Comparison of various base predictors and ensemble predictors in software fault prediction. Exploring new techniques to improve predictions.

Shashank Shukla
School of Computer Science(Scope)
Vellore Institute of Technology
Vellore, India
shashank.shukla2018@vitstudent.ac.in

Mihir Agarwal
School of Computer Science(Scope)
Vellore Institute of Technology
Vellore, India
mihir.agarwal2018@vitstudent.ac.in

Harsh Pandey
School of Computer Science(Scope)
Vellore Institute of Technology
Vellore, India
harsh.pandey2018@vitstudent.ac.in

Abstract—**Software development projects require a software testing phase, a critical and costly step, to investigate efficiency of the resultant product. In this aspect, the required time to test the software, the staff to be allocated and hence the cost are the major points to consider. As the size and complexity of project increases, manual prediction of software defects becomes a time consuming and costly task. As an alternative to manual code reviews, defect predictors have been widely used in organizations to predict software bugs while saving time and reducing cost. There are various ensemble methods used to increase precision, accuracy in software fault prediction. Various ensemble classifiers shown positive improvement towards accuracy, precision and reduce in false alarm rates. There are various techniques like bagging, boosting to ensemble base predictors and increase their prediction accuracy. Many of techniques are unexplored or not used in software fault predictions. Combination of ensemble predictors with feature selection techniques to further evaluate fault prediction performance of ensemble strategies is also to be explored.**

***Index Terms*—Ensemble, prediction, classifier**

I. Introduction

The complexity of software has grown considerably in recent years, making it nearly impossible to detect all faults before pushing to production. Such faults can ultimately lead to failures at run time. Software fault prediction is the process of developing models that can be used by the software practitioners in the early phases of software development life cycle for detecting faulty constructs such as modules or classes. In case of a limited budget, instead of a complete testing of an entire system, software managers might use defect predictors to focus on parts of the system that seem defect-prone. These potential defect prone modules can then be examined with more detail.

In software development, prediction of fault-prone modules is an important challenge for effective software testing. However, high prediction accuracy may not be achieved in cross-project prediction, since there is a large difference in distribution of predictor variables between the base project (for building prediction model) and the target project (for applying prediction model). Recent works have shown that using Machine Learning (ML) algorithms it is possible to create models that can accurately predict such failures. Many researchers focused on classification algorithm for predicting the software defect. On the other hand, classifiers ensemble can effectively improve classification performance when compared with a single classifier.

At the same time, methods that combine several independent learners (i.e., ensembles) have proved to outperform individual models in various problems. While some well-known ensemble algorithms (e.g Bagging) use the same base learners (i.e., homogeneous), using different algorithms (i.e., heterogeneous) may exploit the different biases of each algorithm. To get a highly reliable software product to the market on schedule, software engineers must allocate resources on the fault-prone software modules across the development effort. Software quality models based upon data mining from past projects can identify fault-prone modules in current similar development efforts. So that resources can be focused on fault-prone modules to improve quality prior to release.

From machine learning perspective, defect prediction is a classification task that aims to discriminate fault prone (fp) and non-fault prone (nfp) code modules defined with static code attributes. Thus, there are many candidate classification algorithms that might help software analysts to investigate whether a software component will be predicted fp or nfp. There are many studies aiming to locate software defects with the use of single learners.
However, in most of the literature, we did not find any comprehensive study evaluating ensemble predictor algorithms for software quality estimation. There are only a few studies that make use of ensemble predictors applied to software fault detection. From the software defect prediction point of view, ensemble algorithms combine signals from base defect predictors in the committee to produce an enhanced fault detection algorithm. Furthermore, the committee of predictors (ensemble) provides capturing the strengths of multiple predictors while helping to overcome noisy nature of code metrics that define software modules to be tested.

For the comparison of various base predictors and ensemble predictors in software fault prediction, the algorithms used are as follows-

[1] Bayes Optimal Classifier: -
The Bayes Optimal Classifier is a probability model that makes the most probable prediction for a new example. This model is described using the Bayes Theorem, which provides a principled way for calculating conditional probability. The conditional probability of one outcome given another outcome, using the inverse of this relationship, is calculated as-

$$P(A|B)=(P(B|A)*P(A))/P(B) \text{ where}$$

_P (A|B) = Probability of A given B
_P (A) = Prior Probability of A

[2] Bayesian Model Averaging: -
Bayesian model averaging (BMA) provides a coherent and systematic mechanism for accounting for model uncertainty. BMA produces a straightforward model choice criterion and less risky predictions.

[3] Bayesian Model Combination: -
Bayesian model combination (BMC) is an algorithmic correction to Bayesian model averaging (BMA). Instead of sampling each model in the ensemble individually, as done in Bayesian Model Averaging, it samples from the space of possible ensembles (with model weightings drawn randomly from a Dirichlet distribution having uniform parameters). This modification overcomes the tendency of BMA to converge toward giving all of the weight to a single model. Although BMC is somewhat more computationally expensive than BMA, it tends to yield dramatically better results. The results from BMC have been shown to be better on average (with statistical significance) than BMA, and bagging.

[4] Bucket of Models: -
A "bucket of models" is an ensemble technique in which a model selection algorithm is used to choose the best model for each problem. When tested with only one problem, a bucket of models can produce no better results than the best model in the set, but when evaluated across many problems, it will typically produce much better results, on average, than any model in the set. The most common approach used for model-selection is cross-validation selection (sometimes called a "bake-off contest"). It is described with the following pseudo-code:

[5] Stacking: -
Stacking, also known as stacked generalization, involves training a learning algorithm to combine the predictions of several other learning algorithms. First, all of the other algorithms are trained using the available data, then a combiner algorithm is trained to make a final prediction using all the predictions of the other algorithms as additional inputs. If an arbitrary combiner algorithm is used, then stacking can theoretically represent any of the ensemble techniques described in this article, although, in practice, a logistic regression model is often used as the combiner. Stacking typically yields performance better than any single one of the trained models.

II. Literature Survey

[1] Ayse at al (2011)- In this paper, three machine learning techniques or classifiers are ensembled to get high precision. They build a learning-based defect predictor which would produce high detection rates as well as low false alarm rates. Rather than relying on the outcome of a single classifiifier, the ensemble approach combines signals from multiple classifiifiers to produce an outcome, and it manages to capture the strengths of uncommon techniques as well as helping to overcome problems of noise in data. They build two classifiers, in which one is emsembled and compared in terms of classifification accuracy and cost-benefifit analysis.

[2] **Satoshi et al (2012)**- In this paper they propose an prediction technique called "an ensemble of simple regression models" to improve the prediction accuracy of cross-project prediction. The proposed method uses weighted sum of outputs of simple (e.g. 1-predictor variable) logistic regression models to improve the generalization ability of logistic models. To evaluate the performance of the proposed method, we conducted 132 combinations of cross-project prediction using datasets of 12 projects from NASA IV&V Facility Metrics Data Program. As a result, the proposed method outperformed conventional logistic regression models in terms of AUC of the Alberg diagram.

[3] **Shanthini & Chandrasekaran (2014)**- This paper mainly addresses using ensemble approach of Support Vector Machine (SVM) for fault prediction. Ensemble classifier was examined for Eclipse Package level dataset and NASA KC1 dataset. They showed that proposed ensemble of Support

Vector Machine is superior to individual approach for software fault prediction in terms of classification rate through Root Mean Square Error Rate (RMSE), AUC-ROC, ROC curves.

[4] Santosh & Sandeep (2017): - In this paper, it discusses about the several classification techniques, which have been investigated & evaluated earlier for the software fault prediction, and they have produced different prediction accuracy for different software systems, by discussing the inefficiencies or doubts about one model being better than the other, or whether the classified software modules are faulty or not, which has been done & verified by conducting experimental study of five open-source software projects, and the results obtained are evaluated through intra-release and inter-release prediction.

[5] Santosh & Sandeep (2017): - In this paper, the different techniques used for Software fault predictions using different techniques have been discussed, which have been discussed by various researchers, as stated previously. It has been done so by observing the performances of the Software fault prediction techniques, through which the results showed that the techniques varied from dataset to dataset. So, in order to remove this inaccuracy, the heterogeneous ensemble method for software fault prediction has been implemented/presented for predicting the number of faults in software modules.

[6] Ruchika & Megha (2018): -In this paper, an ensemble method has been studied, which can be used to improve the prediction performance of individual classifiers. For achieving this, four techniques/strategies of ensemble learning has been studied for predicting change prone classes by combining seven individual PSO (Particle Swarm Optimization) based classifiers as constituents of ensembles & aggregating the classifiers using weighted voting. After tests/studies of accuracy and ability to correctly predict hard instances, the analysis of the results, which were obtained through prediction using different fitness functions, showed improved performance of the processed ensemble classifiers.

[7] R. Jothi(June, 2018)- In this paper Kmeans clustering is applied along with its variants. 5 software fault predictions datasets have been taken from PROMISE repository. Results show that proper initial seeding of data is required for effective results.

[8] **JOAO et al (2019)**- This paper presents a case study on using several ML techniques to create heterogeneous ensembles for Online Failure Prediction (OFP). More precisely, it attempts to assess the viability of combining different learners to improve performance and to understand how different combination techniques influence the results. The paper also explores whether the interactions between learners can be studied and leveraged. The results suggest that the combination of certain learners and techniques, not necessarily individually the best, can improve the overall ability to predict failures.

[9] Nguyen et a l(2019)- The main focus of this paper is on fault predictions based on metrics, like measurable properties for identifying error prone parts, and predictive models were built by using historical bug data from similar projects.The paper aims at analyzing the performance of different machine learning algorithms like - Naïve Bayes Decision Tree, Random Forest, Logistic Regression, K-nearest neighbors, Multilayer Perceptron, and SVM.Several datasets like JM1, PC4, KC2, MC1 and PC2 are used to analyze all the algorithms.

[10] IYAD et al(2019)- In this paper, 15 real data sets have been taken and enhanced by using Adaptive synthetic sampling (ADASYN). Moth Flame Optimization(MFO) is converted from continuous to binary versions using transfer functions(TF's) from S-shape and V shape, 2 different groups. Binary Moth Flame Optimization(BMFO) is then optimised to Enhanced Binary Moth Optimization(EBMFO). And the result shows that EBMFO gives better result than k-nearest neighbors (k-NN), Decision Trees (DT), and Linear discriminant analysis (LDA).

[11] Hung et al(2019)- Feature redundancy and class imbalance problems are the 2 most significant problems of classification. Classification being the most popular category of software prediction, this paper focuses on its problems. GFsSDAEsTSE is a proposed deep learning model to improve the performance of software fault predictions. And the results show that the performance is better than state-of-the-art methods, on both small and large data sets.

[12] Fatih et al (2019)- In this paper research on the performance of ten ensemble predictors from WEKA machine learning workbench applied to fifteen datasets from PROMISE repository. In order to make a baseline (determining the best predictor) for evaluation of ensemble algorithms, they first selected sixteen widely used base predictors from software fault detection literature. They calculated performance of predictors for sample datasets in terms of F-measure (FM). As a second step, they tested ten ensemble predictors with the same datasets to determine their performance using FM and AUC metrics. Finally, they compared best ensemble predictors with baseline predictor of step one to interpret the performance of ensemble algorithms in software fault detection.

[13] **Hadeel & Marc (2020):** -In this paper, the software fault prediction has been discussed through Software Maintainability, which is one of the fundamental quality attributes of Software Engineering. Also, the challenge of the accurate prediction of software maintainability for the effective management of the software maintenance process has been discussed. After the study, research and analysis of ensemble & prediction models, the results showed that the ensemble method showed increased accuracy prediction over individual models, and have been useful models in predicting software maintainability.

III. Problem Statement

Faults in a module tend to cause failure of the software product. These defective modules in the software pose considerable risk by increasing the developing cost and decreasing the customer satisfaction. Hence in a software development life cycle it is very important to predict the faulty modules in the software product. Prediction of the defective modules should be done as early as possible so as to improve software developers' ability to identify the defect-prone modules and focus quality assurance activities such as testing and inspections on those defective modules.

IV. Methodology

Various base learners and algorithms-

Many machine algorithms are applied on dataset obtained from promise repository which contains data of software fault detection on softwares like JM1 by NASA. Algorithms like KNN, Neural Networks gave sufficient good amount of accuracy but it could be further improved by ensembling of algorithms. Ensemble learning, in general, is a model that makes predictions based on a number of different models. By combining individual models, the ensemble model tends to be more flexible. Two most popular ensemble methods are bagging and boosting. Bagging: Training a bunch of individual models in a parallel way. Each model is trained by a random subset of the data. Boosting: Training a bunch of individual models in a sequential way. Each individual model learns from mistakes made by the previous model.

Architecture of some base learners-

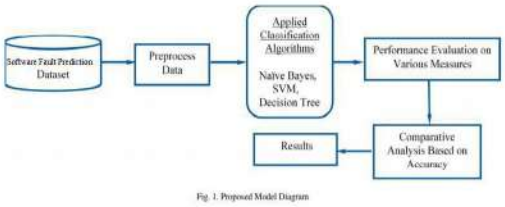
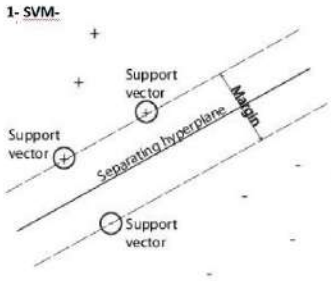
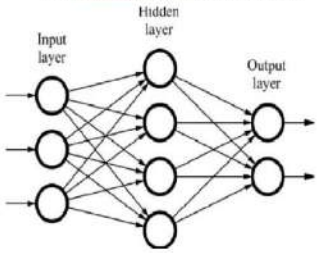


Fig. 1. Proposed Model Diagram

1- SVM-



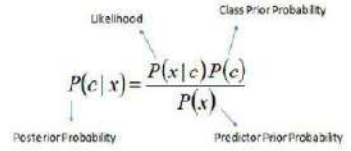
2- MLP(Multilayer Perceptron) or Neural Network-



3-Decision Tree-

4-KNN(K Nearest Neighbor)-

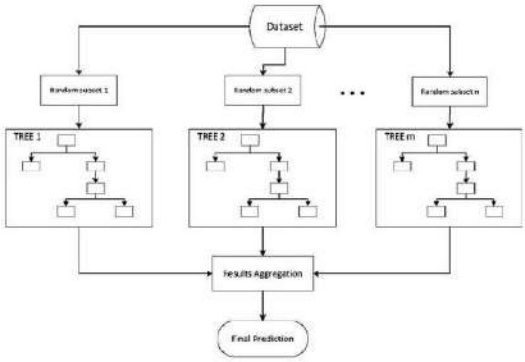
5- Naive Bayes-

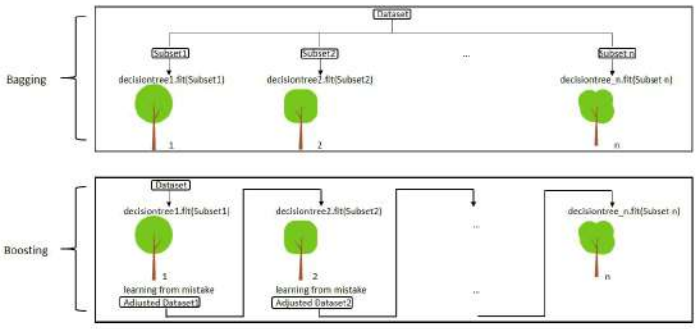


$$P(c|X) = P(x_1|c)\times P(x_2|c)\times\cdots\times P(x_n|c)\times P(c)$$

Architecture of some Ensemble methods-

Bagging(Random Forest)-





V. RELEVENT PSUEDOCODE

Input: *training set S, base predictor B, integer T (# of bootstrap samples)*

A. *for* $i = 1$ *to* T {

B. $S' =$ *bootstrap sample from* S

C. $E_i = B(S')$

D. }

E. $E^*(x) = \underset{y \in Y}{arg\ max} \sum_{i: E_i(x)=y} 1$ *(predicted label* y*)*

Output: *ensemble predictor* E^*

Fig. 2. Pseudocode of Bagging Algorithm.

Input: *training set* S *of size* m*, base predictor* B*, integer* T *(# of trials*

A. $S' = S$ *instance weights assigned to be* 1

B. *For* $i = 1$ *to* T {

C. $E_i = B(S')$

D. $\epsilon_i = \frac{1}{m} \sum_{x_j \in S': E_i(x_j) \neq y_j} weight(x)$ *(weighted error on training set)*

E. *If* $\epsilon_i > 1/2$*, set* S' *to bootstrap sample from* S *with weight* 1 *for every instance and GO TO step C*

F. $\beta_i = \epsilon_i / (1 - \epsilon_i)$

G. *For* $-$ *each* $x_j \epsilon S'$*, if* $E_i(x_j) = y_j$ *then* $weight(x_j) = weight(x_j).\beta_i$

H. *Normalize the weight of instances so that* S' *to be* m

I. }

J. $E^*(x) = \underset{y \in Y}{arg\ max} \sum_{i: E_i(x)=y} \log \frac{1}{\beta_i}$

Output: *ensemble predictor* E^*

Fig. 3. Pseudocode of Boosting Algorithm.

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()
```

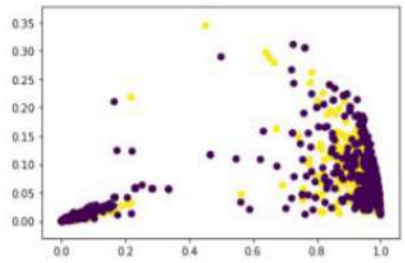
Out[507]:

	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	2.165066e-06	0.000876	0.002418
3	0.002552	0.000040	0.000013	0.000040	0.008059	0.058413	8.059231e-07	0.000226	0.003423
4	0.003581	0.000387	0.000097	0.000387	0.012195	0.057987	5.807237e-06	0.001684	0.003374

5 rows × 21 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
```

Results for JM1 software-
Best Base learning algorithm-

MLP(Multilayer Perceptron):
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
```

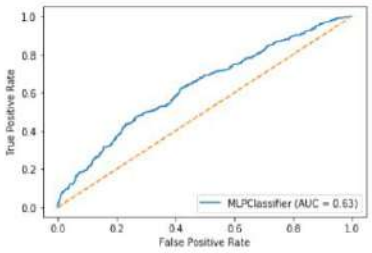
In [546]:

```
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
```

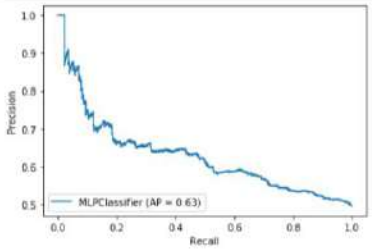
In [547]:

```
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()
```



Best Ensemble learning algorithm- **Bagging**:

2- BAGGING

In [561]:

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

In [562]:

```
bagmodel = BaggingClassifier(base_estimator=None, n_estimators=10) #default=decision tr
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
```

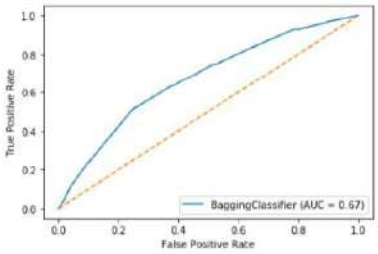
In [564]:

```
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
```

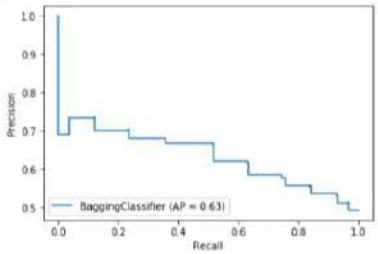
In [565]:

```
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()
```



VI. RESULT

Software Fault Prediction Performance of Base Predictors for FM value:

Dataset	NB	DT	LR	MLP	SVM	KNN
jm1	0.660	0.570	0.513	0.553	0.564	0.556
cm1	0.787	0.714	0.142	0.740	0.010	0.641
kc1	0.658	0.652	0.761	0.852	0.652	0.563
kc2	0.770	0.769	0.759	0.741	0.722	0.670
pc1	0.518	0.593	0.608	0.722	0.524	0.704

Software Fault Prediction Performance of Ensemble Predictors for FM value:

Dataset	AB	BG	ETC	GBC	RF	SC	VC
jm1	0.582	0.581	0.604	0.588	0.604	0.599	0.62
cm1	0.879	0.615	0.846	0.812	0.750	0.896	0.80
kc1	0.858	0.752	0.761	0.852	0.712	0.812	0.72
kc2	0.799	0.765	0.801	0.741	0.812	0.855	0.78
pc1	0.718	0.893	0.808	0.722	0.812	0.759	0.81

Software Fault Prediction Performance of Base Predictors for AUC value:

Dataset	NB	DT	LR	MLP	SVM	KNN
jm1	0.543	0.595	0.572	0.588	0.559	0.576
cm1	0.703	0.676	0.538	0.717	0.501	0.641
kc1	0.602	0.752	0.761	0.752	0.652	0.663
kc2	0.650	0.669	0.659	0.741	0.722	0.770
pc1	0.558	0.793	0.608	0.722	0.724	0.604

Software Fault Prediction Performance of Ensemble Predictors for FM value:

Dataset	AB	BG	ETC	GBC	RF	SC	VC
jm1	0.607	0.631	0.635	0.614	0.631	0.639	0.61
cm1	0.884	0.599	0.839	0.801	0.762	0.875	0.81
kc1	0.858	0.852	0.861	0.852	0.755	0.811	0.78
kc2	0.770	0.769	0.759	0.841	0.812	0.756	0.81
pc1	0.818	0.893	0.708	0.822	0.855	0.798	0.81

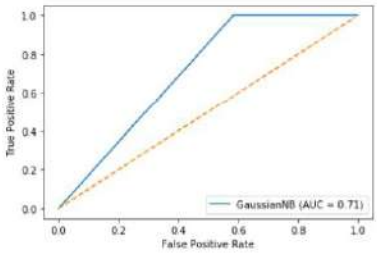
Best base algorithm and Best ensemble algorithm for CM1 dataset:

Naive Bayes-

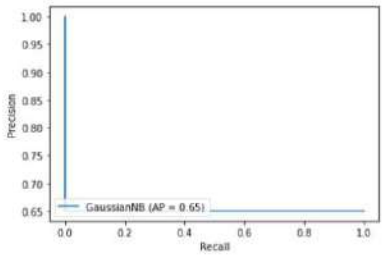
```
Accuracy:  72.0
Probability of detection of defect(Recall, pd):  1.0
Probability of false alarm(pf):  0.0
Probability of correct detection(Precision):  0.65


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()
```



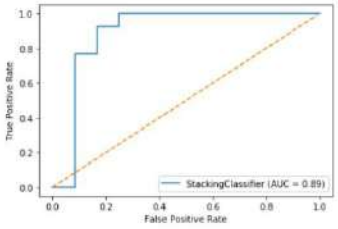
Stacking-

```
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

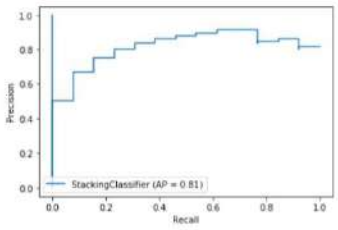
In [445]:

```
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()
```



Best base algorithm and Best ensemble algorithm for KC1 dataset:

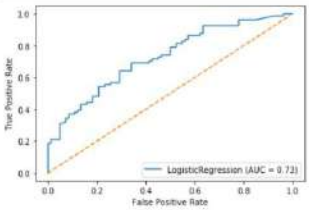
Logistic Regression-

```
Accuracy:  64.41717791411043
Probability of detection of defect(Recall, pd):  0.79012
Probability of false alarm(pf):  0.29310344827586204
Probability of correct detection(Precision):  0.60952380


F1-score or FM:  0.6881720430107527
AUC value:  0.6450617283950617


[[41 41]
 [17 64]]
```

	precision	recall	f1-score	support
0.0	0.71	0.50	0.59	82
1.0	0.61	0.79	0.69	81
accuracy			0.64	163
macro avg	0.66	0.65	0.64	163
weighted avg	0.66	0.64	0.64	163

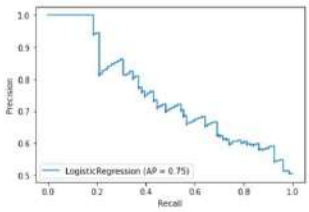
In [143]:

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

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



In [144]:

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



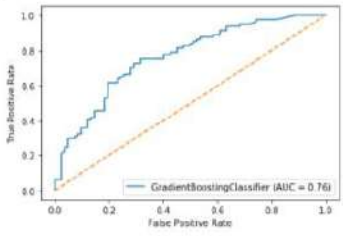
Gradient Boosting Classifier-

```
Accuracy:  71.16564417177914
Probability of detection of defect(Recall, pd):  0.7037037037037037
Probability of false alarm(pf):  0.2891566265060241
Probability of correct detection(Precision):  0.7125


F1-score or FM:  0.7080745341614907
AUC value:  0.7116079494128275


[[59 23]
 [24 57]]
```

	precision	recall	f1-score	support
0.0	0.71	0.72	0.72	82
1.0	0.71	0.70	0.71	81
accuracy			0.71	163
macro avg	0.71	0.71	0.71	163
weighted avg	0.71	0.71	0.71	163

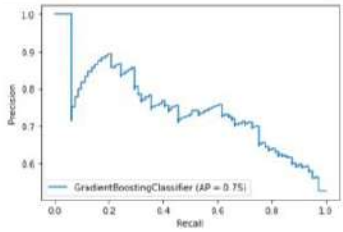
In [179]:

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

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



In [180]:

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



Best base algorithm and Best ensemble algorithm for KC2 dataset:

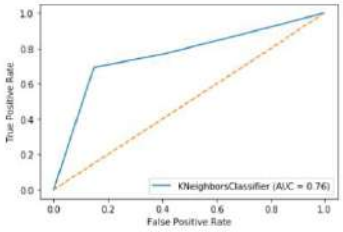
K-Nearest Neighbor-

```
Accuracy:  77.35849056603774
Probability of detection of defect(Recall, pd):  0.692307
Probability of false alarm(pf):  0.25806451612903225
Probability of correct detection(Precision):  0.818181818

F1-score or FM:  0.7500000000000001
AUC value:  0.7720797720797721

[[23  4]
 [ 8 18]]
```

	precision	recall	f1-score	support
0	0.74	0.85	0.79	27
1	0.82	0.69	0.75	26
accuracy			0.77	53
macro avg	0.78	0.77	0.77	53
weighted avg	0.78	0.77	0.77	53

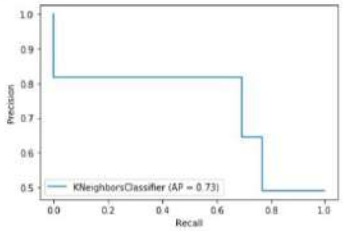
In [130]:

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

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



In [131]:

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



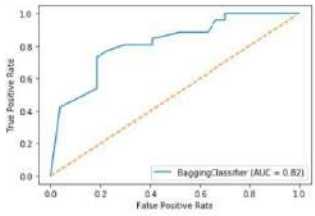
Bagging-

```
Accuracy:  77.35849056603774
Probability of detection of defect(Recall, pd):  0.769230
Probability of false alarm(pf):  0.2222222222222222
Probability of correct detection(Precision):  0.769230769

F1-score or FM:  0.7692307692307693
AUC value:  0.7735042735042735

[[21  6]
 [ 6 20]]
```

	precision	recall	f1-score	support
0	0.78	0.78	0.78	27
1	0.77	0.77	0.77	26
accuracy			0.77	53
macro avg	0.77	0.77	0.77	53
weighted avg	0.77	0.77	0.77	53

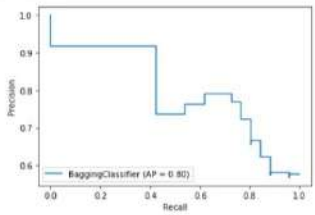
In [96]:

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

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



In [97]:

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



Best base algorithm and Best ensemble algorithm for KC2 dataset:

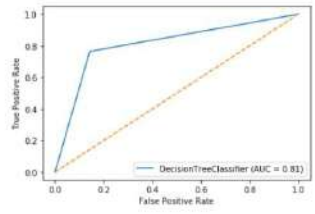
Decision Tree-

```
Accuracy:  81.57894736842105
Probability of detection of defect(Recall, pd):  0.76470
Probability of false alarm(pf):  0.18181818181818182
Probability of correct detection(Precision):  0.8125

F1-score or FM:  0.787878787878788
AUC value:  0.8109243697478993

[[18  3]
 [ 4 13]]
```

	precision	recall	f1-score	support
0.0	0.82	0.86	0.84	21
1.0	0.81	0.76	0.79	17
accuracy			0.82	38
macro avg	0.82	0.81	0.81	38
weighted avg	0.82	0.82	0.82	38

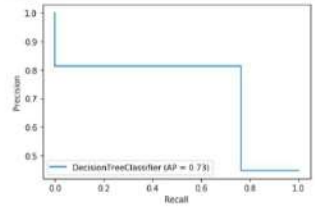
In [74]:

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

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



In [75]:

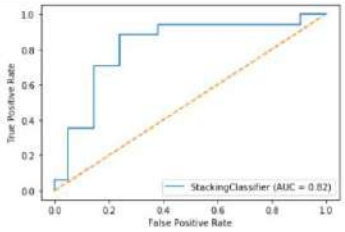
```
disp = plot_precision_recall_curve(tmodel, xtest, ytest)
plt.show()
```



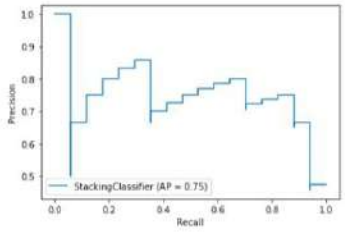
Stacking-

```
Accuracy:  81.57894736842105
Probability of detection of defect(Recall, pd):  0.8823
Probability of false alarm(pf):  0.1111111111111111
Probability of correct detection(Precision):  0.75


F1-score or FM:  0.8108108108108107
AUC value:  0.8221288515406161


[[16  5]
 [ 2 15]]


              precision    recall  f1-score   support

         0.0       0.89      0.76      0.82        21
         1.0       0.75      0.88      0.81        17

    accuracy                           0.82        38
   macro avg       0.82      0.82      0.82        38
weighted avg       0.83      0.82      0.82        38
```

```
In [110]:
x=np.arange(100)*0.01
y=x

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




```
In [111]:
disp = plot_precision_recall_curve(stmodel, xtest, ytest)
plt.show()
```



VII. Conclusion

In this report after trying various base predictor algorithms and various ensemble learning algorithms on different fault prediction dataset such as CM1, JM1 etc. we can conclude that ensemble learners performed fairly well than base learners. We have used various datasets from promise repository which are contributed by NASA on various softwares like CM1, JM1, KC1, KC2 and PC1. Different base algorithms like SVM(Support Vector Machine), KNN(K Nearest Neighbor) etc. are used, total of 6 algorithms. Various ensemble learners like Adaboost, Bagging, Stacking, Voting etc. are used, total of 7 algorithms. All there results are compared in tables, graphs and results of best performing algorithms are shown in required section for each dataset. From the table and graphs it is clearly visible that in all datasets, ensemble learners performs better than base algorithms. For all datasets, FM and AUC values are better for ensemble algorithms. Metircs used for result comparison are FM(F-measure) value, AUC(Area Under ROC curve) value, Recall, Precision value, PF(Probability of False Alarm). All these values for different algorithms are shown in result section. Graph includes ROC(receiver operating characteristic curve) and Precision-Recall curve. A precision-recall curve shows the relationship between precision (= positive predictive value= TP / (TP + FP)) and recall(= sensitivity = TP / (TP + FN)). An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification threshold.

Precision-

$$PR = \frac{TP}{TP + FP}$$

Recall-

$$PD = \frac{TP}{TP + FN}$$

Accuracy-

$$ACC = \frac{TP + TN}{TP + FP + TN + FN}$$

Probability of False Alarm(PF)-

$$PF = \frac{FP}{FP + TN}$$

F-measure(FOM)-

$$FM = \frac{2 \times Recall \times Precision}{Recall + Precision}$$

Future Work-

Since dataset is quite unbalanced, oversampling can be used to increase accuracy. Due to unbalanced data Precision, Recall and F-measure are used, any other good metrics can substitute them. Probability of false alarm can be further decreased by tuning hyperparameters of different algorithms. Advanced deep learning concepts has also a place in coming times to increase accuracy of algorithms using different ways.

References

[1] An industrial case study of classifiifier ensembles for locating software defects, Ayse Tosun Mısırlı, Ayse Basar Bener and Burak Turhan, online: 13 January 2011, Springer Science+Business Media, LLC 2011.

[2] An Ensemble Approach of Simple Regression Models to Cross-Project Fault Prediction, Satoshi Uchigaki, Shinji Uchida, Koji Toda, Akito Monden, 2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing

[3] Analyzing the Effect of Bagged Ensemble Approach for Software Fault Prediction in Class Level and Package Level Metrics, Shanthini. A and Chandrasekaran.RM, ICICES2014 - S.A.Engineering College, Chennai, Tamil Nadu, India

[4] Linear and non-linear heterogeneous ensemble methods to predict the number of fault in software systems Knowledge-Based Systems, Volume 119, 1 March 2017, Pages 232-256

Santosh Singh Rathore, Sandeep Kumar

[5]Towards an ensemble bsed system for predicting the number of software fault Expert Systems with Applications, Volume 82, 1 October 2017, Pages 357-382

Santosh Singh Rathore, Sandeep Kumar.

[6]Particle swarm optimization-based ensemble learning for software change prediction Information and Software Technology, Volume 102, October 2018, Pages 65-84 Ruchika Malhotra, Megha Khanna.

[7] A comparative study of unsupervised learning algorithms for software fault prediction---R. Jothi---IEEE.

[8] Improving Failure Prediction by Ensembling the Decisions of Machine Learning Models: A Case Study,JOAO R. CAMPOS , ERNESTO COSTA , AND MARCO VIEIRA DEI/CISUC, University of Coimbra, 3030-290 Coimbra, Portugal December 23, 2019.

[9] EXPERIMENTAL STUDY ON SOFTWARE FAULT PREDICTION. USING MACHINE LEARNING MODEL, Thi Minh Phuong Ha, -Duy Hung Tran, LE Thi My Hanh, Nguyen Thanh Binh

[10] Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability, Fatih Yucalar, Akin Ozcift, Emin Borandag, Deniz Kilinc, Manisa Celal Bayar University, Department of Software Engineering, Manisa, Turkey, 18 October 2019

[11] Enhanced Binary Moth Flame Optimization as a Feature Selection Algorithm to Predict Software Fault Prediction IYAD TUMAR 1, (Member, IEEE), YOUSEF HASSOUNEH 2, HAMZA TURABIEH 3, AND THAER THAHER 4

[12]Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability, Fatih Yucalar, Akin Ozcift, Emin Borandag, Deniz Kilinc, Manisa Celal Bayar University, Department of Software Engineering, Manisa, Turkey, 18 October 2019

[13]A systematic literature review of machine learning techniques for software maintainability prediction.Information and software Technology, Volume 119, March 2020, Article 106214 Hadeel Alsolai, Marc Roper.