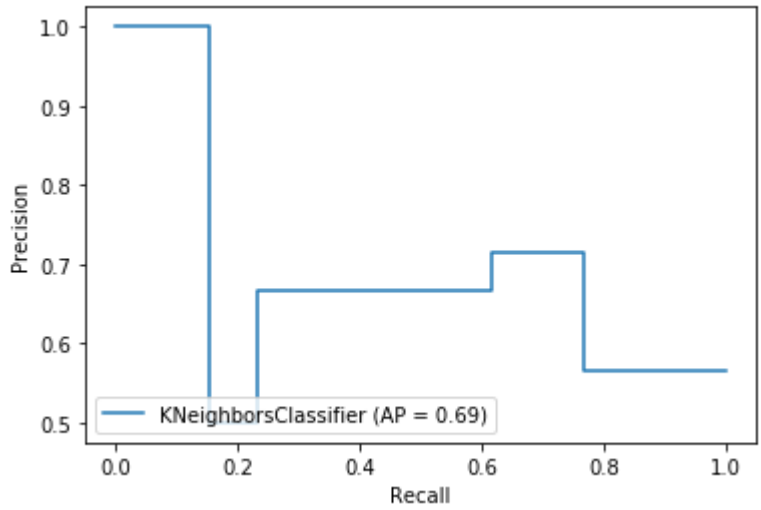
In [382]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(knn, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [383]:

```
disp = plot_precision_recall_curve(knn, xtest, ytest)
plt.show()
```



In [384]:

```
# try K=1 through K=25 and record testing accuracy
k_range = range(1, 15)

# We can create Python dictionary using [] or dict()
scores = []

# We use a loop through the range 1 to 26
# We append the scores in the dictionary
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(xtrain, ytrain)
    y_pred = knn.predict(xtest)
    scores.append(metrics.accuracy_score(ytest, y_pred))

print(scores)
```

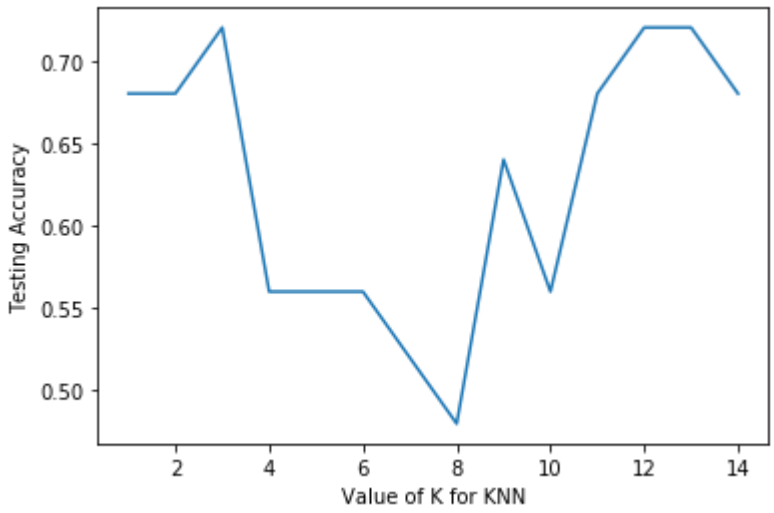
```
[0.68, 0.68, 0.72, 0.56, 0.56, 0.56, 0.52, 0.48, 0.64, 0.56, 0.68, 0.72, 0.7
2, 0.68]
```

In [385]:

```
# plot the relationship between K and testing accuracy
# plt.plot(x_axis, y_axis)
plt.plot(k_range, scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Testing Accuracy')
```

Out[385]:

```
Text(0, 0.5, 'Testing Accuracy')
```



3- NAIVE BAYES

In [386]:

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
```

In [387]:

```
gnb.fit(xtrain,ytrain)
```

Out[387]:

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [388]:

```
predg=gnb.predict(xtest)
gnb.score(xtest,ytest)*100
```

Out[388]:

```
72.0
```

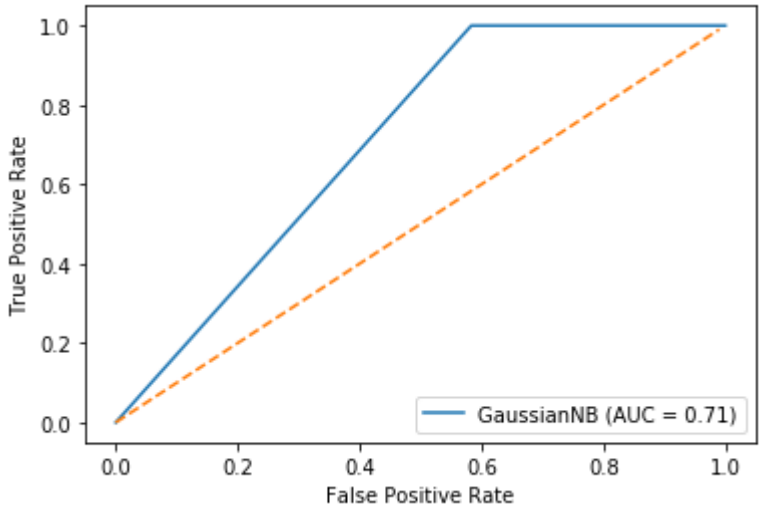
In [389]:

```
accuracy=confusion_matrix(ytest,predg)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predg, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predg))
print("\n")
print(confusion_matrix(ytest,predg))
print("\n")
print(classification_report(ytest,predg))
```

```
Accuracy:  72.0
Probability of detection of defect(Recall, pd):  1.0
Probability of false alarm(pf):  0.0
Probability of correct detection(Precision):  0.65


F1-score or FM:  0.787878787878788
AUC value:  0.7083333333333333


[[ 5  7]
 [ 0 13]]


              precision    recall  f1-score   support

         0.0       1.00      0.42      0.59        12
         1.0       0.65      1.00      0.79        13

    accuracy                           0.72        25
   macro avg       0.82      0.71      0.69        25
weighted avg       0.82      0.72      0.69        25
```

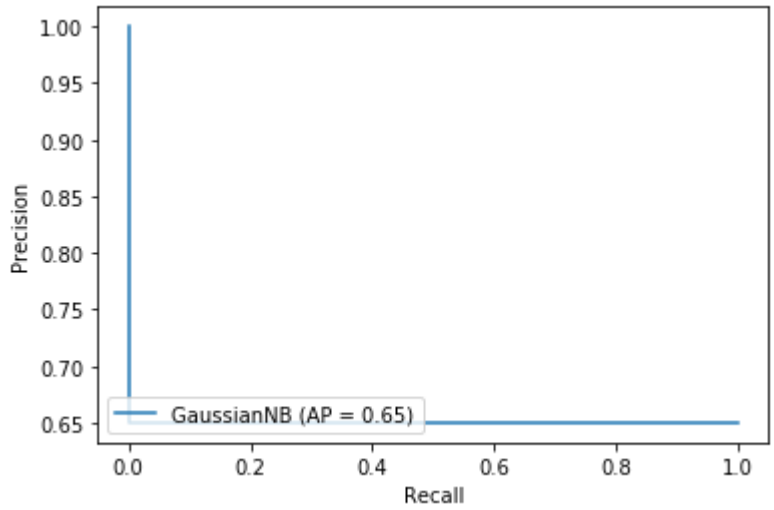
In [390]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(gnb, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [391]:

```
disp = plot_precision_recall_curve(gnb, xtest, ytest)
plt.show()
```



In [392]:

```
c=0
l=len(ytest)
for i in range(0,l):
    if(predg[i]!=ytest[i]):
        c=c+1
print("Number of mislabeled points out of a total %d points : %d" %(l,c))
```

```
Number of mislabeled points out of a total 25 points : 7
```

4- LOGISTIC REGRESSION

In [393]:

```
from sklearn.linear_model import LogisticRegression
logmodel=LogisticRegression()
```

In [394]:

```
logmodel.fit(xtrain,ytrain)
```

Out[394]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [395]:

```
predlog=logmodel.predict(xtest)
logistic_score=logmodel.score(xtest,ytest)*100
logistic_score
```

Out[395]:

```
52.0
```

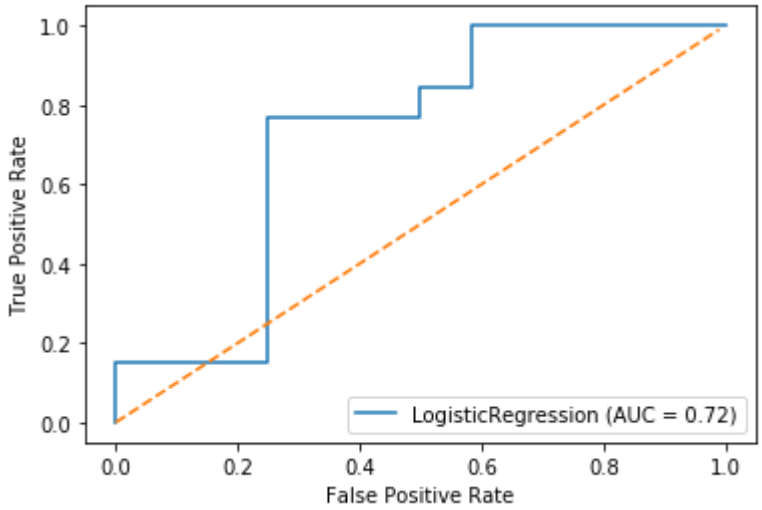
In [396]:

```
accuracy=confusion_matrix(ytest,predlog)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predlog, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predlog))
print("\n")
print(confusion_matrix(ytest,predlog))
print("\n")
print(classification_report(ytest,predlog))
```

```
Accuracy:  52.0
Probability of detection of defect(Recall, pd):  0.07692307692307693
Probability of false alarm(pf):  0.5
Probability of correct detection(Precision):  1.0


F1-score or FM:  0.14285714285714288
AUC value:  0.5384615384615384


[[12  0]
 [12  1]]


              precision    recall  f1-score   support

         0.0       0.50      1.00      0.67        12
         1.0       1.00      0.08      0.14        13

    accuracy                           0.52        25
   macro avg       0.75      0.54      0.40        25
weighted avg       0.76      0.52      0.39        25
```

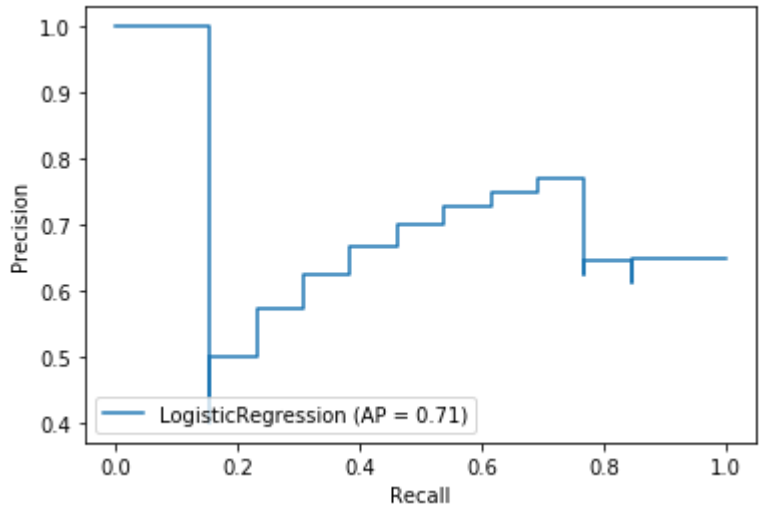
In [397]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(logmodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [398]:

```
disp = plot_precision_recall_curve(logmodel, xtest, ytest)
plt.show()
```



5- MLP

In [399]:

```
from sklearn.neural_network import MLPClassifier
```

In [400]:

```
model=MLPClassifier(hidden_layer_sizes=(20,20),max_iter=2000)
model.fit(xtrain,ytrain)
```

Out[400]:

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.
9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(20, 20), learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=2000,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=None, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
```

In [401]:

```
predn=model.predict(xtest)
model.score(xtest,ytest)*100
```

Out[401]:

```
72.0
```

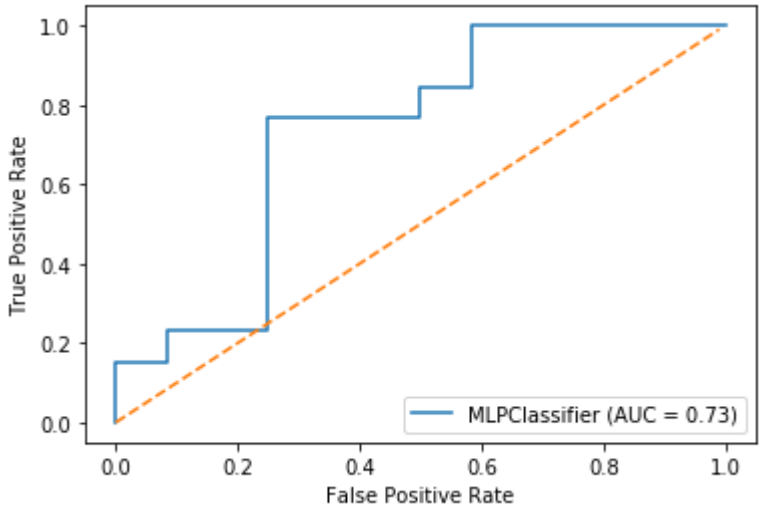
In [402]:

```
accuracy=confusion_matrix(ytest,predn)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predn, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predn))
print("\n")
print(confusion_matrix(ytest,predn))
print("\n")
print(classification_report(ytest,predn))
```

```
Accuracy:  72.0
Probability of detection of defect(Recall, pd):  0.7692307692307693
Probability of false alarm(pf):  0.2727272727272727
Probability of correct detection(Precision):  0.7142857142857143


F1-score or FM:  0.7407407407407408
AUC value:  0.717948717948718


[[ 8  4]
 [ 3 10]]


              precision    recall  f1-score   support

         0.0       0.73      0.67      0.70        12
         1.0       0.71      0.77      0.74        13

    accuracy                           0.72        25
   macro avg       0.72      0.72      0.72        25
weighted avg       0.72      0.72      0.72        25
```

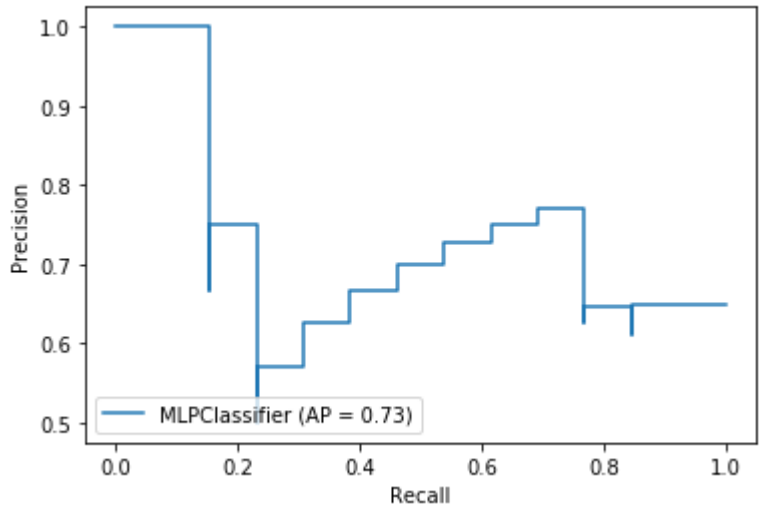
In [403]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(model, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [404]:

```
disp = plot_precision_recall_curve(model, xtest, ytest)
plt.show()
```



6- DECISION TREE

In [405]:

```
from sklearn import tree
```

In [406]:

```
tmodel=tree.DecisionTreeClassifier()
tmodel.fit(xtrain,ytrain)
```

Out[406]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=Non
e,
                       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')
```

In [407]:

```
predt=tmodel.predict(xtest)
tmodel.score(xtest,ytest)*100
```

Out[407]:

```
68.0
```

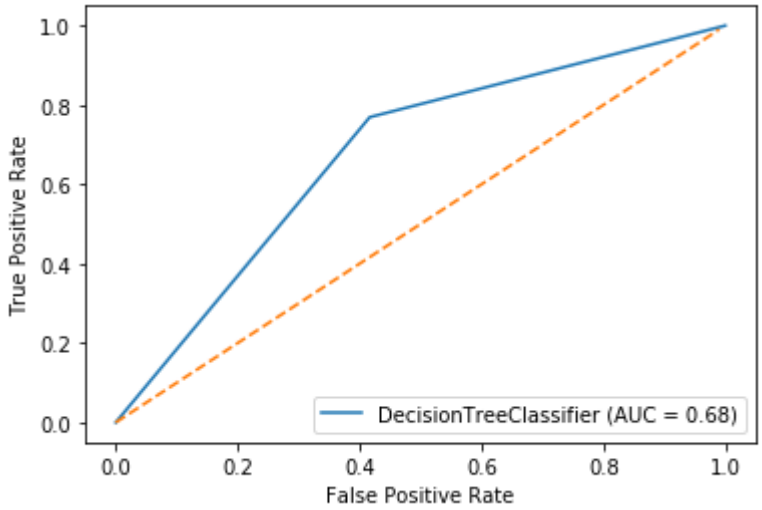
In [408]:

```
accuracy=confusion_matrix(ytest,predt)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predt, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predt))
print("\n")
print(confusion_matrix(ytest,predt))
print("\n")
print(classification_report(ytest,predt))
```

```
Accuracy:  68.0
Probability of detection of defect(Recall, pd):  0.7692307692307693
Probability of false alarm(pf):  0.3
Probability of correct detection(Precision):  0.6666666666666666


F1-score or FM:  0.7142857142857142
AUC value:  0.6762820512820512


[[ 7  5]
 [ 3 10]]


              precision    recall  f1-score   support

         0.0       0.70      0.58      0.64        12
         1.0       0.67      0.77      0.71        13

    accuracy                           0.68        25
   macro avg       0.68      0.68      0.68        25
weighted avg       0.68      0.68      0.68        25
```

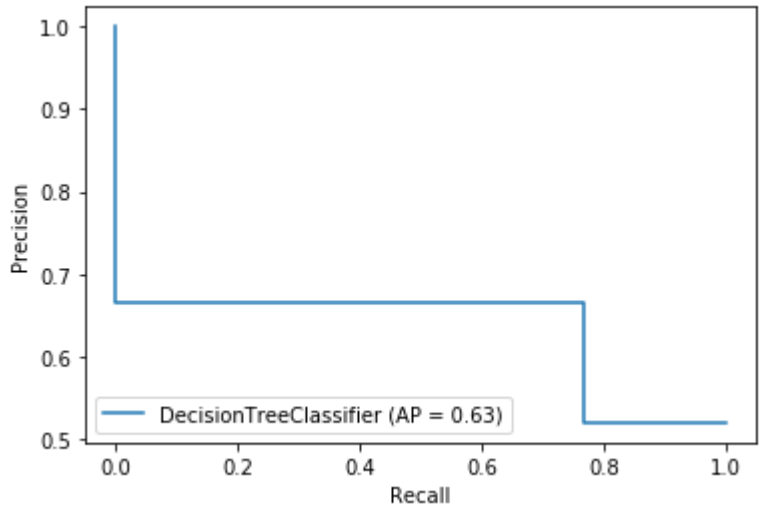
In [409]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(tmodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [410]:

```
disp = plot_precision_recall_curve(tmodel, xtest, ytest)
plt.show()
```



ENSEMBLE PREDICTORS

1- ADABOOST

In [411]:

```
from sklearn.ensemble import AdaBoostClassifier
```

In [412]:

```
adamodel = AdaBoostClassifier(n_estimators=100)
adamodel.fit(xtrain,ytrain)
```

Out[412]:

```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=
1.0,
                   n_estimators=100, random_state=None)
```

In [413]:

```
predada=adamodel.predict(xtest)
adamodel.score(xtest,ytest)*100
```

Out[413]:

```
88.0
```

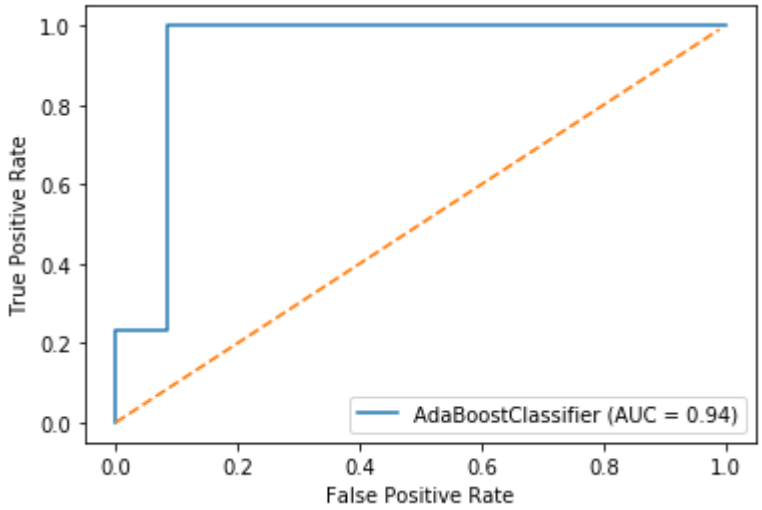
In [414]:

```
accuracy=confusion_matrix(ytest,predada)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predada, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predada))
print("\n")
print(confusion_matrix(ytest,predada))
print("\n")
print(classification_report(ytest,predada))
```

```
Accuracy:  88.0
Probability of detection of defect(Recall, pd):  0.8461538461538461
Probability of false alarm(pf):  0.15384615384615385
Probability of correct detection(Precision):  0.9166666666666666


F1-score or FM:  0.8799999999999999
AUC value:  0.8814102564102564


[[11  1]
 [ 2 11]]


              precision    recall  f1-score   support

         0.0       0.85      0.92      0.88        12
         1.0       0.92      0.85      0.88        13

    accuracy                           0.88        25
   macro avg       0.88      0.88      0.88        25
weighted avg       0.88      0.88      0.88        25
```

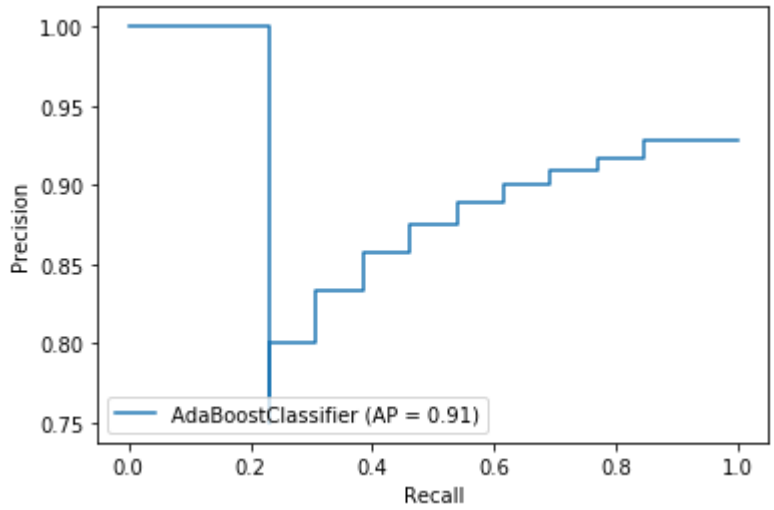
In [415]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(adamodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [416]:

```
disp = plot_precision_recall_curve(adamodel, xtest, ytest)
plt.show()
```



2- BAGGING

In [417]:

```
from sklearn.ensemble import BaggingClassifier
```

In [418]:

```
bagmodel = BaggingClassifier(base_estimator=None, n_estimators=10) #default=decision tree,
bagmodel.fit(xtrain, ytrain)
```

Out[418]:

```
BaggingClassifier(base_estimator=None, bootstrap=True, bootstrap_features=Fa
lse,
                  max_features=1.0, max_samples=1.0, n_estimators=10,
                  n_jobs=None, oob_score=False, random_state=None, verbose=
0,
                  warm_start=False)
```

In [419]:

```
predbag=bagmodel.predict(xtest)
bagmodel.score(xtest, ytest)*100
```

Out[419]:

```
60.0
```

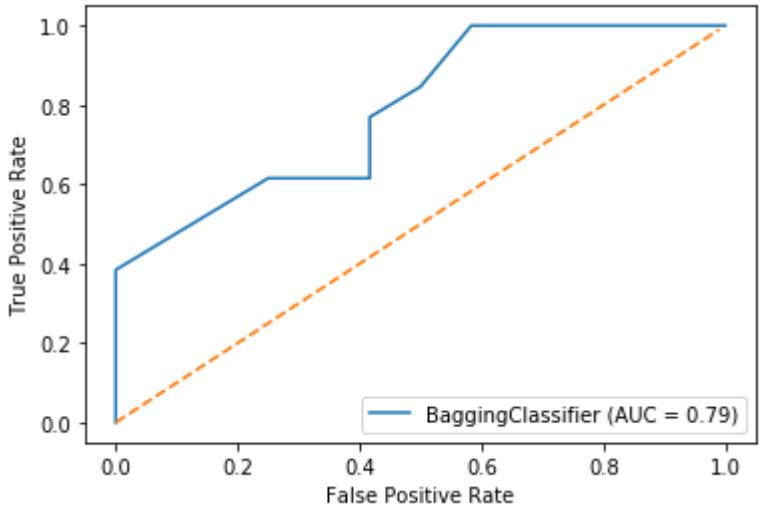
In [420]:

```
accuracy=confusion_matrix(ytest,predbag)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predbag, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predbag))
print("\n")
print(confusion_matrix(ytest,predbag))
print("\n")
print(classification_report(ytest,predbag))
```

```
Accuracy:  60.0
Probability of detection of defect(Recall, pd):  0.6153846153846154
Probability of false alarm(pf):  0.4166666666666667
Probability of correct detection(Precision):  0.6153846153846154


F1-score or FM:  0.6153846153846154
AUC value:  0.5993589743589743


[[7 5]
 [5 8]]


              precision    recall  f1-score   support

         0.0       0.58      0.58      0.58        12
         1.0       0.62      0.62      0.62        13

    accuracy                           0.60        25
   macro avg       0.60      0.60      0.60        25
weighted avg       0.60      0.60      0.60        25
```

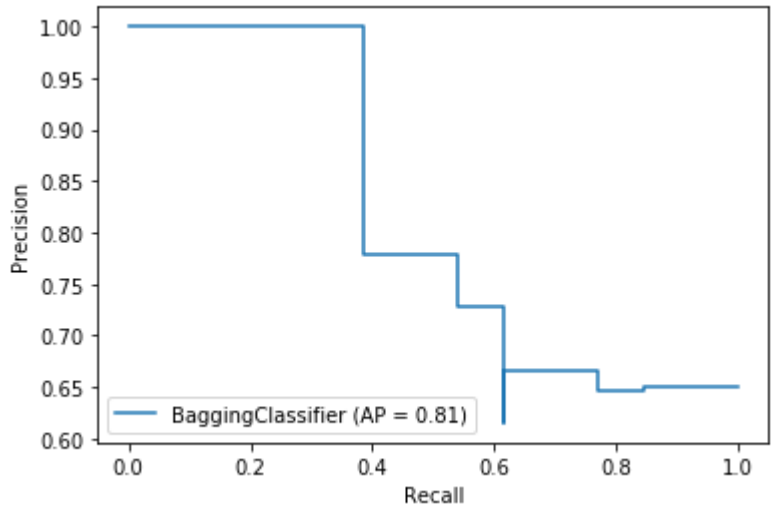
In [421]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(bagmodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [422]:

```
disp = plot_precision_recall_curve(bagmodel, xtest, ytest)
plt.show()
```



3- Extra_Tree_Classifier

In [423]:

```
from sklearn.ensemble import ExtraTreesClassifier
```

In [424]:

```
exmodel = ExtraTreesClassifier(n_estimators=100)
exmodel.fit(xtrain, ytrain)
```

Out[424]:

```
ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                     criterion='gini', max_depth=None, max_features='auto',
                     max_leaf_nodes=None, max_samples=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=100,
                     n_jobs=None, oob_score=False, random_state=None, verbos
e=0,
                     warm_start=False)
```

In [425]:

```
predex=exmodel.predict(xtest)
exmodel.score(xtest,ytest)*100
```

Out[425]:

```
84.0
```

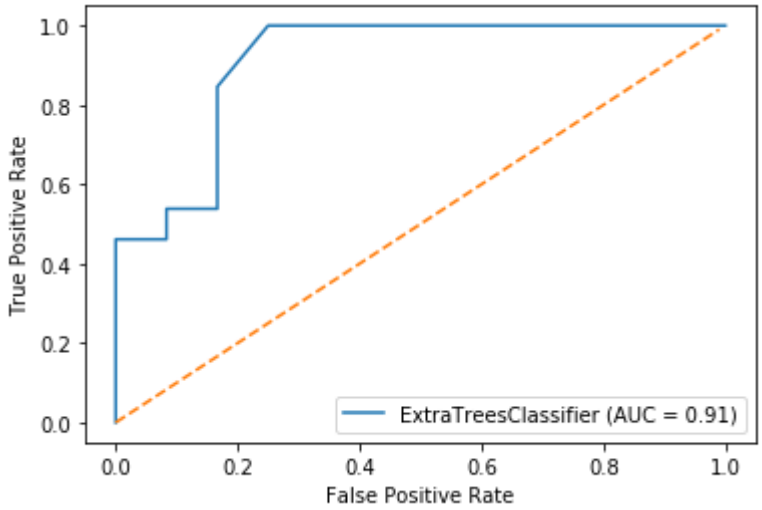
In [426]:

```
accuracy=confusion_matrix(ytest,predex)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predex, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predex))
print("\n")
print(confusion_matrix(ytest,predex))
print("\n")
print(classification_report(ytest,predex))
```

```
Accuracy:  84.0
Probability of detection of defect(Recall, pd):  0.8461538461538461
Probability of false alarm(pf):  0.16666666666666666
Probability of correct detection(Precision):  0.8461538461538461


F1-score or FM:  0.8461538461538461
AUC value:  0.8397435897435898


[[10  2]
 [ 2 11]]


              precision    recall  f1-score   support

         0.0       0.83      0.83      0.83        12
         1.0       0.85      0.85      0.85        13

    accuracy                           0.84        25
   macro avg       0.84      0.84      0.84        25
weighted avg       0.84      0.84      0.84        25
```

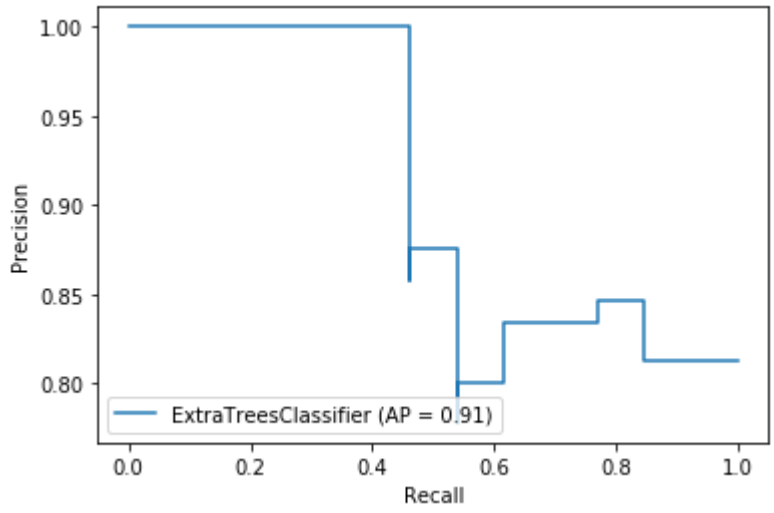
In [427]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(exmodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [428]:

```
disp = plot_precision_recall_curve(exmodel, xtest, ytest)
plt.show()
```



4- Gradient_Boosting_Classifier

In [429]:

```
from sklearn.ensemble import GradientBoostingClassifier
```

In [430]:

```
gradmodel = GradientBoostingClassifier()
gradmodel.fit(xtrain,ytrain)
```

Out[430]:

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=Non
e,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=Non
e,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

In [431]:

```
predgrad=gradmodel.predict(xtest)
gradmodel.score(xtest,ytest)*100
```

Out[431]:

```
80.0
```

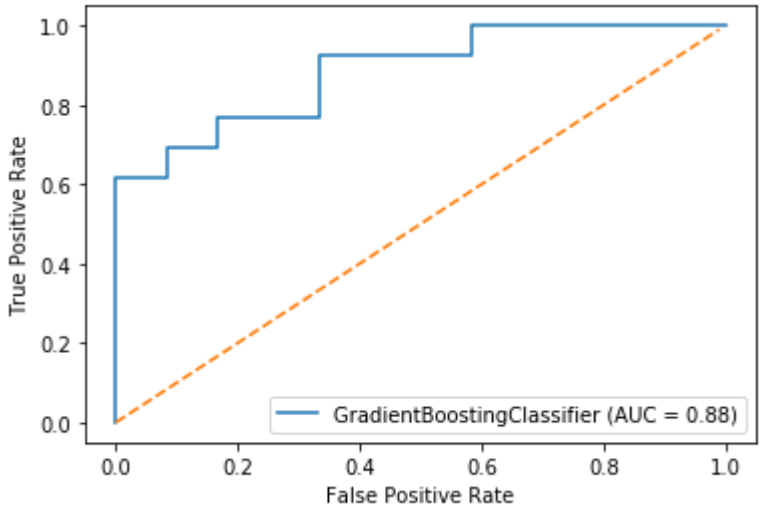
In [432]:

```
accuracy=confusion_matrix(ytest,predgrad)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predgrad, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predgrad))
print("\n")
print(confusion_matrix(ytest,predgrad))
print("\n")
print(classification_report(ytest,predgrad))
```

```
Accuracy:  80.0
Probability of detection of defect(Recall, pd):  0.7692307692307693
Probability of false alarm(pf):  0.23076923076923078
Probability of correct detection(Precision):  0.8333333333333334


F1-score or FM:  0.8
AUC value:  0.8012820512820513


[[10  2]
 [ 3 10]]


              precision    recall  f1-score   support

         0.0       0.77      0.83      0.80        12
         1.0       0.83      0.77      0.80        13

    accuracy                           0.80        25
   macro avg       0.80      0.80      0.80        25
weighted avg       0.80      0.80      0.80        25
```

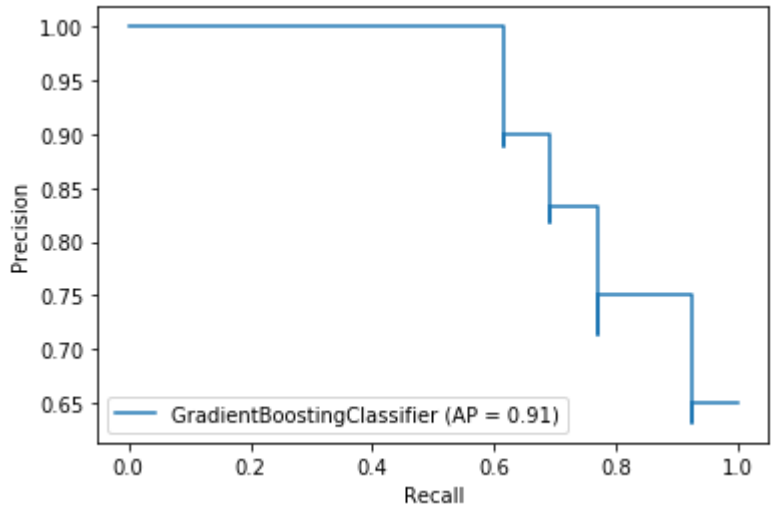
In [433]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(gradmodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [434]:

```
disp = plot_precision_recall_curve(gradmodel, xtest, ytest)
plt.show()
```



5- Random_Forest_Classifier

In [435]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [436]:

```
randmodel = RandomForestClassifier()
randmodel.fit(xtrain,ytrain)
```

Out[436]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='aut
o',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [437]:

```
predrand=randmodel.predict(xtest)
randmodel.score(xtest,ytest)*100
```

Out[437]:

```
76.0
```

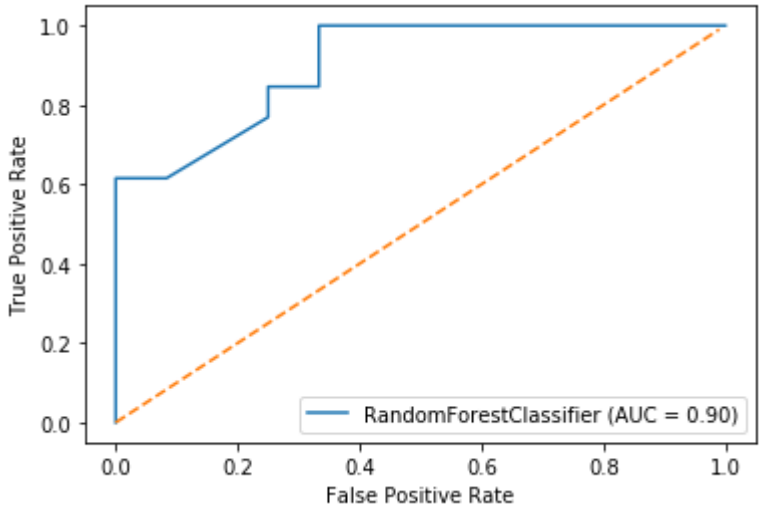
In [438]:

```
accuracy=confusion_matrix(ytest,predrand)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predrand, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predrand))
print("\n")
print(confusion_matrix(ytest,predrand))
print("\n")
print(classification_report(ytest,predrand))
```

```
Accuracy:  76.0
Probability of detection of defect(Recall, pd):  0.6923076923076923
Probability of false alarm(pf):  0.2857142857142857
Probability of correct detection(Precision):  0.8181818181818182


F1-score or FM:  0.7500000000000001
AUC value:  0.7628205128205129


[[10  2]
 [ 4  9]]


              precision    recall  f1-score   support

         0.0       0.71      0.83      0.77        12
         1.0       0.82      0.69      0.75        13

    accuracy                           0.76        25
   macro avg       0.77      0.76      0.76        25
weighted avg       0.77      0.76      0.76        25
```

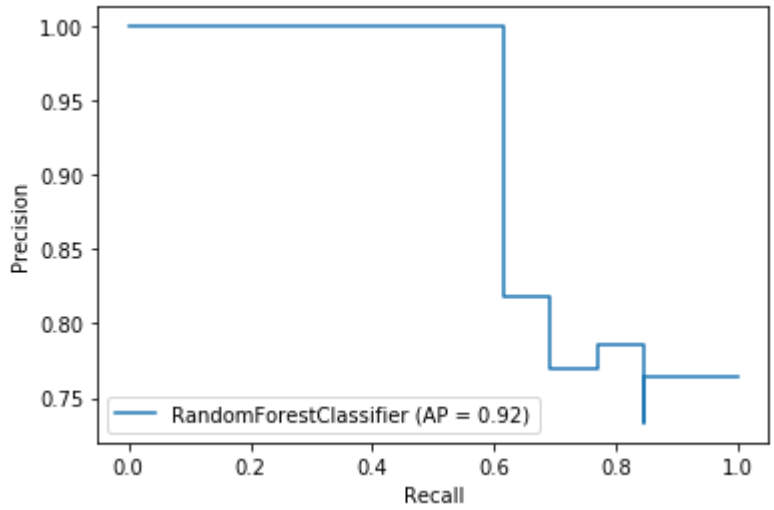
In [439]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(randmodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [440]:

```
disp = plot_precision_recall_curve(randmodel, xtest, ytest)
plt.show()
```



6- Stacking_Classifier

In [441]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import StackingClassifier
```

In [442]:

```
estimators = [('rf', RandomForestClassifier(n_estimators=10, random_state=42)),
              ('svr', make_pipeline(StandardScaler(), LinearSVC(random_state=42)))]

stmodel = StackingClassifier(estimators=estimators, final_estimator=LogisticRegression())
stmodel.fit(xtrain,ytrain)
```

Out[442]:

```
StackingClassifier(cv=None,
                   estimators=[('rf',
                                RandomForestClassifier(bootstrap=True,
                                                       ccp_alpha=0.0,
                                                       class_weight=None,
                                                       criterion='gini',
                                                       max_depth=None,
                                                       max_features='auto',
                                                       max_leaf_nodes=None,
                                                       max_samples=None,
                                                       min_impurity_decrease
=0.0,
                                                       min_impurity_split=No
ne,
                                                       min_samples_leaf=1,
                                                       min_samples_split=2,
                                                       min_weight_fraction_l
eaf=0.0,
                                                       n_estimators=10,
                                                       n_jobs=None,...
                                                           tol=0.0001,
                                                           verbose=0))],
                                               verbose=False))],
                   final_estimator=LogisticRegression(C=1.0, class_weight=No
ne,
                                                      dual=False,
                                                      fit_intercept=True,
                                                      intercept_scaling=1,
                                                      l1_ratio=None,
                                                      max_iter=100,
                                                      multi_class='auto',
                                                      n_jobs=None, penalty
='l2',
                                                      random_state=None,
                                                      solver='lbfgs',
                                                      tol=0.0001, verbose=0,
                                                      warm_start=False),
                   n_jobs=None, passthrough=False, stack_method='auto',
                   verbose=0)
```

In [443]:

```
predst=stmodel.predict(xtest)
stmodel.score(xtest,ytest)*100
```

Out[443]:

```
88.0
```

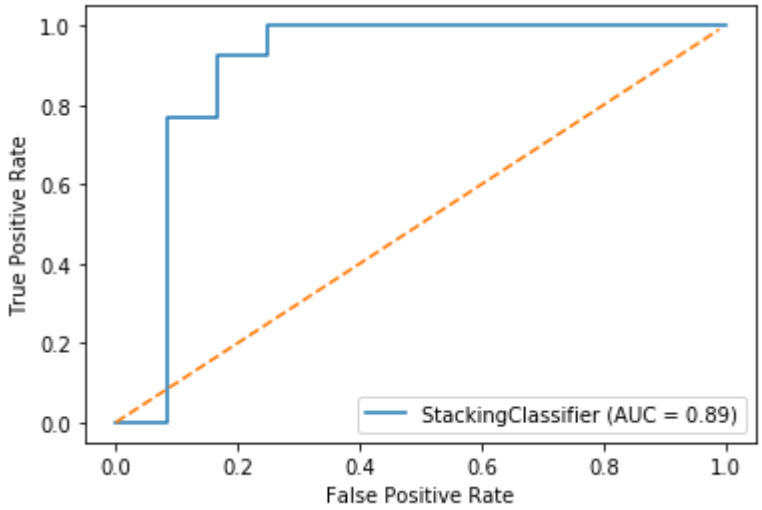
In [444]:

```
accuracy=confusion_matrix(ytest,predst)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predst, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predst))
print("\n")
print(confusion_matrix(ytest,predst))
print("\n")
print(classification_report(ytest,predst))
```

```
Accuracy:  88.0
Probability of detection of defect(Recall, pd):  1.0
Probability of false alarm(pf):  0.0
Probability of correct detection(Precision):  0.8125


F1-score or FM:  0.896551724137931
AUC value:  0.875


[[ 9  3]
 [ 0 13]]


              precision    recall  f1-score   support

         0.0       1.00      0.75      0.86        12
         1.0       0.81      1.00      0.90        13

    accuracy                           0.88        25
   macro avg       0.91      0.88      0.88        25
weighted avg       0.90      0.88      0.88        25
```

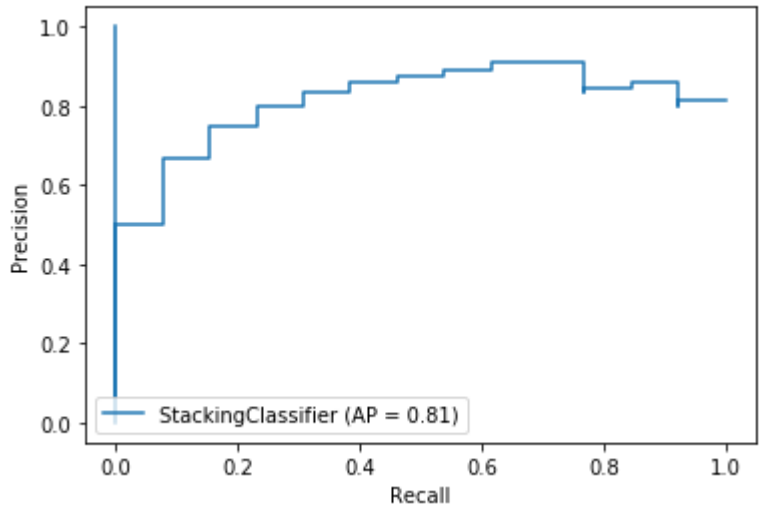
In [445]:

```
x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(stmodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()
```



In [446]:

```
disp = plot_precision_recall_curve(stmodel, xtest, ytest)
plt.show()
```



7- Voting_Classifier

In [447]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
```

In [448]:

```
clf1 = LogisticRegression()
clf2 = RandomForestClassifier()#n_estimators=50, random_state=1)
clf3 = GaussianNB()
votmodel = VotingClassifier(estimators=[('lr', clf1), ('rf', clf2), ('gnb', clf3)], voting=
votmodel.fit(xtrain,ytrain)
```

Out[448]:

```
VotingClassifier(estimators=[('lr',
                              LogisticRegression(C=1.0, class_weight=None,
                                                 dual=False, fit_intercept=T
rue,
                                                 intercept_scaling=1,
                                                 l1_ratio=None, max_iter=10
0,
                                                 multi_class='auto',
                                                 n_jobs=None, penalty='l2',
                                                 random_state=None,
                                                 solver='lbfgs', tol=0.0001,
                                                 verbose=0, warm_start=Fals
e)),
                             ('rf',
                              RandomForestClassifier(bootstrap=True,
                                                     ccp_alpha=0.0,
                                                     class_weight=None,
                                                     cr...
                                                     max_leaf_nodes=None,
                                                     max_samples=None,
                                                     min_impurity_decrease=
0.0,
                                                     min_impurity_split=Non
e,
                                                     min_samples_leaf=1,
                                                     min_samples_split=2,
                                                     min_weight_fraction_lea
f=0.0,
                                                     n_estimators=100,
                                                     n_jobs=None,
                                                     oob_score=False,
                                                     random_state=None,
                                                     verbose=0,
                                                     warm_start=False)),
                             ('gnb',
                              GaussianNB(priors=None, var_smoothing=1e-0
9))],
                 flatten_transform=True, n_jobs=None, voting='hard',
                 weights=None)
```

In [449]:

```
predvot=votmodel.predict(xtest)
votmodel.score(xtest,ytest)*100
```

Out[449]:

```
80.0
```

In [450]:

```
accuracy=confusion_matrix(ytest,predvot)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predvot, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predvot))
print("\n")
print(confusion_matrix(ytest,predvot))
print("\n")
print(classification_report(ytest,predvot))
```

```
Accuracy:  80.0
Probability of detection of defect(Recall, pd):  0.7692307692307693
Probability of false alarm(pf):  0.23076923076923078
Probability of correct detection(Precision):  0.8333333333333334


F1-score or FM:  0.8
AUC value:  0.8012820512820513


[[10  2]
 [ 3 10]]


              precision    recall  f1-score   support

         0.0       0.77      0.83      0.80        12
         1.0       0.83      0.77      0.80        13

    accuracy                           0.80        25
   macro avg       0.80      0.80      0.80        25
weighted avg       0.80      0.80      0.80        25
```

In [ ]:

```

```

In [ ]:

```

```