In [500]:

```
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 [501]:

```
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/promise2_data_1.txt",names=header
data.head()
```

Out[501]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComm
0	1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	...	2	
1	1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	...	1	
2	72.0	7.0	1.0	6.0	198.0	1134.13	0.05	20.31	55.85	23029.10	...	51	
3	190.0	3.0	1.0	3.0	600.0	4348.76	0.06	17.06	254.87	74202.67	...	129	
4	37.0	4.0	1.0	4.0	126.0	599.12	0.06	17.19	34.86	10297.30	...	28	

5 rows × 22 columns

In [502]:

```
data=pd.DataFrame(data)

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

In [503]:

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

```
(array([   1,    2,    3, ..., 2104, 2105, 2106], dtype=int64),)
```

In [504]:

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

In [505]:

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

In [506]:

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

In [507]:

```
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[507]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i
0	0.165213	0.210271	0.210271	0.210271	0.195252	0.195252	1.952515e-01	0.195252	0.195252
1	0.218218	0.218218	0.218218	0.218218	0.218218	0.218218	2.182179e-01	0.218218	0.218218
2	0.003118	0.000303	0.000043	0.000260	0.008574	0.049109	2.165060e-06	0.000879	0.002418
3	0.002552	0.000040	0.000013	0.000040	0.008059	0.058413	8.059231e-07	0.000229	0.003423
4	0.003581	0.000387	0.000097	0.000387	0.012195	0.057987	5.807237e-06	0.001664	0.003374

5 rows × 21 columns

In [508]:

```
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()
```

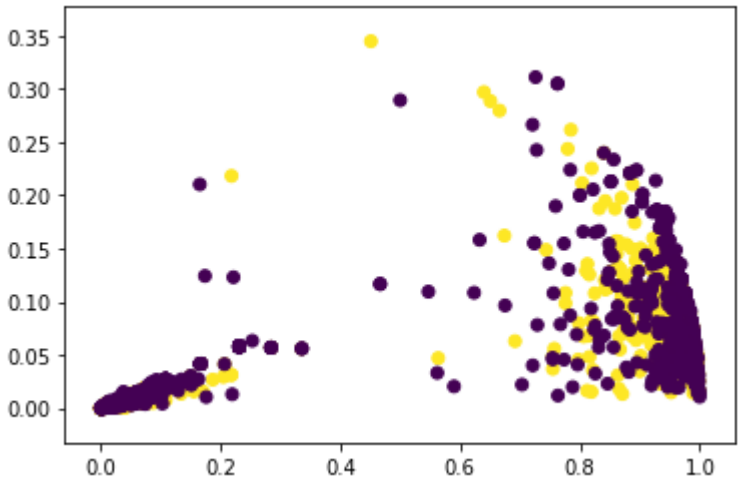
Out[508]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i
0	0.165213	0.210271	0.210271	0.210271	0.195252	0.195252	1.952515e-01	0.195252	0.195252
1	0.218218	0.218218	0.218218	0.218218	0.218218	0.218218	2.182179e-01	0.218218	0.218218
2	0.003118	0.000303	0.000043	0.000260	0.008574	0.049109	2.165060e-06	0.000879	0.002418
3	0.002552	0.000040	0.000013	0.000040	0.008059	0.058413	8.059231e-07	0.000229	0.003423
4	0.003581	0.000387	0.000097	0.000387	0.012195	0.057987	5.807237e-06	0.001664	0.003374

5 rows × 22 columns

In [509]:

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



In [510]:

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

In [511]:

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

In [512]:

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

```
(3369, 21) (3369, 1) (1124, 21) (1124, 1)
```

In [513]:

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

In [514]:

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

```
ones: 1548, zeroes: 1821
```

In [515]:

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

```
[7.77173703e-03 1.21433391e-03 2.42866782e-04 1.21433391e-03
 1.94293426e-02 9.53373554e-02 2.42866782e-05 2.53067187e-03
 9.14879169e-03 9.93519433e-01 3.15726817e-05 5.51963336e-02
 4.85733565e-03 2.42866782e-04 2.18580104e-03 0.00000000e+00
 2.67153461e-03 4.61446886e-03 1.06861384e-02 8.74320416e-03
 2.18580104e-03]


[1.70831387e-03 2.21858944e-04 2.21858944e-05 2.21858944e-04
 5.96800559e-03 3.23840845e-02 6.65576832e-07 6.83547406e-04
 1.05094582e-03 9.97906222e-01 1.08710883e-05 5.54392222e-02
 1.06492293e-03 4.21531994e-04 1.77487155e-04 0.00000000e+00
 3.54974310e-04 5.99019149e-04 3.66067258e-03 2.30733302e-03
 4.21531994e-04]


[5.68630655e-02 1.42157664e-02 1.42157664e-02 1.42157664e-02
 1.42157664e-01 4.26472991e-01 7.10788319e-03 2.84315328e-02
 2.13236496e-01 8.52945983e-01 1.42157664e-04 4.73385020e-02
```

BASE PREDICTIORS

1- SVM

In [516]:

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

In [517]:

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

Out[517]:

```
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 [518]:

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

Out[518]:

```
55.96085409252669
```

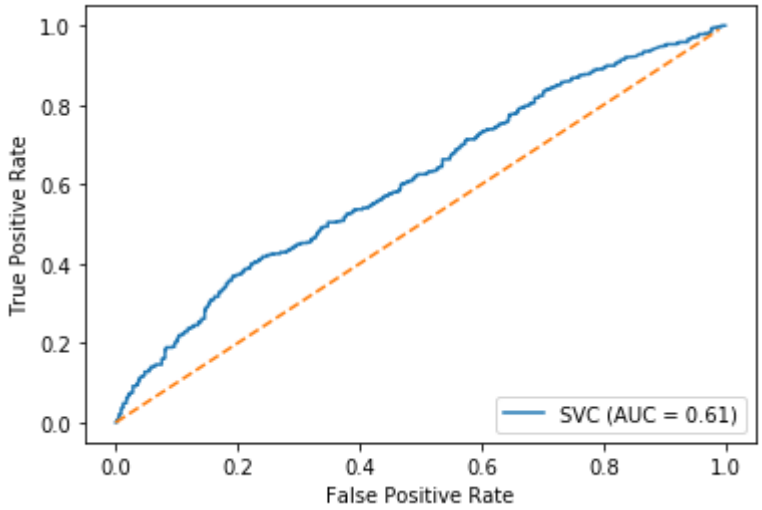
In [519]:

```
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:  55.96085409252669
Probability of detection of defect(Recall, pd):  0.5513698630136986
Probability of false alarm(pf):  0.43148148148148147
Probability of correct detection(Precision):  0.5801801801801801


F1-score or FM:  0.5654082528533801
AUC value:  0.5598616190883325


   P    N
[[307 262]
 [233 322]]


              precision    recall  f1-score   support

         0.0       0.57      0.54      0.55       569
         1.0       0.55      0.58      0.57       555

    accuracy                           0.56      1124
   macro avg       0.56      0.56      0.56      1124
weighted avg       0.56      0.56      0.56      1124
```

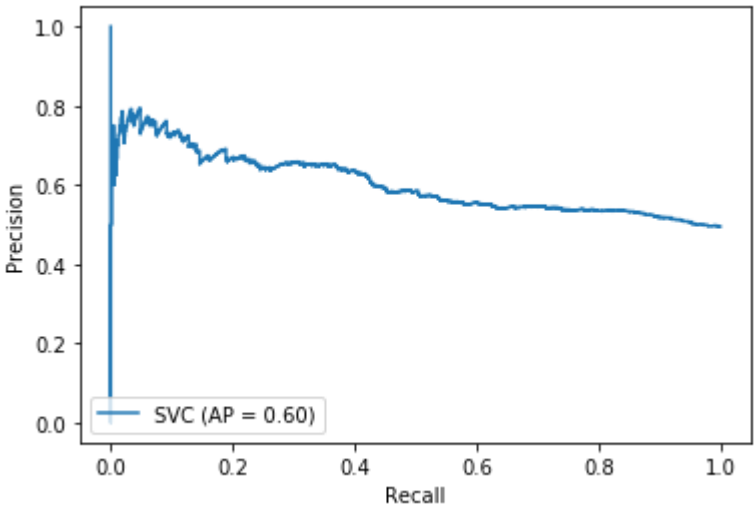
In [520]:

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

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



In [521]:

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



2- KNN

In [522]:

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=11)
```

In [523]:

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

Out[523]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=11, p=2,
                     weights='uniform')
```

In [524]:

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

Out[524]:

```
57.740213523131665
```

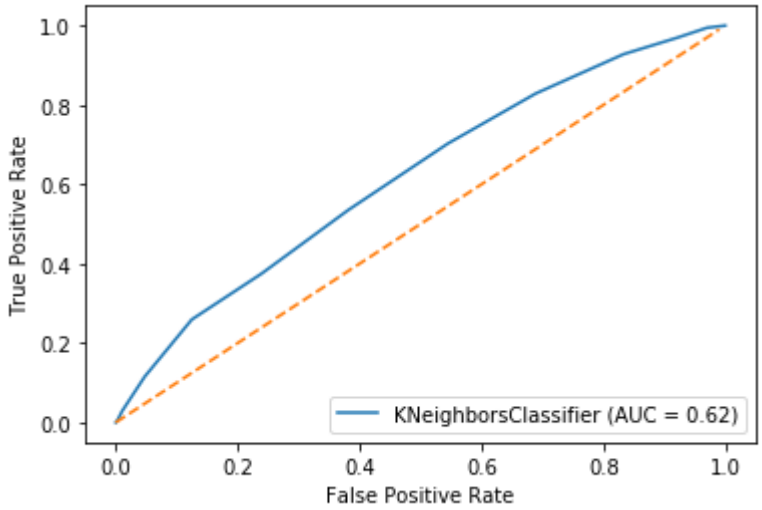
In [525]:

```
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:  57.740213523131665
Probability of detection of defect(Recall, pd):  0.5778210116731517
Probability of false alarm(pf):  0.42295081967213116
Probability of correct detection(Precision):  0.5351351351351351


F1-score or FM:  0.5556594948550045
AUC value:  0.5768821545622951


[[352 217]
 [258 297]]


              precision    recall  f1-score   support

         0.0       0.58      0.62      0.60       569
         1.0       0.58      0.54      0.56       555

    accuracy                           0.58      1124
   macro avg       0.58      0.58      0.58      1124
weighted avg       0.58      0.58      0.58      1124
```

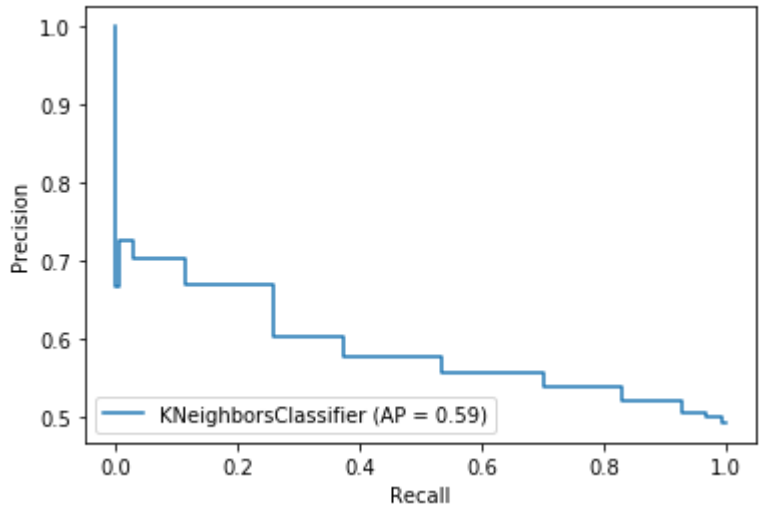
In [526]:

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

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



In [527]:

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



In [528]:

```
# 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)
```

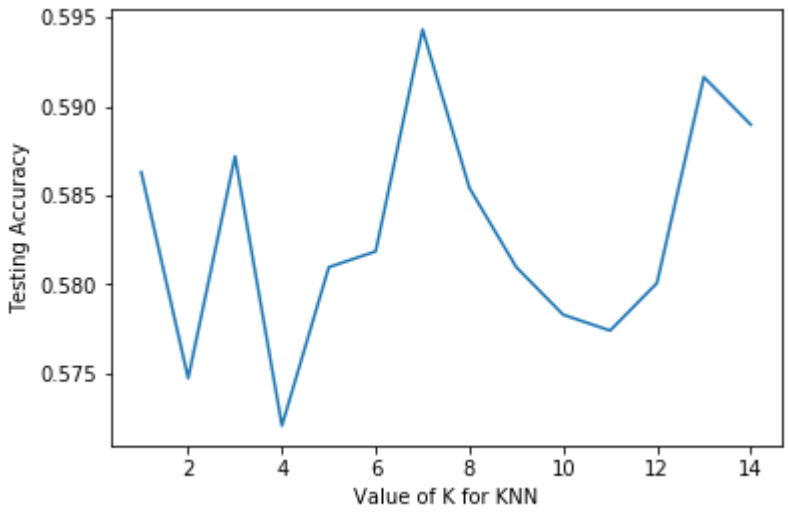
```
[0.5862989323843416, 0.5747330960854092, 0.5871886120996441, 0.5720640569395
018, 0.5809608540925267, 0.5818505338078291, 0.594306049822064, 0.5854092526
690391, 0.5809608540925267, 0.5782918149466192, 0.5774021352313167, 0.580071
1743772242, 0.5916370106761566, 0.5889679715302492]
```

In [529]:

```
# 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[529]:

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



3- NAIVE BAYES

In [530]:

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

In [531]:

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

Out[531]:

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

In [532]:

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

Out[532]:

```
53.91459074733096
```

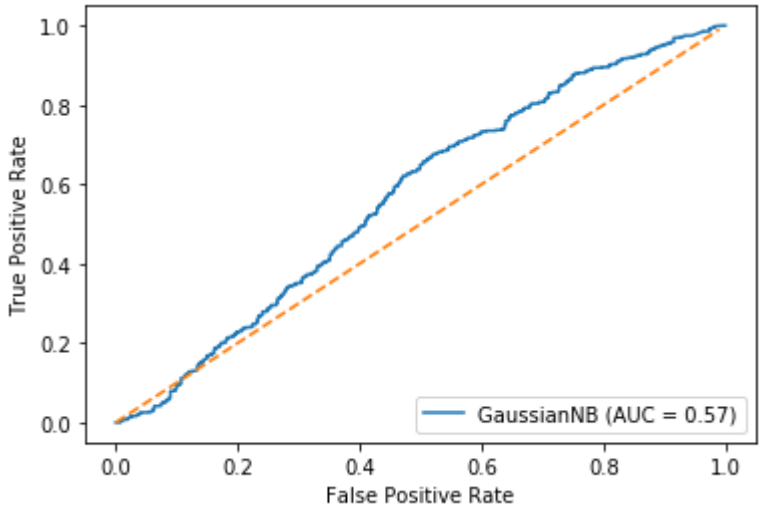
In [533]:

```
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:  53.91459074733096
Probability of detection of defect(Recall, pd):  0.5190133607399794
Probability of false alarm(pf):  0.33112582781456956
Probability of correct detection(Precision):  0.9099099099099099


F1-score or FM:  0.6609947643979057
AUC value:  0.5437071517915103


[[101 468]
 [ 50 505]]


              precision    recall  f1-score   support

         0.0       0.67      0.18      0.28       569
         1.0       0.52      0.91      0.66       555

    accuracy                           0.54      1124
   macro avg       0.59      0.54      0.47      1124
weighted avg       0.59      0.54      0.47      1124
```

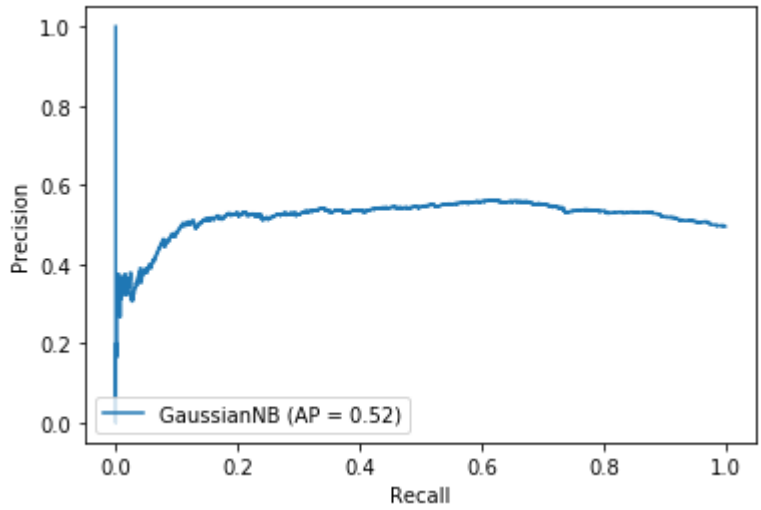
In [534]:

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

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



In [535]:

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



In [536]:

```
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 1124 points : 518
```

4- LOGISTIC REGRESSION

In [537]:

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

In [538]:

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

Out[538]:

```
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 [539]:

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

Out[539]:

```
57.38434163701067
```

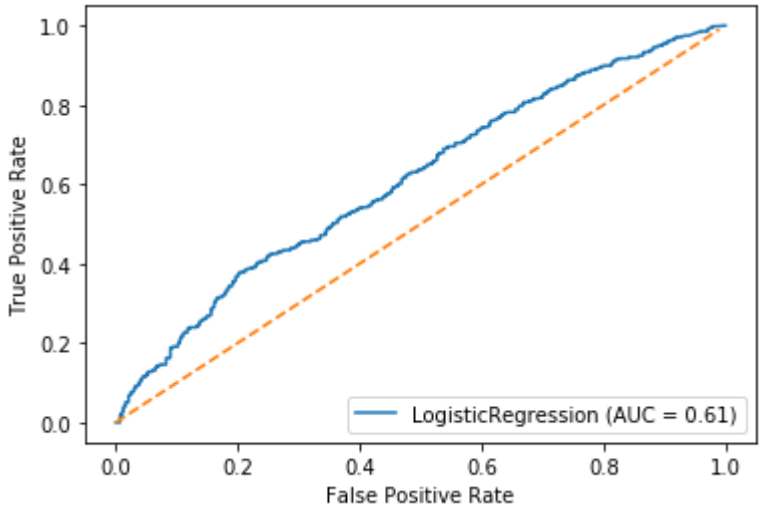
In [540]:

```
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:  57.38434163701067
Probability of detection of defect(Recall, pd):  0.5883720930232558
Probability of false alarm(pf):  0.43515850144092216
Probability of correct detection(Precision):  0.45585585585585586


F1-score or FM:  0.5137055837563452
AUC value:  0.57239189980842


[[392 177]
 [302 253]]


              precision    recall  f1-score   support

         0.0       0.56      0.69      0.62       569
         1.0       0.59      0.46      0.51       555

    accuracy                           0.57      1124
   macro avg       0.58      0.57      0.57      1124
weighted avg       0.58      0.57      0.57      1124
```

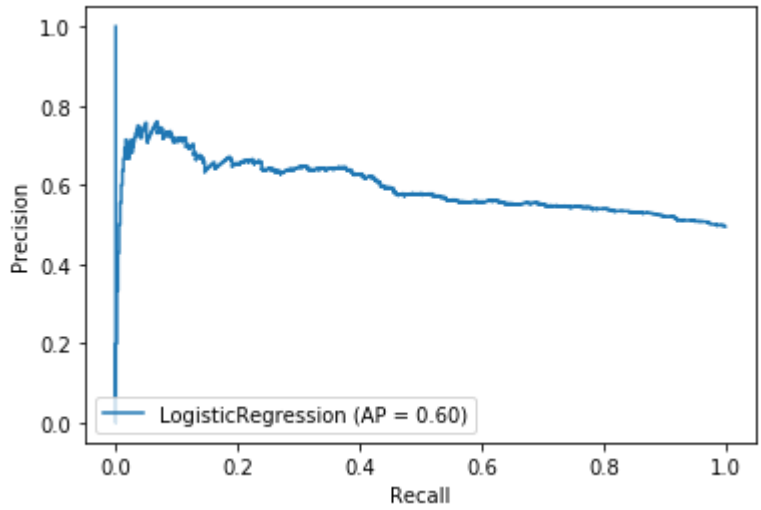
In [541]:

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

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



In [542]:

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



5- MLP

In [543]:

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

In [544]:

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

Out[544]:

```
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 [545]:

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

Out[545]:

```
58.89679715302491
```

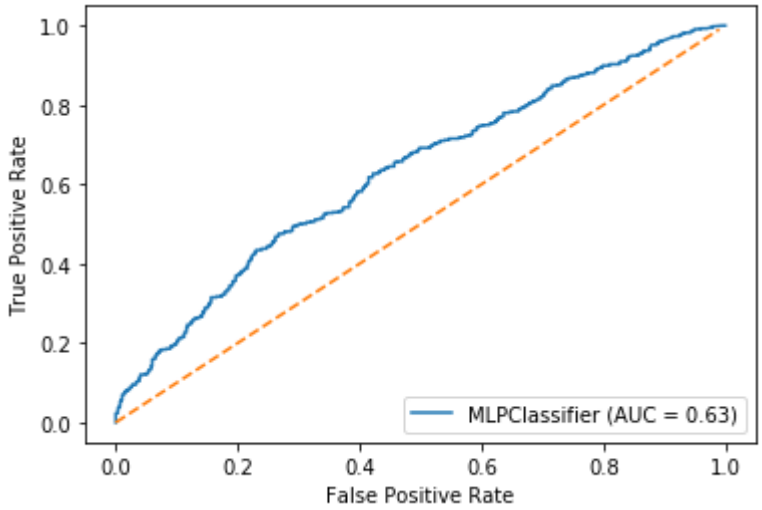
In [546]:

```
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:  58.89679715302491
Probability of detection of defect(Recall, pd):  0.5970772442588727
Probability of false alarm(pf):  0.4170542635658915
Probability of correct detection(Precision):  0.5153153153153153


F1-score or FM:  0.5531914893617021
AUC value:  0.5880618755838439


[[376 193]
 [269 286]]


              precision    recall  f1-score   support

         0.0       0.58      0.66      0.62       569
         1.0       0.60      0.52      0.55       555

    accuracy                           0.59      1124
   macro avg       0.59      0.59      0.59      1124
weighted avg       0.59      0.59      0.59      1124
```

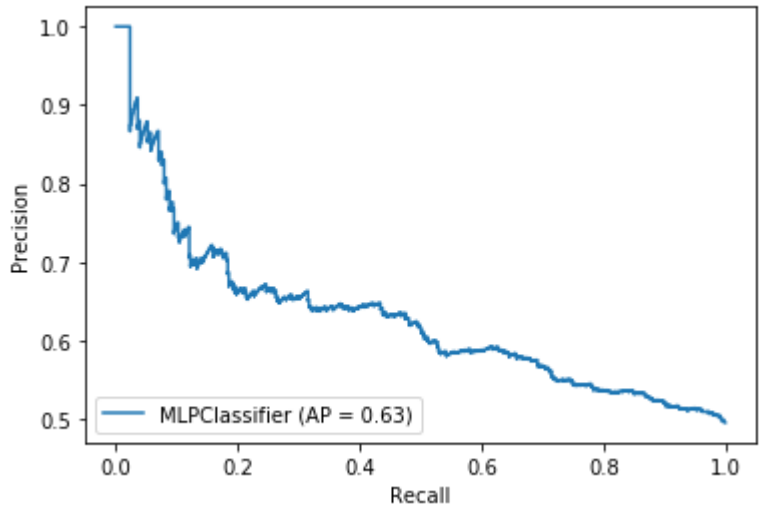
In [547]:

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

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



In [548]:

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



6- DECISION TREE

In [549]:

```
from sklearn import tree
```

In [550]:

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

Out[550]:

```
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 [551]:

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

Out[551]:

```
59.60854092526691
```

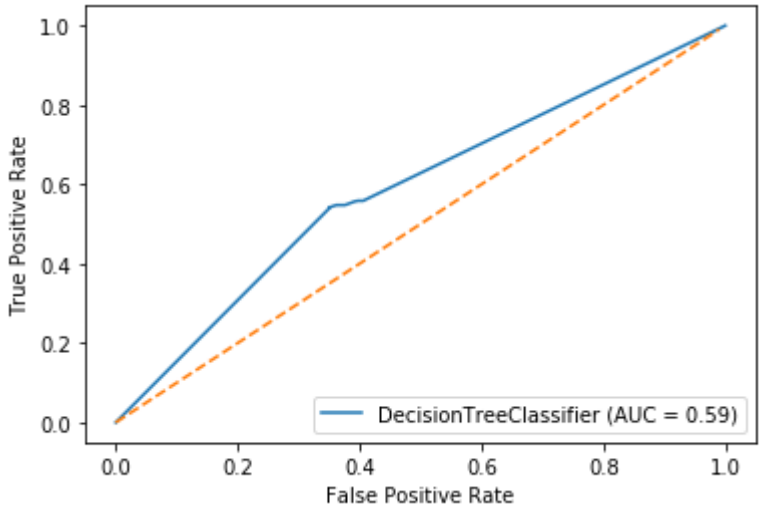
In [552]:

```
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:  59.60854092526691
Probability of detection of defect(Recall, pd):  0.6007984031936128
Probability of false alarm(pf):  0.40770465489566615
Probability of correct detection(Precision):  0.5423423423423424


F1-score or FM:  0.5700757575757577
AUC value:  0.5954242467423487


[[369 200]
 [254 301]]


              precision    recall  f1-score   support

         0.0       0.59      0.65      0.62       569
         1.0       0.60      0.54      0.57       555

    accuracy                           0.60      1124
   macro avg       0.60      0.60      0.59      1124
weighted avg       0.60      0.60      0.59      1124
```

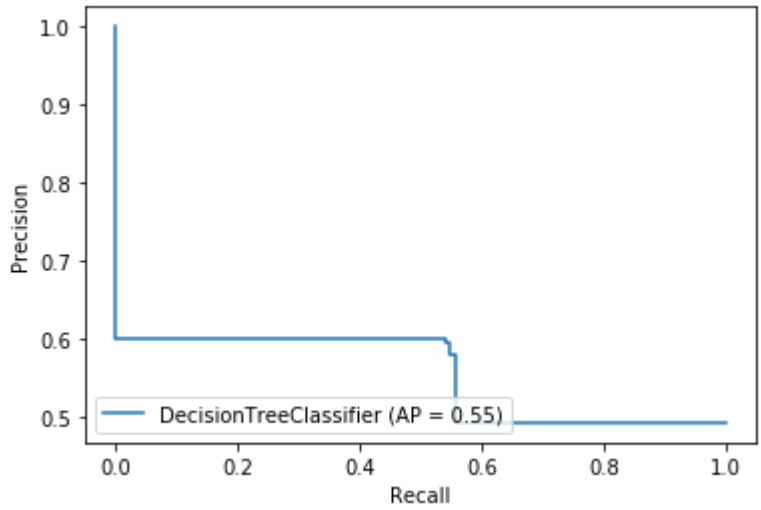
In [553]:

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

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



In [554]:

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



ENSEMBLE PREDICTORS

1- ADABOOST

In [555]:

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

In [556]:

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

Out[556]:

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

In [557]:

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

Out[557]:

```
60.76512455516014
```

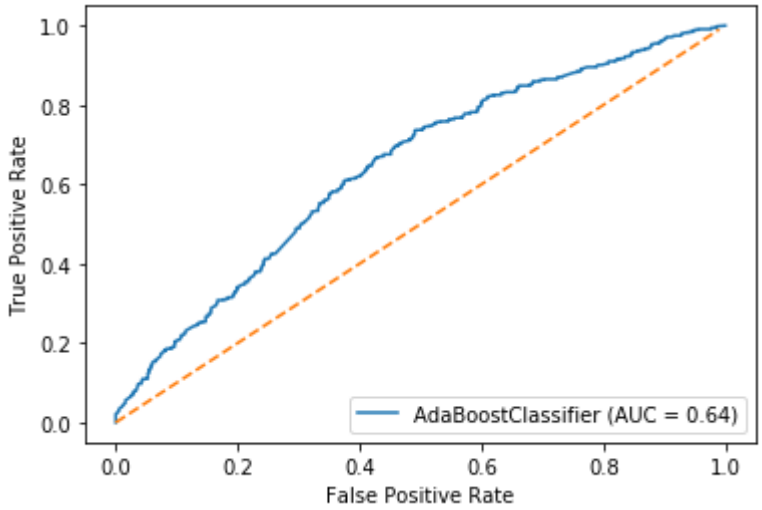
In [558]:

```
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:  60.76512455516014
Probability of detection of defect(Recall, pd):  0.6135458167330677
Probability of false alarm(pf):  0.3971061093247588
Probability of correct detection(Precision):  0.554954954954955


F1-score or FM:  0.5827814569536424
AUC value:  0.6070029607815197


[[375 194]
 [247 308]]


              precision    recall  f1-score   support

         0.0       0.60      0.66      0.63       569
         1.0       0.61      0.55      0.58       555

    accuracy                           0.61      1124
   macro avg       0.61      0.61      0.61      1124
weighted avg       0.61      0.61      0.61      1124
```

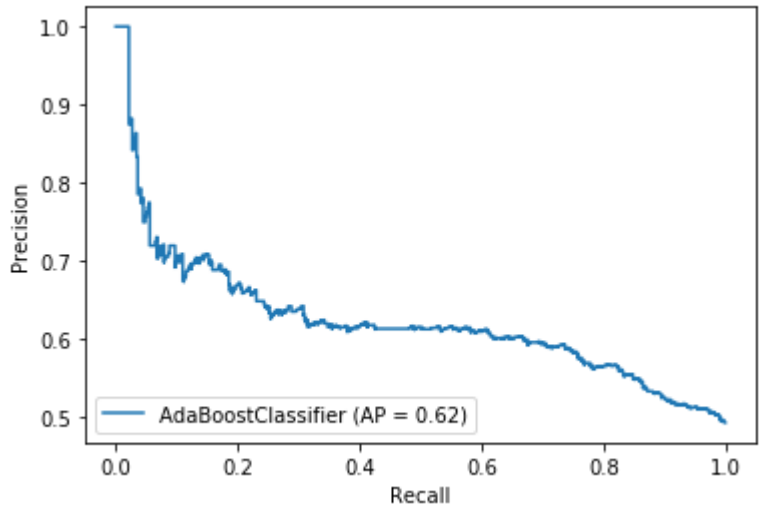
In [559]:

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

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



In [560]:

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



2- BAGGING

In [561]:

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

In [562]:

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

Out[562]:

```
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 [563]:

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

Out[563]:

```
63.25622775800712
```

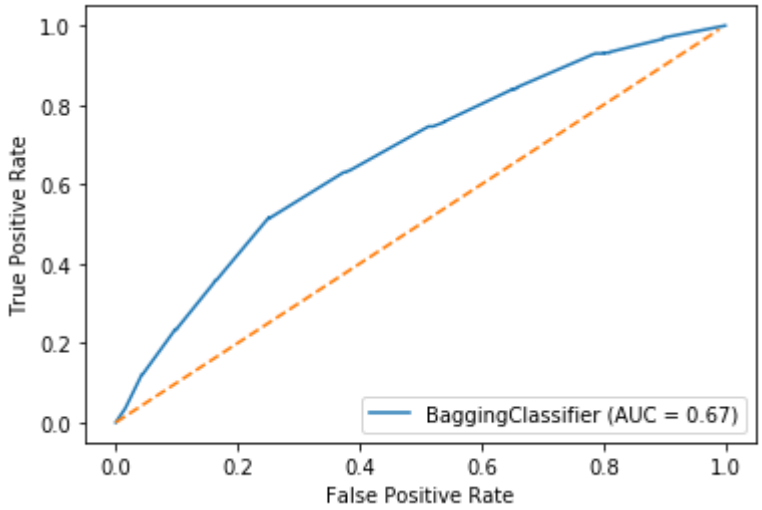
In [564]:

```
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:  63.25622775800712
Probability of detection of defect(Recall, pd):  0.6643518518518519
Probability of false alarm(pf):  0.3872832369942196
Probability of correct detection(Precision):  0.5171171171171172


F1-score or FM:  0.5815602836879433
AUC value:  0.6311420383476622


[[424 145]
 [268 287]]


              precision    recall  f1-score   support

         0.0       0.61      0.75      0.67       569
         1.0       0.66      0.52      0.58       555

    accuracy                           0.63      1124
   macro avg       0.64      0.63      0.63      1124
weighted avg       0.64      0.63      0.63      1124
```

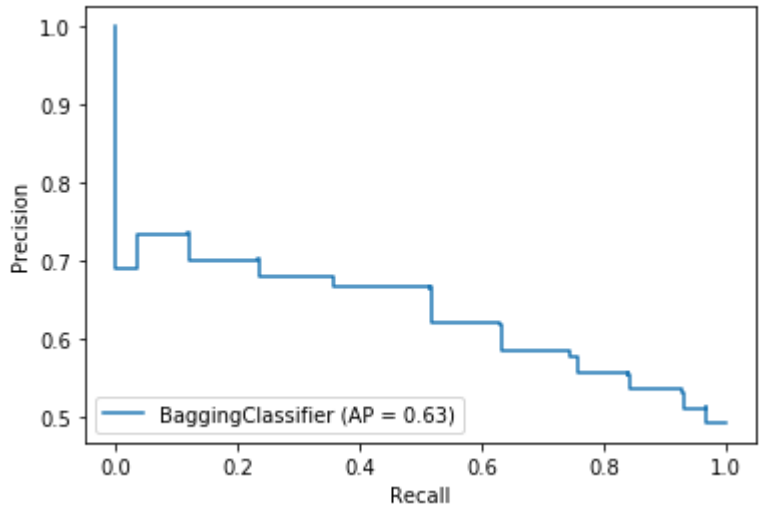
In [565]:

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

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



In [566]:

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



3- Extra_Tree_Classifier

In [567]:

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

In [568]:

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

Out[568]:

```
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 [569]:

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

Out[569]:

```
63.612099644128115
```

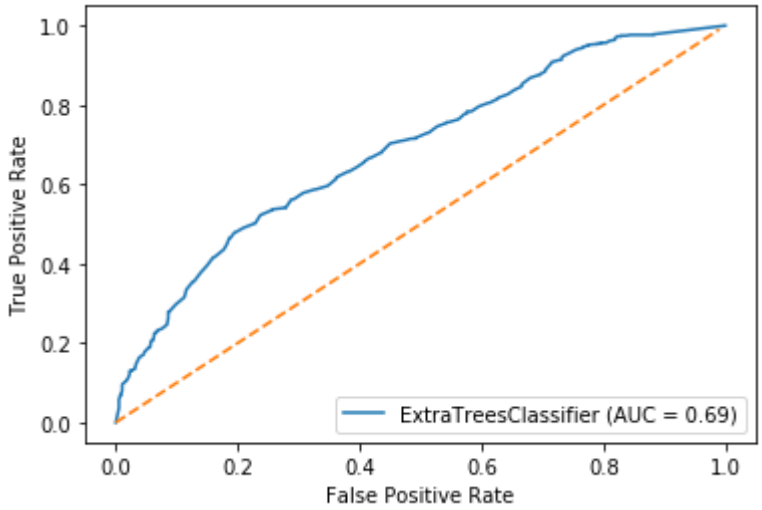
In [570]:

```
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:  63.612099644128115
Probability of detection of defect(Recall, pd):  0.6527196652719666
Probability of false alarm(pf):  0.3761609907120743
Probability of correct detection(Precision):  0.5621621621621622


F1-score or FM:  0.6040658276863504
AUC value:  0.6352111338051584


[[403 166]
 [243 312]]


              precision    recall  f1-score   support

         0.0       0.62      0.71      0.66       569
         1.0       0.65      0.56      0.60       555

    accuracy                           0.64      1124
   macro avg       0.64      0.64      0.63      1124
weighted avg       0.64      0.64      0.63      1124
```

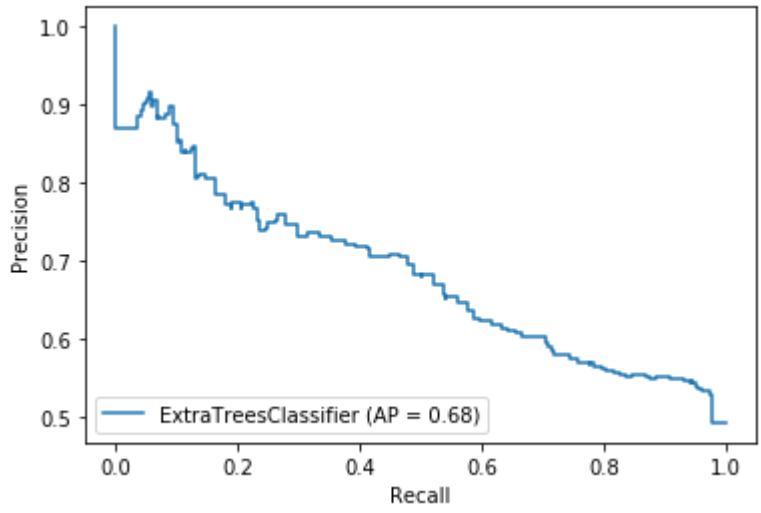
In [571]:

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

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



In [572]:

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



4- Gradient_Boosting_Classifier

In [573]:

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

In [574]:

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

Out[574]:

```
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 [575]:

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

Out[575]:

```
61.47686832740214
```

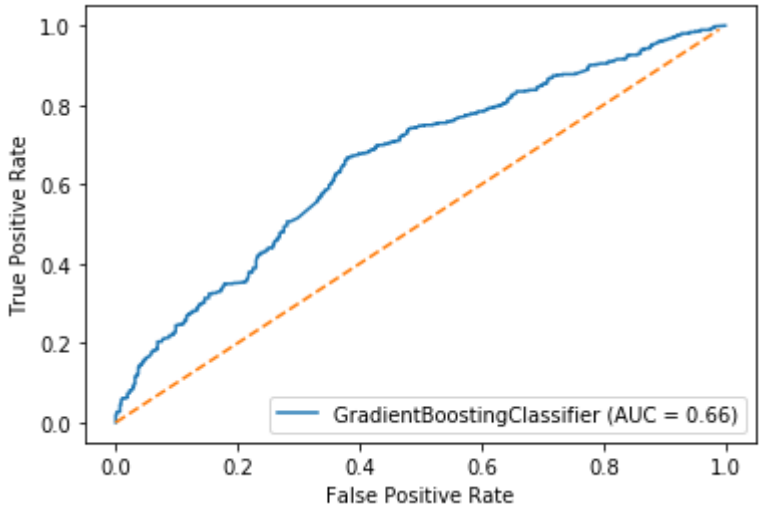
In [576]:

```
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:  61.47686832740214
Probability of detection of defect(Recall, pd):  0.6224899598393574
Probability of false alarm(pf):  0.3913738019169329
Probability of correct detection(Precision):  0.5585585585585585


F1-score or FM:  0.5887939221272555
AUC value:  0.6140771703161861


[[381 188]
 [245 310]]


              precision    recall  f1-score   support

         0.0       0.61      0.67      0.64       569
         1.0       0.62      0.56      0.59       555

    accuracy                           0.61      1124
   macro avg       0.62      0.61      0.61      1124
weighted avg       0.62      0.61      0.61      1124
```

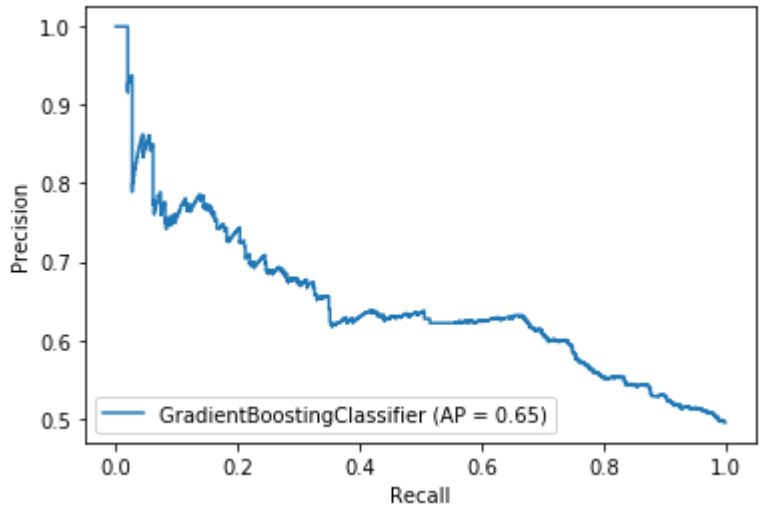
In [577]:

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

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



In [578]:

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



5- Random_Forest_Classifier

In [579]:

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

In [580]:

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

Out[580]:

```
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 [581]:

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

Out[581]:

```
63.25622775800712
```

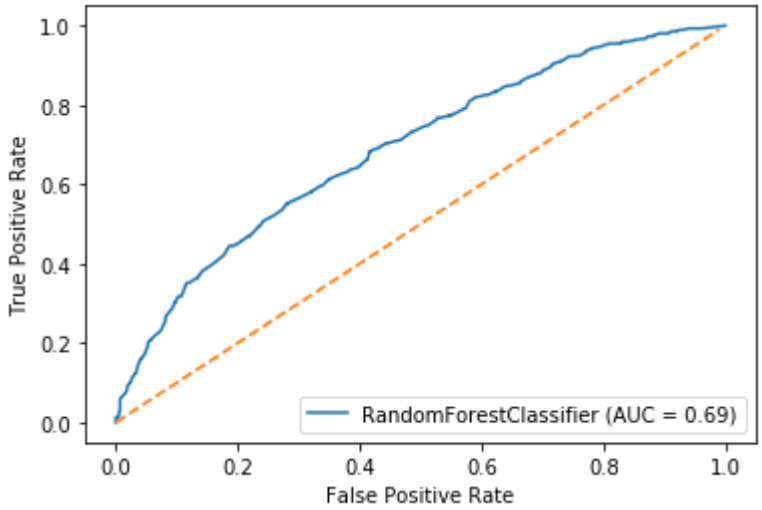
In [582]:

```
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:  63.25622775800712
Probability of detection of defect(Recall, pd):  0.6448979591836734
Probability of false alarm(pf):  0.37697160883280756
Probability of correct detection(Precision):  0.5693693693693693


F1-score or FM:  0.6047846889952153
AUC value:  0.6317848604316091


[[395 174]
 [239 316]]


              precision    recall  f1-score   support

         0.0       0.62      0.69      0.66       569
         1.0       0.64      0.57      0.60       555

    accuracy                           0.63      1124
   macro avg       0.63      0.63      0.63      1124
weighted avg       0.63      0.63      0.63      1124
```

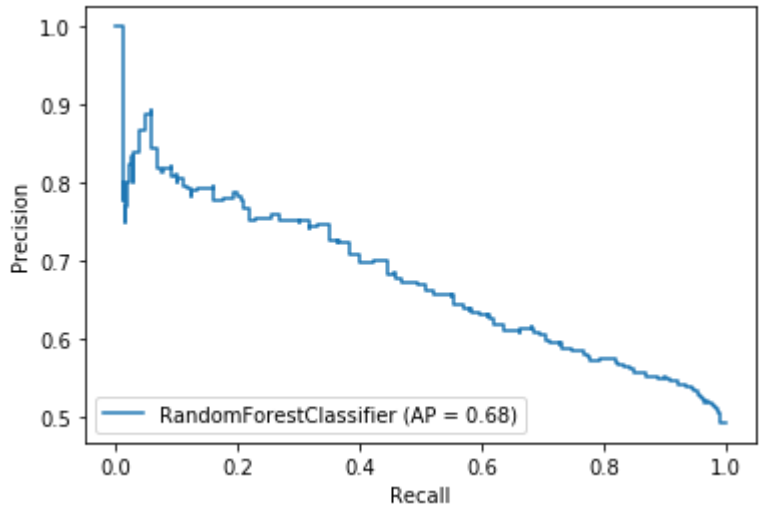
In [583]:

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

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



In [584]:

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



6- Stacking_Classifier

In [585]:

```
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 [586]:

```
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)
```

```
C:\ProgramData\Anaconda3\envs\myenv\lib\site-packages\sklearn\svm\_base.py:9
47: ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
C:\ProgramData\Anaconda3\envs\myenv\lib\site-packages\sklearn\svm\_base.py:9
47: ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
C:\ProgramData\Anaconda3\envs\myenv\lib\site-packages\sklearn\svm\_base.py:9
47: ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
C:\ProgramData\Anaconda3\envs\myenv\lib\site-packages\sklearn\svm\_base.py:9
47: ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
C:\ProgramData\Anaconda3\envs\myenv\lib\site-packages\sklearn\svm\_base.py:9
47: ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
C:\ProgramData\Anaconda3\envs\myenv\lib\site-packages\sklearn\svm\_base.py:9
47: ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
```

Out[586]:

```
StackingClassifier(cv=None,
                   estimators=[('rf',
                                RandomForestClassifier(bootstrap=True,
                                                       ccp_alpha=0.0,
                                                       class_weight=None,
                                                       criterion='gini',
                                                       max_depth=None,
                                                       max_features='aut
o',
                                                       max_leaf_nodes=Non
e,
                                                       max_samples=None,
                                                       min_impurity_decrea
se=0.0,
                                                       min_impurity_split=
None,
                                                       min_samples_leaf=1,
                                                       min_samples_split=
2,
                                                       min_weight_fraction
_leaf=0.0,
                                                       n_estimators=10,
                                                       n_jobs=None,...
```

```
                                                    tol=0.0001,
                                                    verbose=0))],
                                    verbose=False))],
                    final_estimator=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),
                    n_jobs=None, passthrough=False, stack_method='auto',
                    verbose=0)
```

In [587]:

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

Out[587]:

```
64.05693950177937
```

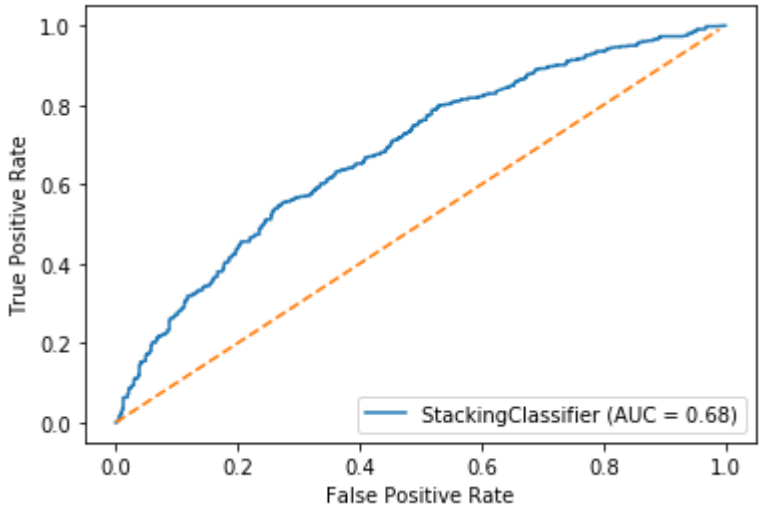
In [588]:

```
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:  64.05693950177937
Probability of detection of defect(Recall, pd):  0.6666666666666666
Probability of false alarm(pf):  0.3770491803278688
Probability of correct detection(Precision):  0.5441441441441441


F1-score or FM:  0.5992063492063492
AUC value:  0.6393831441283111


[[418 151]
 [253 302]]


              precision    recall  f1-score   support

         0.0       0.62      0.73      0.67       569
         1.0       0.67      0.54      0.60       555

    accuracy                           0.64      1124
   macro avg       0.64      0.64      0.64      1124
weighted avg       0.64      0.64      0.64      1124
```

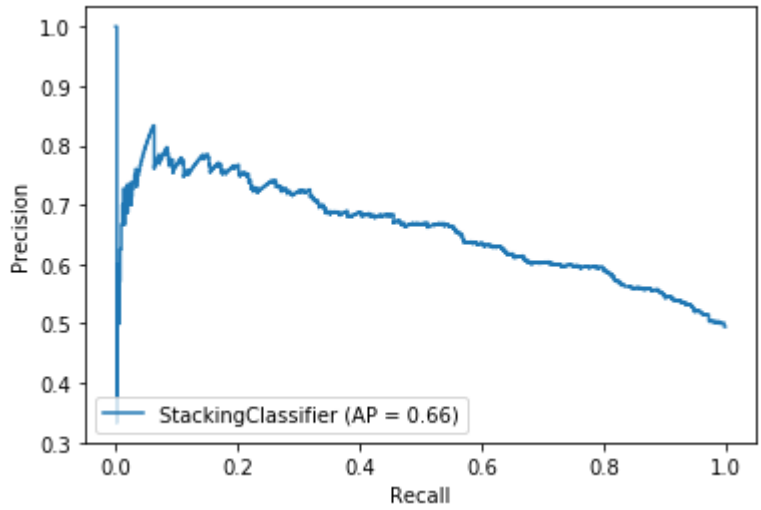
In [589]:

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

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



In [590]:

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



7- Voting_Classifier

In [591]:

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

In [592]:

```
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[592]:

```
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 [593]:

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

Out[593]:

```
61.56583629893239
```

In [594]:

```
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:  61.56583629893239
Probability of detection of defect(Recall, pd):  0.6016528925619835
Probability of false alarm(pf):  0.36801541425818884
Probability of correct detection(Precision):  0.6558558558558558


F1-score or FM:  0.6275862068965518
AUC value:  0.6161528839911968


[[328 241]
 [191 364]]


              precision    recall  f1-score   support

         0.0       0.63      0.58      0.60       569
         1.0       0.60      0.66      0.63       555

    accuracy                           0.62      1124
   macro avg       0.62      0.62      0.62      1124
weighted avg       0.62      0.62      0.62      1124
```

In [ ]:

In [ ]: