NEW JERSEY INSTITUTE OF TECHNOLOGY
CS634 DATA MINING

FINAL PROJECT

Khang Tran
kt36@njit.edu

1. INTRODUCTION:

In this final project, I implemented three machine learning algorithms: Support Vector Machine, Random Forrest and Naïve Bayes, to solve a **binary classification problem** - breast cancer prediction base on the patient's features. Every year, thousands of patients pass away because of breast cancer and most of them found out about their own disease when they already have symptoms like breast pain. Predicting breast cancer early through the bodies' feature will help reduce the number of deaths every year which is a very essential task in healthcare and medical service.

I used scikit-learn package – a python package for machine learning – to implement the algorithms. To evaluate my models for this task, I re-implement twelve metrics based on the confusion matrix and the results are significantly great which implies the models are goof and trustable for this task.

The structure of this report is as follow: in section 2, I will introduce the datasets; in section 3, I will go through the codes and implementation of the algorithms; finally, results will be displayed in the section 4.

Github link: https://github.com/khangtran2020/CS634_finalproject.git

2. DATASET:

Link to dataset: https://www.kaggle.com/c/breast-cancer-detection/data

I downloaded the breast cancer dataset from a Kaggle competition from the link above. This dataset contains a train file and a test file. However, in this project, I only used the train file since the test-set's labels are hidden by the competition's organizers. In this training set, it contains 33 columns: 32 features columns and 1 label column – `['id', 'radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean', 'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean', 'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se', 'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se', 'fractal_dimension_se', 'radius_worst', 'texture_worst', _worst', 'area_worst', 'smoothness_worst', 'compactness_worst', 'concavity_worst', 'concave points_worst', 'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32', 'diagnosis']`. However, there are 2 useless feature columns: `'Unnamed: 32'` an d `'id'`, so I dropped these columns. The label (target) of the data is the `'diagnosis'` which has value 'B' for benign and 'M' for malicious.

There's not null value in the dataset, so I don't have to fill in the null value. All of the column s are numerical value except the label. Therefore, I normalized the data before training and also applied the label encoder to the label columns.

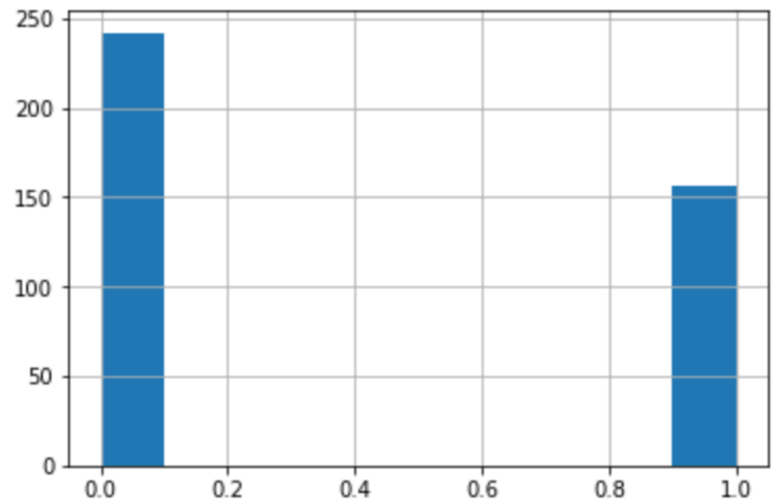


Figure 1: Distribution of the labels

The labels are imbalanced, so having a right and trustable metrics for this task is essential step. In the implementation I will go more detail how I deal with the data, but briefly, the data consists of 398 rows represent for the patients. I also implement k-Fold with 10-Fold, so each fold contains about 39 – 40 data points.

3. IMPLEMENTATION:

In this part, I will report my implementation for this project. I will go through the libraries, data processing, metrics implementation and training process.

a. Libraries:

In this project, I used the libraries as in figure 2. First of all, I used pandas and os to read files and read the datasets since they're very strong in data processing. Pandas changes read the csv files and put it into a dataframe which can be easily manipulated and processed. I also used seaborn and matplotlib to plot the distribution of the features. Finally, for the machine learning algorithms, I used sklearn – scikit learn package which has many implemented machine learning algorithms in an optimized way.

```
In [1]: import pandas as pd
        import numpy as np
        import os
        from sklearn import preprocessing
        from sklearn.model_selection import KFold
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.svm import SVC
        from sklearn.naive_bayes import GaussianNB
        from sklearn.metrics import confusion_matrix
        import seaborn as sns
        import matplotlib.pyplot as plt
```

Figure 2: Libraries used

And also, I used numpy combining with sklearn confusion matrix to implement the metrics.

b. Data preprocessing:

As mention in section 2, some of the columns are not useful since they don't carry any information regards to the labels. Therefore, I dropped the useless features. For the rest of the columns, they are numerical value and have different scales, so I use standard scaler to scale the columns so that they all have the same range from 0 to 1.

```
In [4]: data = data.drop(['Unnamed: 32','id'], axis = 1)
        data.head()
Out[4]:
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_
0	11.80	16.58	78.99	432.0	0.10910	0.17000	0.16590	0.07415	0.2678	
1	10.26	14.71	66.20	321.6	0.09882	0.09159	0.03581	0.02037	0.1633	
2	15.30	25.27	102.40	732.4	0.10820	0.16970	0.16830	0.08751	0.1926	
3	12.47	17.31	80.45	480.1	0.08928	0.07630	0.03609	0.02369	0.1526	
4	15.32	17.27	103.20	713.3	0.13350	0.22840	0.24480	0.12420	0.2398	

5 rows × 31 columns

```
In [5]: for col in data.columns:
            if data[col].dtypes == 'object':
                le = preprocessing.LabelEncoder()
                data[col] = le.fit_transform(data[col])
            else:
                scaler = preprocessing.StandardScaler()
                data[col] = scaler.fit_transform(np.array(data[col]).reshape(-1, 1))
```

Figure 3: Data preprocessing

And finally, for the diagnosis columns, since it's the label for the patients and originally its type is object, I use the label encoder of sklearn to change it into numerical: 1 for not normal and 0 for normal.

c. Metrics:

I used the sklearn confusion matrix to get the confusion matrix given the prediction and the true value. Then, from the confusion matrix, I got the true positive (tp), false positive (fp), true negative (tn) and false negative (fn) from the confusion matrix and put it in a metrics function which is re-implemented.

```python
def metric(tn, fp, fn, tp):
    result = []
    tpr = tp/(tp+fn)
    result.append(tpr)
    tnr = tn/(tn+fp)
    result.append(tnr)
    fpr = fp/(tn+fp)
    result.append(fpr)
    fnr = fn/(tp+fn)
    result.append(fnr)
    recall = tp/(tp+fn)
    result.append(recall)
    precision = tp/(tp+fp)
    result.append(precision)
    f1 = (2*tp)/(2*tp+fp+fn)
    result.append(f1)
    acc = (tp+tn)/(tp+fp+fn+tn)
    result.append(acc)
    err = (fp+fn)/(tp+fp+fn+tn)
    result.append(err)
    bacc = (tpr+tnr)/2
    result.append(bacc)
    tss = tp/(tp+fn) - fp/(fp+tn)
    result.append(tss)
    hss = 2*(tp*tn - fp*fn)/((tp+fn)*(fn+tn) + (tp+fp)*(fp+tn))
    result.append(hss)
    return np.array(result)
```

Figure 4: Metrics

The metrics I used includes:

- True positive rate: tp/(tp+fn)
- False negative rate: fn/(tp+fn)
- False positive rate: fp/(tn+fp)
- True negative rate: tn/(tn+fp)
- Recall: tp/(tp+fn)
- Precision: tp/(tp+fp)
- F1-score: (2tp)/(2tp+fp+fn)
- Accuracy: (tp+tn)/(tp+fp+fn+tn)
- Error: (fp+fn)/(tp+fp+fn+tn)
- BACC: (tpr+tnr)/2
- TSS: tp/(tp+fn) - fp/(fp+tn)
- HSS: 2(tp*tn - fp*fn)/((tp+fn)*(fn+tn) + (tp+fp)*(fp+tn))

d. Training:

For this project, I did 10-fold cross-validation. To implement cross-validation, I used sklearn to generate 10-fold data. For each fold, I created new models, re-trained them and performed validate on 1-fold data.

```python
[11]: kf = KFold(n_splits=10, random_state=123)
fold = 0
svc_mean = np.zeros(12)
rf_mean = np.zeros(12)
gnb_mean = np.zeros(12)
for train_index, test_index in kf.split(X, y):
    fold += 1
    print("Fold", str(fold))
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

Figure 5: 10-fold split

For SVM, I used the SVC class of sklearn package with gamma as “auto”. At each fold I re-created a new SVC model and train it on the X_train and y_train of that fold. Then I used the trained model to predict on X_test and apply the metric function to get the evaluation of that fold. Before the cross-validation process, I created a svc_mean list to keep up the evaluation for svc model of each fold.

```
#SVM
print("\tSVM model result:")
svc = SVC(gamma='auto')
svc.fit(X_train, y_train)
y_pred_svc = svc.predict(X_test)
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_svc).ravel()
svc_result = metric(tn, fp, fn, tp)
svc_mean += svc_result
print("\t\tTrue positive rate:", svc_result[0])
print("\t\tTrue negative rate:", svc_result[1])
print("\t\tFalse positive rate:", svc_result[2])
print("\t\tFalse negative rate:", svc_result[3])
print("\t\tRecall:", svc_result[4])
print("\t\tPrecision:", svc_result[5])
print("\t\tF1:", svc_result[6])
print("\t\tAccuracy:", svc_result[7])
print("\t\tError Rate:", svc_result[8])
print("\t\tBalance Accuracy:", svc_result[9])
print("\t\tTrue skill statistics:", svc_result[10])
print("\t\tHeidke skill score:", svc_result[11])
```

Figure 6: SVM model

For Random Forrest, I used the Random Forrest Classifier (RandomForestClassifier) class of sklearn package with max_depth equal 4 and 100 estimator. At each fold I re-created a new Random Forrest model and train it on the X_train and y_train of that fold. Then I used the trained model to predict on X_test and apply the metric function to get the evaluation of that fold. The same as SVC, before the cross-validation process, I created a rf_mean list to keep up the evaluation for random forrest model of each fold.

```
#Random Forrest
print("\tRandom Forest model result:")
rf = RandomForestClassifier(max_depth=5, random_state=0)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_rf).ravel()
rf_result = metric(tn, fp, fn, tp)
rf_mean += rf_result
print("\t\tTrue positive rate:", rf_result[0])
print("\t\tTrue negative rate:", rf_result[1])
print("\t\tFalse positive rate:", rf_result[2])
print("\t\tFalse negative rate:", rf_result[3])
print("\t\tRecall:", rf_result[4])
print("\t\tPrecision:", rf_result[5])
print("\t\tF1:", rf_result[6])
print("\t\tAccuracy:", rf_result[7])
print("\t\tError Rate:", rf_result[8])
print("\t\tBalance Accuracy:", rf_result[9])
print("\t\tTrue skill statistics:", rf_result[10])
print("\t\tHeidke skill score:", rf_result[11])
```

Figure 7: Random Forrest Classifier

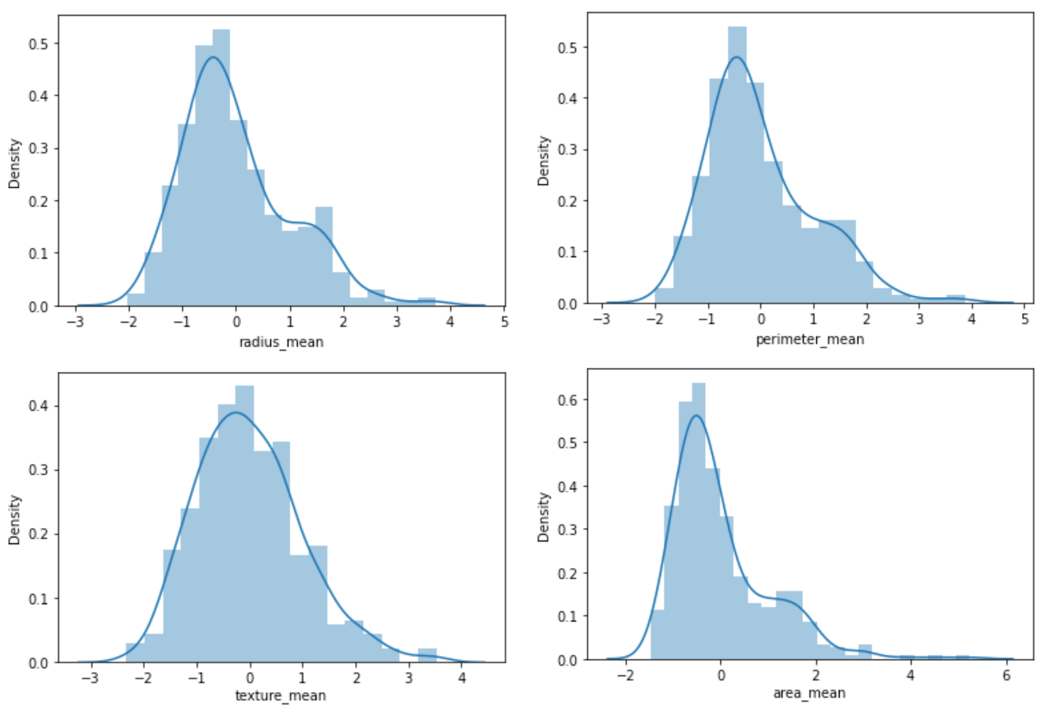
For Naïve Bayes, I used the Gaussian Naïve Bayes (GaussianNB) class of sklearn package with default setting since the features are numerical values. At each fold I re-created a new GaussianNB model and train it on the X_train and y_train of that fold. Then I used the trained model to predict on X_test and apply the metric function to get the evaluation of that fold. The same as SVC, before the cross-validation process, I created a gnb_mean list to keep up the evaluation for naïve bayes model of each fold.

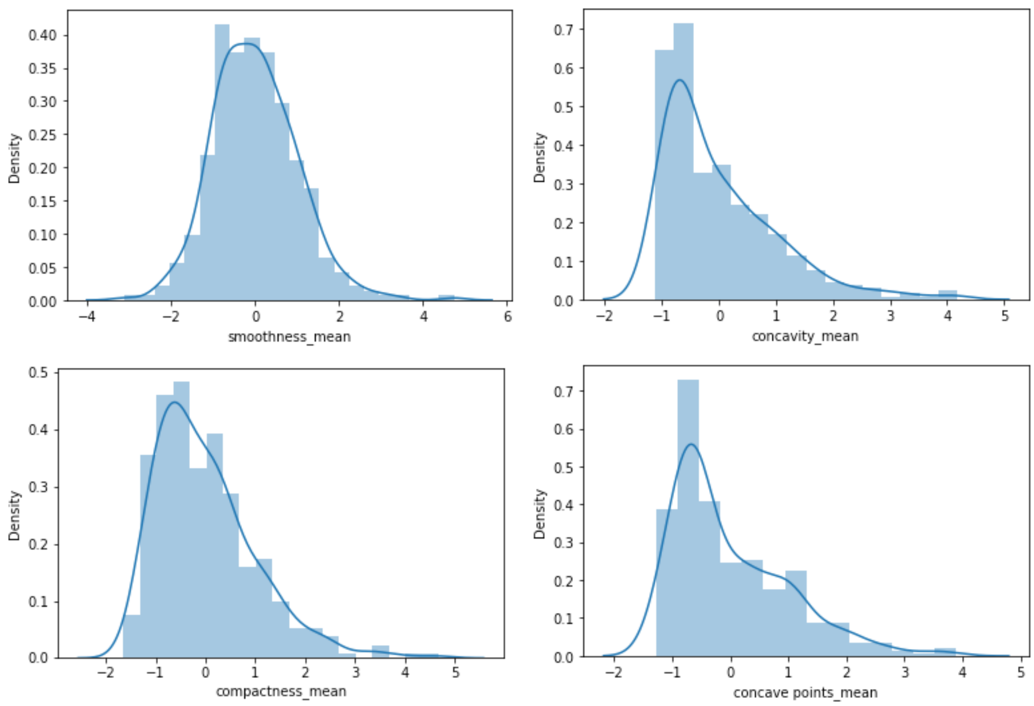
```
#Naive Bayes
print("\tNaive Bayes model result:")
gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred_gnb = gnb.predict(X_test)
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_gnb).ravel()
gnb_result = metric(tn, fp, fn, tp)
gnb_mean += gnb_result
print("\t\tTrue positive rate:", gnb_result[0])
print("\t\tTrue negative rate:", gnb_result[1])
print("\t\tFalse positive rate:", gnb_result[2])
print("\t\tFalse negative rate:", gnb_result[3])
print("\t\tRecall:", gnb_result[4])
print("\t\tPrecision:", gnb_result[5])
print("\t\tF1:", gnb_result[6])
print("\t\tAccuracy:", gnb_result[7])
print("\t\tError Rate:", gnb_result[8])
print("\t\tBalance Accuracy:", gnb_result[9])
print("\t\tTrue skill statistics:", gnb_result[10])
print("\t\tHeidke skill score:", gnb_result[11])
```

Figure 8: Naive Bayes

4. RESULTS:

a. Data Distribution:





b. Model Evaluation:

```
Fold 1
        SVM model result:
                True positive rate: 1.0
                True negative rate: 1.0
                False positive rate: 0.0
                False negative rate: 0.0
                Recall: 1.0
                Precision: 1.0
                F1: 1.0
                Accuracy: 1.0
                Error Rate: 0.0
                Balance Accuracy: 1.0
                True skill statistics: 1.0
                Heidke skill score: 1.0
        Random Forest model result:
                True positive rate: 0.9411764705882353
                True negative rate: 1.0
                False positive rate: 0.0
                False negative rate: 0.058823529411764705
                Recall: 0.9411764705882353
                Precision: 1.0
                F1: 0.9696969696969697
                Accuracy: 0.975
                Error Rate: 0.025
                Balance Accuracy: 0.9705882352941176
                True skill statistics: 0.9411764705882353
                Heidke skill score: 0.9484536082474226
        Naive Bayes model result:
                True positive rate: 0.8823529411764706
                True negative rate: 1.0
                False positive rate: 0.0
                False negative rate: 0.11764705882352941
                Recall: 0.8823529411764706
                Precision: 1.0
                F1: 0.9375
                Accuracy: 0.95
                Error Rate: 0.05
                Balance Accuracy: 0.9411764705882353
                True skill statistics: 0.8823529411764706
                Heidke skill score: 0.8961038961038961
```

```
Fold 2
        SVM model result:
                True positive rate: 1.0
                True negative rate: 1.0
                False positive rate: 0.0
                False negative rate: 0.0
                Recall: 1.0
                Precision: 1.0
                F1: 1.0
                Accuracy: 1.0
                Error Rate: 0.0
                Balance Accuracy: 1.0
                True skill statistics: 1.0
                Heidke skill score: 1.0
        Random Forest model result:
                True positive rate: 0.9411764705882353
                True negative rate: 0.9565217391304348
                False positive rate: 0.043478260869565216
                False negative rate: 0.058823529411764705
                Recall: 0.9411764705882353
                Precision: 0.9411764705882353
                F1: 0.9411764705882353
                Accuracy: 0.95
                Error Rate: 0.05
                Balance Accuracy: 0.948849104859335
                True skill statistics: 0.8976982097186701
                Heidke skill score: 0.8976982097186701
        Naive Bayes model result:
                True positive rate: 0.9411764705882353
                True negative rate: 1.0
                False positive rate: 0.0
                False negative rate: 0.058823529411764705
                Recall: 0.9411764705882353
                Precision: 1.0
                F1: 0.9696969696969697
                Accuracy: 0.975
                Error Rate: 0.025
                Balance Accuracy: 0.9705882352941176
                True skill statistics: 0.9411764705882353
                Heidke skill score: 0.9484536082474226
```

```
Fold 9
    SVM model result:
        True positive rate: 0.9473684210526315
        True negative rate: 1.0
        False positive rate: 0.0
        False negative rate: 0.05263157894736842
        Recall: 0.9473684210526315
        Precision: 1.0
        F1: 0.972972972972973
        Accuracy: 0.9743589743589743
        Error Rate: 0.02564102564102564
        Balance Accuracy: 0.9736842105263157
        True skill statistics: 0.9473684210526315
        Heidke skill score: 0.9486166007905138
    Random Forest model result:
        True positive rate: 0.9473684210526315
        True negative rate: 1.0
        False positive rate: 0.0
        False negative rate: 0.05263157894736842
        Recall: 0.9473684210526315
        Precision: 1.0
        F1: 0.972972972972973
        Accuracy: 0.9743589743589743
        Error Rate: 0.02564102564102564
        Balance Accuracy: 0.9736842105263157
        True skill statistics: 0.9473684210526315
        Heidke skill score: 0.9486166007905138
    Naive Bayes model result:
        True positive rate: 0.8947368421052632
        True negative rate: 1.0
        False positive rate: 0.0
        False negative rate: 0.10526315789473684
        Recall: 0.8947368421052632
        Precision: 1.0
        F1: 0.9444444444444444
        Accuracy: 0.9487179487179487
        Error Rate: 0.05128205128205128
        Balance Accuracy: 0.9473684210526316
        True skill statistics: 0.8947368421052632
        Heidke skill score: 0.8970976253298153
```

```
Fold 10
    SVM model result:
        True positive rate: 0.9411764705882353
        True negative rate: 1.0
        False positive rate: 0.0
        False negative rate: 0.058823529411764705
        Recall: 0.9411764705882353
        Precision: 1.0
        F1: 0.9696969696969697
        Accuracy: 0.9743589743589743
        Error Rate: 0.02564102564102564
        Balance Accuracy: 0.9705882352941176
        True skill statistics: 0.9411764705882353
        Heidke skill score: 0.9475100942126514
    Random Forest model result:
        True positive rate: 0.9411764705882353
        True negative rate: 1.0
        False positive rate: 0.0
        False negative rate: 0.058823529411764705
        Recall: 0.9411764705882353
        Precision: 1.0
        F1: 0.9696969696969697
        Accuracy: 0.9743589743589743
        Error Rate: 0.02564102564102564
        Balance Accuracy: 0.9705882352941176
        True skill statistics: 0.9411764705882353
        Heidke skill score: 0.9475100942126514
    Naive Bayes model result:
        True positive rate: 0.8235294117647058
        True negative rate: 1.0
        False positive rate: 0.0
        False negative rate: 0.17647058823529413
        Recall: 0.8235294117647058
        Precision: 1.0
        F1: 0.9032258064516129
        Accuracy: 0.9230769230769231
        Error Rate: 0.07692307692307693
        Balance Accuracy: 0.9117647058823529
        True skill statistics: 0.8235294117647058
        Heidke skill score: 0.8403819918144612
```

```
    SVM model result:
        True positive rate: 1.0
        True negative rate: 1.0
        False positive rate: 0.0
        False negative rate: 0.0
        Recall: 1.0
        Precision: 1.0
        F1: 1.0
        Accuracy: 1.0
        Error Rate: 0.0
        Balance Accuracy: 1.0
        True skill statistics: 1.0
        Heidke skill score: 1.0
    Random Forest model result:
        True positive rate: 0.8888888888888888
        True negative rate: 1.0
        False positive rate: 0.0
        False negative rate: 0.1111111111111111
        Recall: 0.8888888888888888
        Precision: 1.0
        F1: 0.9411764705882353
        Accuracy: 0.975
        Error Rate: 0.025
        Balance Accuracy: 0.9444444444444444
        True skill statistics: 0.8888888888888888
        Heidke skill score: 0.9253731343283582
    Naive Bayes model result:
        True positive rate: 0.7777777777777778
        True negative rate: 1.0
        False positive rate: 0.0
        False negative rate: 0.2222222222222222
        Recall: 0.7777777777777778
        Precision: 1.0
        F1: 0.875
        Accuracy: 0.95
        Error Rate: 0.05
        Balance Accuracy: 0.8888888888888888
        True skill statistics: 0.7777777777777778
        Heidke skill score: 0.8443579766536965
```

```
        True negative rate: 0.9523809523809523
        False positive rate: 0.047619047619047616
        False negative rate: 0.10526315789473684
        Recall: 0.8947368421052632
        Precision: 0.9444444444444444
        F1: 0.918918918918919
        Accuracy: 0.925
        Error Rate: 0.075
        Balance Accuracy: 0.9235588972431077
        True skill statistics: 0.8471177944862156
        Heidke skill score: 0.8492462311557789
    Random Forest model result:
        True positive rate: 0.8947368421052632
        True negative rate: 0.9523809523809523
        False positive rate: 0.047619047619047616
        False negative rate: 0.10526315789473684
        Recall: 0.8947368421052632
        Precision: 0.9444444444444444
        F1: 0.918918918918919
        Accuracy: 0.925
        Error Rate: 0.075
        Balance Accuracy: 0.9235588972431077
        True skill statistics: 0.8471177944862156
        Heidke skill score: 0.8492462311557789
    Naive Bayes model result:
        True positive rate: 0.8947368421052632
        True negative rate: 0.9523809523809523
        False positive rate: 0.047619047619047616
        False negative rate: 0.10526315789473684
        Recall: 0.8947368421052632
        Precision: 0.9444444444444444
        F1: 0.918918918918919
        Accuracy: 0.925
        Error Rate: 0.075
        Balance Accuracy: 0.9235588972431077
        True skill statistics: 0.8471177944862156
        Heidke skill score: 0.8492462311557789
```

c. Overall Evaluation:

```
svc_mean /= 10
rf_mean /= 10
gnb_mean /= 10
```

Figure 9: Calculate overall evaluation

```
print("Overall result for SVM model:")
print("\tTrue positive rate:", svc_mean[0])
print("\tTrue negative rate:", svc_mean[1])
print("\tFalse positive rate:", svc_mean[2])
print("\tFalse negative rate:", svc_mean[3])
print("\tRecall:", svc_mean[4])
print("\tPrecision:", svc_mean[5])
print("\tF1:", svc_mean[6])
print("\tAccuracy:", svc_mean[7])
print("\tError Rate:", svc_mean[8])
print("\tBalance Accuracy:", svc_mean[9])
print("\tTrue skill statistics:", svc_mean[10])
print("\tHeidke skill score:", svc_mean[11])
```

```
Overall result for SVM model:
        True positive rate: 0.9629435579899976
        True negative rate: 0.9743915343915344
        False positive rate: 0.025608465608465608
        False negative rate: 0.03705644201000238
        Recall: 0.9629435579899976
        Precision: 0.9633755133755134
        F1: 0.9620123660123661
        Accuracy: 0.9698717948717949
        Error Rate: 0.030128205128205132
        Balance Accuracy: 0.968667546190766
        True skill statistics: 0.9373350923815321
        Heidke skill score: 0.9359636913678747
```

Figure 10: Overall results for SVM

```
print("Overall result for Random Forest model:")
print("\tTrue positive rate:", rf_mean[0])
print("\tTrue negative rate:", rf_mean[1])
print("\tFalse positive rate:", rf_mean[2])
print("\tFalse negative rate:", rf_mean[3])
print("\tRecall:", rf_mean[4])
print("\tPrecision:", rf_mean[5])
print("\tF1:", rf_mean[6])
print("\tAccuracy:", rf_mean[7])
print("\tError Rate:", rf_mean[8])
print("\tBalance Accuracy:", rf_mean[9])
print("\tTrue skill statistics:", rf_mean[10])
print("\tHeidke skill score:", rf_mean[11])
```

```
Overall result for Random Forest model:
        True positive rate: 0.9427600486888412
        True negative rate: 0.9734828617437312
        False positive rate: 0.02651713825626869
        False negative rate: 0.057239951311158735
        Recall: 0.9427600486888412
        Precision: 0.9642031171442937
        F1: 0.952650792679487
        Accuracy: 0.9623717948717948
        Error Rate: 0.03762820512820513
        Balance Accuracy: 0.9581214552162862
        True skill statistics: 0.9162429104325724
        Heidke skill score: 0.9197071842705571
```

Figure 11: Overall Results for Random Forrest

```python
print("Overall result for Naive Bayes model:")
print("\tTrue positive rate:", gnb_mean[0])
print("\tTrue negative rate:", gnb_mean[1])
print("\tFalse positive rate:", gnb_mean[2])
print("\tFalse negative rate:", gnb_mean[3])
print("\tRecall:", gnb_mean[4])
print("\tPrecision:", gnb_mean[5])
print("\tF1:", gnb_mean[6])
print("\tAccuracy:", gnb_mean[7])
print("\tError Rate:", gnb_mean[8])
print("\tBalance Accuracy:", gnb_mean[9])
print("\tTrue skill statistics:", gnb_mean[10])
print("\tHeidke skill score:", gnb_mean[11])
```

```
Overall result for Naive Bayes model:
        True positive rate: 0.8800207721415152
        True negative rate: 0.969126984126984
        False positive rate: 0.030873015873015868
        False negative rate: 0.11997922785848483
        Recall: 0.8800207721415152
        Precision: 0.957081807081807
        F1: 0.9152587428360954
        Accuracy: 0.9371794871794872
        Error Rate: 0.06282051282051282
        Balance Accuracy: 0.9245738781342496
        True skill statistics: 0.8491477562684994
        Heidke skill score: 0.8622785377668631
```

Figure 12: Overall results of Naive Bayes

5. CONCLUSION:

In this project, I implement SVM, Random Forrest and Naïve Bayes for Breast Cancer Classification task. The results of three models are very good and trustable since the evaluation have great value. From these results, I believe that these models can be used for this task in the futures.