NAME: ANMOL RAJ

ROLL NO: 001811001069

ML(Machine Learning) ASSIGNMENT 1

DATASETS USED : IRIS Dataset, Diabetes Dataset, Breast Cancer Dataset

Question 1: Naïve Bayes Algorithm on all datasets

Before applying any training algorithm split data into training and test sets.

```
In [60]: X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=8)
```

IRIS DATASET

Prepare Dataset for training

```
In [52]: col_names = ["sepal_length", "sepal_width", "petal_length", "petal_width", "class"]
```

```
In [53]: df = pd.read_csv("data/iris.data", names=col_names)
         df.head()
```

Out[53]:

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
In [55]: features = df.iloc[:, 0:4]
         labels = df['class']
```

```
In [56]: label_encoder = sklearn.preprocessing.LabelEncoder()
         label_encoder.fit(labels)

Out[56]: LabelEncoder()
```

```
In [57]: label_encoder.classes_

Out[57]: array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

```
In [58]: labels = label_encoder.transform(labels)
```

```
In [59]: labels

Out[59]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
In [60]: X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=8)
```

Multinomial Naïve Bayes

In [61]:
```
MultiNB = MultinomialNB()
MultiNB.fit(X_train, y_train)
```

Out[61]:
```
MultinomialNB()
```

In [62]:
```
print(f"Accuracy Score of Training Set:  {accuracy_score(y_train, MultiNB.predict(X_train))}")

y_pred_MNB = MultiNB.predict(X_test)
print(f"Accuracy Score of Test Set: {accuracy_score(y_test, y_pred_MNB)}")

f1 = f1_score(y_test, y_pred_MNB, average='weighted')
print(f"F1 Score of Test Set: {f1}")

print("Classification Report")
print(classification_report(y_test, y_pred_MNB))
```

```
Accuracy Score of Training Set:  0.9333333333333333
Accuracy Score of Test Set: 0.9
F1 Score of Test Set: 0.899248120300752
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       0.75      1.00      0.86         9
           2       1.00      0.73      0.84        11

    accuracy                           0.90        30
   macro avg       0.92      0.91      0.90        30
weighted avg       0.93      0.90      0.90        30
```

Bernoulli Naïve Bayes

In [63]:
```
BernNB = BernoulliNB()
BernNB.fit(X_train, y_train)
```

Out[63]:
```
BernoulliNB()
```

In [64]:
```
y_pred_BNB = BernNB.predict(X_test)
```

In [65]:
```
print(f"Accuracy Score of Training Set:  {accuracy_score(y_train, BernNB.predict(X_train))}")

print(f"Accuracy Score of Test Set: {accuracy_score(y_test, y_pred_BNB)}")

f1 = f1_score(y_test, y_pred_BNB, average='micro')
print(f"F1 Score of Test Set: {f1}")

print("Classification Report")
print(classification_report(y_test, y_pred_BNB))
```

```
Accuracy Score of Training Set:  0.3416666666666667
Accuracy Score of Test Set: 0.3
F1 Score of Test Set: 0.3
Classification Report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        10
           1       0.30      1.00      0.46         9
           2       0.00      0.00      0.00        11

    accuracy                           0.30        30
   macro avg       0.10      0.33      0.15        30
weighted avg       0.09      0.30      0.14        30
```

Gaussian Naïve Bayes

In [66]:
```
GaussNB = GaussianNB()
GaussNB.fit(X_train, y_train)
```

Out[66]:
```
GaussianNB()
```

In [67]:
```
y_pred_GNB = BernNB.predict(X_test)
```

In [68]:
```
print(f"Accuracy Score of Training Set:  {accuracy_score(y_train, GaussNB.predict(X_train))}")

print(f"Accuracy Score of Test Set: {accuracy_score(y_test, y_pred_GNB)}")

f1 = f1_score(y_test, y_pred_GNB, average='micro')
print(f"F1 Score of Test Set: {f1}")

print("Classification Report")
print(classification_report(y_test, y_pred_GNB))
```

```
Accuracy Score of Training Set:  0.975
Accuracy Score of Test Set: 0.3
F1 Score of Test Set: 0.3
Classification Report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        10
           1       0.30      1.00      0.46         9
           2       0.00      0.00      0.00        11

    accuracy                           0.30        30
   macro avg       0.10      0.33      0.15        30
weighted avg       0.09      0.30      0.14        30
```

As we can see by running the above algorithms on the Iris Dataset, Multinomial Naïve Bayes gives the best result.

DIABETES DATASET

Prepare Dataset for training

In [80]:
```
diabetes = pd.read_csv('data/diabetes.tab.txt', delimiter = "\t")
```

In [81]:
```
diabetes.columns
```

Out[81]:
```
Index(['AGE', 'SEX', 'BMI', 'BP', 'S1', 'S2', 'S3', 'S4', 'S5', 'S6', 'Y'], dtype='object')
```

In [82]:
```
features = diabetes.loc[:, diabetes.columns != 'SEX']
labels = diabetes['SEX']
```

In [83]:
```
features
```

Out[83]:

	AGE	BMI	BP	S1	S2	S3	S4	S5	S6	Y
0	59	32.1	101.00	157	93.2	38.0	4.00	4.8598	87	151
1	48	21.6	87.00	183	103.2	70.0	3.00	3.8918	69	75
2	72	30.5	93.00	156	93.6	41.0	4.00	4.6728	85	141
3	24	25.3	84.00	198	131.4	40.0	5.00	4.8903	89	206
4	50	23.0	101.00	192	125.4	52.0	4.00	4.2905	80	135
...	...	...	...	...	...	...	...	...	...	...
437	60	28.2	112.00	185	113.8	42.0	4.00	4.9836	93	178
438	47	24.9	75.00	225	166.0	42.0	5.00	4.4427	102	104
439	60	24.9	99.67	162	106.6	43.0	3.77	4.1271	95	132
440	36	30.0	95.00	201	125.2	42.0	4.79	5.1299	85	220
441	36	19.6	71.00	250	133.2	97.0	3.00	4.5951	92	57

442 rows × 10 columns

In [84]:
```
labels
```

Out[84]:
```
0      2
1      1
2      2
3      1
4      1
      ..
437    2
438    2
439    2
440    1
441    1
Name: SEX, Length: 442, dtype: int64
```

In [85]:
```
scaler = StandardScaler().fit(features)
```

In [86]:
```
X_scaled = scaler.transform(features)
```

In [87]:
```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, labels, test_size=0.2, random_state=8)
```

Multinomial Naïve Bayes algorithm is not applicable to negative feature values

Bernoulli Naïve Bayes

In [88]:
```
BernNB = BernoulliNB()
BernNB.fit(X_train, y_train)

y_pred_BNB = BernNB.predict(X_test)
print(f"Accuracy Score of Training Set:  {accuracy_score(y_train, BernNB.predict(X_train))}")

print(f"Accuracy Score of Test Set: {accuracy_score(y_test, y_pred_BNB)}")

f1 = f1_score(y_test, y_pred_BNB, average='micro')
print(f"F1 Score of Test Set: {f1}")

print("Classification Report")
print(classification_report(y_test, y_pred_BNB))
```

```
Accuracy Score of Training Set:  0.660056657223796
Accuracy Score of Test Set: 0.6966292134831461
F1 Score of Test Set: 0.6966292134831461
Classification Report
              precision    recall  f1-score   support

           1       0.71      0.72      0.72        47
           2       0.68      0.67      0.67        42

    accuracy                           0.70        89
   macro avg       0.70      0.70      0.70        89
weighted avg       0.70      0.70      0.70        89
```

Gaussian Naïve Bayes

In [89]:
```
GaussNB = GaussianNB()
GaussNB.fit(X_train, y_train)

print(f"Accuracy Score of Training Set:  {accuracy_score(y_train, GaussNB.predict(X_train))}")

y_pred_GNB = GaussNB.predict(X_test)
print(f"Accuracy Score of Test Set: {accuracy_score(y_test, y_pred_GNB)}")

f1 = f1_score(y_test, y_pred_GNB, average='micro')
print(f"F1 Score of Test Set: {f1}")

print("Classification Report")
print(classification_report(y_test, y_pred_GNB))
```

```
Accuracy Score of Training Set:  0.6572237960339944
Accuracy Score of Test Set: 0.7415730337078652
F1 Score of Test Set: 0.7415730337078652
Classification Report
              precision    recall  f1-score   support

           1       0.75      0.77      0.76        47
           2       0.73      0.71      0.72        42

    accuracy                           0.74        89
   macro avg       0.74      0.74      0.74        89
weighted avg       0.74      0.74      0.74        89
```

As we can see by running the algorithms, Gaussian Naïve Bayes gives the best result.

BREAST CANCER DATASET

Prepare Dataset

```
In [114]: data = pd.read_csv("data/breast-cancer-wisconsin.data", header=None)

In [115]: data = data[data[6] != '?']

In [116]: data.shape

Out[116]: (683, 11)

In [117]: # Preprocess

In [118]: X = data.iloc[:, 1: -1]
          y = data[10]

In [119]: y = y.replace(2, 0)
          y = y.replace(4, 1)
```

```
In [122]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=8)
```

Multinomial Naïve Bayes

```
In [123]: MultiNB = MultinomialNB()
          MultiNB.fit(X_train, y_train)

          print(f"Accuracy Score of Training Set:  {accuracy_score(y_train, MultiNB.predict(X_train))}")

          y_pred_MNB = GaussNB.predict(X_test)
          print(f"Accuracy Score of Test Set: {accuracy_score(y_test, y_pred_MNB)}")

          f1 = f1_score(y_test, y_pred_MNB, average='micro')
          print(f"F1 Score of Test Set: {f1}")

          print("Classification Report")
          print(classification_report(y_test, y_pred_MNB))
```

```
Accuracy Score of Training Set:  0.8992673992673993
Accuracy Score of Test Set: 0.9854014598540146
F1 Score of Test Set: 0.9854014598540146
Classification Report
              precision    recall  f1-score   support

           0       0.99      0.99      0.99        84
           1       0.98      0.98      0.98        53

    accuracy                           0.99       137
   macro avg       0.98      0.98      0.98       137
weighted avg       0.99      0.99      0.99       137
```

Bernoulli Naïve Bayes

```
In [125]: BernNB = BernoulliNB()
BernNB.fit(X_train, y_train)

y_pred_BNB = BernNB.predict(X_test)
print(f"Accuracy Score of Training Set:  {accuracy_score(y_train, BernNB.predict(X_train))}")

print(f"Accuracy Score of Test Set: {accuracy_score(y_test, y_pred_BNB)}")

f1 = f1_score(y_test, y_pred_BNB, average='micro')
print(f"F1 Score of Test Set: {f1}")

print("Classification Report")
print(classification_report(y_test, y_pred_BNB))
```

```
Accuracy Score of Training Set:  0.6593406593406593
Accuracy Score of Test Set: 0.6131386861313869
F1 Score of Test Set: 0.6131386861313869
Classification Report
              precision    recall  f1-score   support

           0       0.61      1.00      0.76        84
           1       0.00      0.00      0.00        53

    accuracy                           0.61       137
   macro avg       0.31      0.50      0.38       137
weighted avg       0.38      0.61      0.47       137
```

Gaussian Naïve Bayes

```
In [124]: GaussNB = GaussianNB()
GaussNB.fit(X_train, y_train)

print(f"Accuracy Score of Training Set:  {accuracy_score(y_train, GaussNB.predict(X_train))}")

y_pred_GNB = GaussNB.predict(X_test)
print(f"Accuracy Score of Test Set: {accuracy_score(y_test, y_pred_GNB)}")

f1 = f1_score(y_test, y_pred_GNB, average='micro')
print(f"F1 Score of Test Set: {f1}")

print("Classification Report")
print(classification_report(y_test, y_pred_GNB))
```

```
Accuracy Score of Training Set:  0.9542124542124543
Accuracy Score of Test Set: 0.9854014598540146
F1 Score of Test Set: 0.9854014598540146
Classification Report
              precision    recall  f1-score   support

           0       0.99      0.99      0.99        84
           1       0.98      0.98      0.98        53

    accuracy                           0.99       137
   macro avg       0.98      0.98      0.98       137
weighted avg       0.99      0.99      0.99       137
```

As we can see by running the algorithms, Gaussian Naïve Bayes gives the best result.

Question 2: Decision Tree Algorithm on all datasets

Datasets can be prepared in the same way as mentioned above.

IRIS DATASET

Without Parameter Tuning

In [71]:
```
dtclf = DecisionTreeClassifier()
dtclf.fit(X_train, y_train)
```

Out[71]:
```
DecisionTreeClassifier()
```

In [72]:
```
dtclf.get_params()
```

Out[72]:
```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 '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 [86]:
```
print(f"Accuracy Score of Training Set:  {accuracy_score(y_train, dtclf.predict(X_train))}")

y_pred_dtclf = dtclf.predict(X_test)
print(f"Accuracy Score of Test Set: {accuracy_score(y_test, y_pred_dtclf)}")

f1 = f1_score(y_test, y_pred_dtclf, average='weighted')
print(f"F1 Score of Test Set: {f1}")

print("Classification Report")
print(classification_report(y_test, y_pred_dtclf))
```

```
Accuracy Score of Training Set:  1.0
Accuracy Score of Test Set: 0.9
F1 Score of Test Set: 0.9
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       0.91      0.83      0.87        12
           2       0.83      0.91      0.87        11

    accuracy                           0.90        30
   macro avg       0.91      0.91      0.91        30
weighted avg       0.90      0.90      0.90        30
```

With Parameter Tuning

In [92]:
```
param_grid = {
    "max_depth" : [1,3,5,7,9,11,12],
    "min_samples_leaf":[1,2,3,4,5,6,7,8,9,10],
    "max_leaf_nodes":[None,10,20,30,40,50,60,70,80,90]
}

grid_search = GridSearchCV(estimator=dtclf,
                           param_grid=param_grid,
                           scoring='accuracy',
                           cv=5)
```

In [93]:
```
grid_search.fit(X_train, y_train)
```

Out[93]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': [1, 3, 5, 7, 9, 11, 12],
                         'max_leaf_nodes': [None, 10, 20, 30, 40, 50, 60, 70,
                                            80, 90],
                         'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]},
             scoring='accuracy')
```

In [94]:
```
grid_search.best_params_
```

Out[94]:
```
{'max_depth': 3, 'max_leaf_nodes': None, 'min_samples_leaf': 2}
```

In [95]:
```
grid_search.best_score_
```

Out[95]:
```
0.9833333333333334
```

```
In [96]: bestNB = grid_search.best_estimator_
         print(f"Accuracy Score of Training Set:  {accuracy_score(y_train, bestNB.predict(X_train))}")

         y_pred_bestNB = bestNB.predict(X_test)
         print(f"Accuracy Score of Test Set: {accuracy_score(y_test, y_pred_bestNB)}")

         f1 = f1_score(y_test, y_pred_bestNB, average='micro')
         print(f"F1 Score of Test Set: {f1}")

         print("Classification Report")
         print(classification_report(y_test, y_pred_bestNB))
```

```
Accuracy Score of Training Set:  0.9916666666666667
Accuracy Score of Test Set: 0.9
F1 Score of Test Set: 0.9
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       0.91      0.83      0.87        12
           2       0.83      0.91      0.87        11

    accuracy                           0.90        30
   macro avg       0.91      0.91      0.91        30
weighted avg       0.90      0.90      0.90        30
```

Observation: There is a very small improvement in the algorithm upon parameter tuning.

DIABETES DATASET

Without Parameter Tuning

```
In [31]: dtclf1 = DecisionTreeClassifier()
```

```
In [32]: dtclf1.fit(X_train, y_train)

Out[32]: DecisionTreeClassifier()
```

```
In [33]: print(f"Accuracy Score of Training Set:  {accuracy_score(y_train, dtclf1.predict(X_train))}")

         y_pred_dtclf1 = dtclf1.predict(X_test)
         print(f"Accuracy Score of Test Set: {accuracy_score(y_test, y_pred_dtclf1)}")

         f1 = f1_score(y_test, y_pred_dtclf1, average='micro')
         print(f"F1 Score of Test Set: {f1}")

         print("Classification Report")
         print(classification_report(y_test, y_pred_dtclf1))
```

```
Accuracy Score of Training Set:  1.0
Accuracy Score of Test Set: 0.6292134831460674
F1 Score of Test Set: 0.6292134831460674
Classification Report
              precision    recall  f1-score   support

           1       0.65      0.66      0.65        47
           2       0.61      0.60      0.60        42

    accuracy                           0.63        89
   macro avg       0.63      0.63      0.63        89
weighted avg       0.63      0.63      0.63        89
```

With Parameter Tuning

```
In [45]: param_grid = {
             "max_depth" : [1,3,5,7,9,11,12],
             "min_samples_leaf":[1,2,3,4,5,6,7,8,9,10],
             "max_leaf_nodes":[None,10,20,30,40,50,60,70,80,90]
         }

         grid_search = GridSearchCV(estimator=dtclf1,
                                    param_grid=param_grid,
                                    scoring='f1_micro',
                                    cv=5)
```

```
In [46]: grid_search.fit(X_train, y_train)

Out[46]: GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
                      param_grid={'max_depth': [1, 3, 5, 7, 9, 11, 12],
                                  'max_leaf_nodes': [None, 10, 20, 30, 40, 50, 60, 70,
                                                     80, 90],
                                  'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]},
                      scoring='f1_micro')
```

```
In [47]: grid_search.best_params_

Out[47]: {'max_depth': 7, 'max_leaf_nodes': 10, 'min_samples_leaf': 6}
```

```
In [49]: bestNB = grid_search.best_estimator_
         print(f"Accuracy Score of Training Set:  {accuracy_score(y_train, bestNB.predict(X_train))}")

         y_pred_bestNB = bestNB.predict(X_test)
         print(f"Accuracy Score of Test Set: {accuracy_score(y_test, y_pred_bestNB)}")

         f1 = f1_score(y_test, y_pred_bestNB, average='micro')
         print(f"F1 Score of Test Set: {f1}")

         print("Classification Report")
         print(classification_report(y_test, y_pred_bestNB))

         Accuracy Score of Training Set:  0.7790368271954674
         Accuracy Score of Test Set: 0.6179775280898876
         F1 Score of Test Set: 0.6179775280898876
         Classification Report
                       precision    recall  f1-score   support

                    1       0.64      0.62      0.63        47
                    2       0.59      0.62      0.60        42

             accuracy                           0.62        89
            macro avg       0.62      0.62      0.62        89
         weighted avg       0.62      0.62      0.62        89
```

Observation: As we can see, there is no real improvement in model upon parameter tuning. However, the tuned algorithm is not overfitting the data unlike the untuned algorithm.

BREAST CANCER DATASET

Without parameter tuning

In [57]:
```
dtclf2 = DecisionTreeClassifier()
dtclf2.fit(X_train, y_train)
```

Out[57]:
```
DecisionTreeClassifier()
```

In [58]:
```
print(f"Accuracy Score of Training Set:  {accuracy_score(y_train, dtclf2.predict(X_train))}")

y_pred_dtclf = dtclf2.predict(X_test)
print(f"Accuracy Score of Test Set: {accuracy_score(y_test, y_pred_dtclf)}")

f1 = f1_score(y_test, y_pred_dtclf, average='weighted')
print(f"F1 Score of Test Set: {f1}")

print("Classification Report")
print(classification_report(y_test, y_pred_dtclf))
```

```
Accuracy Score of Training Set:  1.0
Accuracy Score of Test Set: 0.9562043795620438
F1 Score of Test Set: 0.9559817777087538
Classification Report
              precision    recall  f1-score   support

           0       0.96      0.98      0.97        89
           1       0.96      0.92      0.94        48

    accuracy                           0.96       137
   macro avg       0.96      0.95      0.95       137
weighted avg       0.96      0.96      0.96       137
```

With Parameter Tuning

In [59]:
```
param_grid = {
    "max_depth" : [1,3,5,7,9,11,12],
    "min_samples_leaf":[1,2,3,4,5,6,7,8,9,10],
    "max_leaf_nodes":[None,10,20,30,40,50,60,70,80,90]
}

grid_search = GridSearchCV(estimator=dtclf2,
                           param_grid=param_grid,
                           scoring='f1',
                           cv=5)
```

In [60]:
```
grid_search.fit(X_train, y_train)
```

Out[60]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': [1, 3, 5, 7, 9, 11, 12],
                         'max_leaf_nodes': [None, 10, 20, 30, 40, 50, 60, 70,
                                            80, 90],
                         'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]},
             scoring='f1')
```

In [61]:
```
grid_search.best_params_
```

Out[61]:
```
{'max_depth': 12, 'max_leaf_nodes': 40, 'min_samples_leaf': 1}
```

In [62]:
```python
bestNB = grid_search.best_estimator_
print(f"Accuracy Score of Training Set:  {accuracy_score(y_train, bestNB.predict(X_train))}")

y_pred_bestNB = bestNB.predict(X_test)
print(f"Accuracy Score of Test Set: {accuracy_score(y_test, y_pred_bestNB)}")

f1 = f1_score(y_test, y_pred_bestNB, average='micro')
print(f"F1 Score of Test Set: {f1}")

print("Classification Report")
print(classification_report(y_test, y_pred_bestNB))
```

```
Accuracy Score of Training Set:  1.0
Accuracy Score of Test Set: 0.9635036496350365
F1 Score of Test Set: 0.9635036496350365
Classification Report
              precision    recall  f1-score   support

           0       0.97      0.98      0.97        89
           1       0.96      0.94      0.95        48

    accuracy                           0.96       137
   macro avg       0.96      0.96      0.96       137
weighted avg       0.96      0.96      0.96       137
```

Observation: There is small improvement in the model upon tuning.