CSAI_801_PROJECT_ABOELELA, 22398556

UNDER SUPERVISION OF

DR. MARWA EL SAYED

1. Data Exploration.

Findings:

a. The data had an unnecessary index column named “Unnamed: 0”, so we removed it using the Drop function.

Unnamed: 0	location	country	gender	age	vis_wuhan	from_wuhan	symptom1	symptom2	symptom3	symptom4	symptom5	symptom6	diff_sym_hos	result
0	104	8	1	66.0	1	0	14	31	19	12	3	1	8	1
1	101	8	0	56.0	0	1	14	31	19	12	3	1	0	0
2	137	8	1	46.0	0	1	14	31	19	12	3	1	13	0
3	116	8	0	60.0	1	0	14	31	19	12	3	1	0	0
4	116	8	1	58.0	0	0	14	31	19	12	3	1	0	0

b. The “age” column had some float values , thus it was necessary to change the type of the column to integer for better classification.

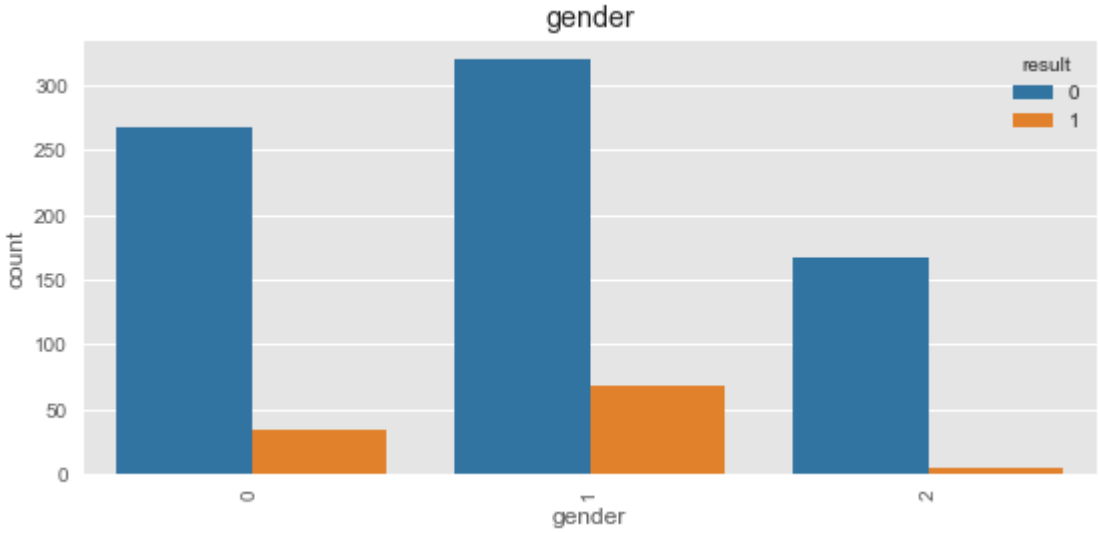
```
df["age"]
```

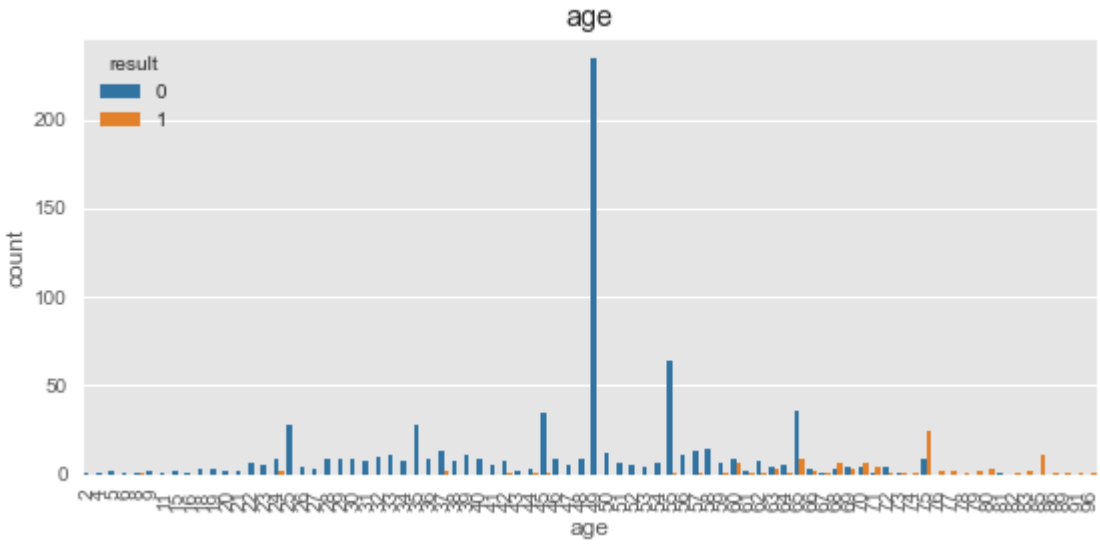
```
0      66.0
1      56.0
2      46.0
3      60.0
4      58.0
       ...
858    24.0
859    35.0
860    49.4
861    49.4
862    70.0
```

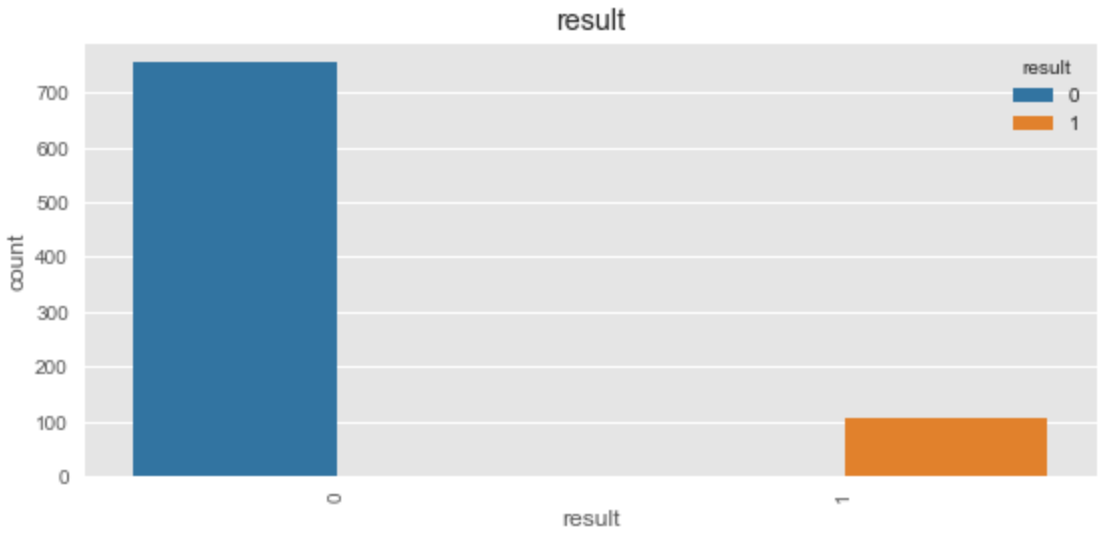
```
df["age"] = df["age"].astype(int)
df["age"]
```

```
0      66
1      56
2      46
3      60
4      58
       ..
858    24
859    35
860    49
861    49
862    70
```

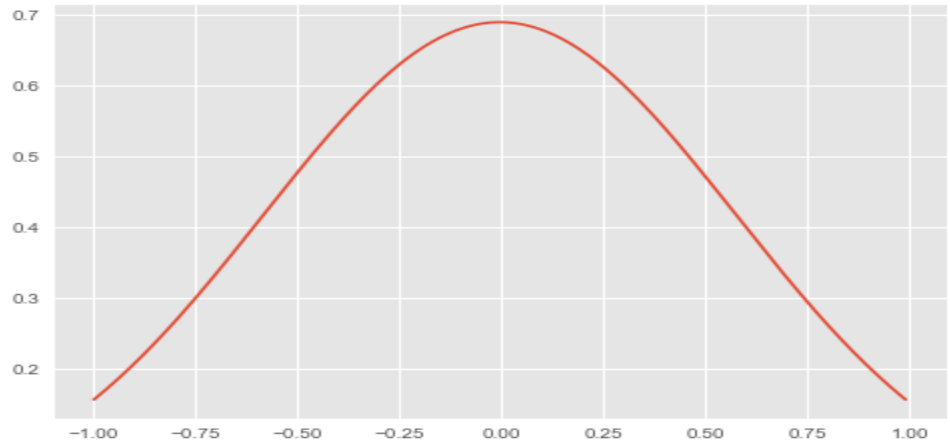
c. We visualized the the count of the each class in the label column and found that the data is highly imbalanced towards the class 0 , which means that the data needs standardarization, and recall ,percesion and F1-score are the suitable evaluation metrics for the problem.

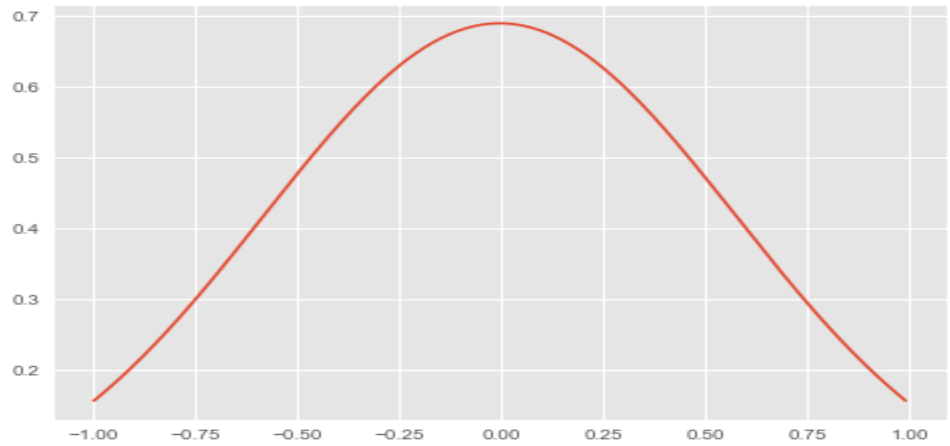






d. We visualized and checked the distribution of each Feature and found that the data is normally distributed , which Standard Scaler is the appropriate Scaling method.





2. Data Splitting and Scaling.

a. In splitting the data to training and testing we used **Stratify=y** , the stratify attribute of the train_test_split function splits the class proportionally between training and test set based on each class percentage to help with the imbalance issue.

b. Since the data is imbalanced and normally distributed we used the Standard Scaler on the training and testing data.

```
print(X_train,X_test)
```

```
[[ 0.93266532  1.53832489  1.57002126 ...  0.06116084  0.03809697
  -0.41482357]
 [ 1.00855138  0.14285627  0.18647762 ...  0.06116084  0.03809697
  -0.41482357]
 [-1.39450715  0.14285627  0.18647762 ...  0.06116084  0.03809697
   0.8921548 ]
 ...
 [ 1.03384674 -1.25261235 -1.19706602 ...  0.06116084  0.03809697
  -0.41482357]
 [ 0.93266532  1.53832489  1.57002126 ...  0.06116084  0.03809697
  -0.41482357]
 [-1.64746067 -0.61830843  0.18647762 ...  0.06116084  0.03809697
  -0.41482357]] [[ 0.45205362  0.14285627 -1.19706602 ...  0.06116084
  -0.41482357]
 [ 1.48916309  1.03088176 -1.19706602 ...  0.06116084  0.03809697
  -0.41482357]
 [ 0.19910009 -0.61830843  1.57002126 ...  0.06116084  0.03809697
  -0.41482357]
 ...
 [ 0.55323503 -1.12575156  0.18647762 ...  0.06116084  0.03809697
   0.45649534]
 [ 1.03384674 -1.25261235 -1.19706602 ...  0.06116084  0.03809697
  -0.41482357]
 [-1.72334673 -0.61830843  0.18647762 ...  0.06116084  0.03809697
  -0.41482357]]
```

3. Hyperparameter Optimization Method.

a. We used **GridSearchCV** which does Cross-Validation , so we don't have to split the data again.

b. Parameters chosen to be optimized for each Model.

   i. **K- Nearest Neighbor**: number of neighbors , weights.

   ii. **Logistic Regression**: C , Penalty , Solver.

   iii. **Naive Bayes**: Smoothing.

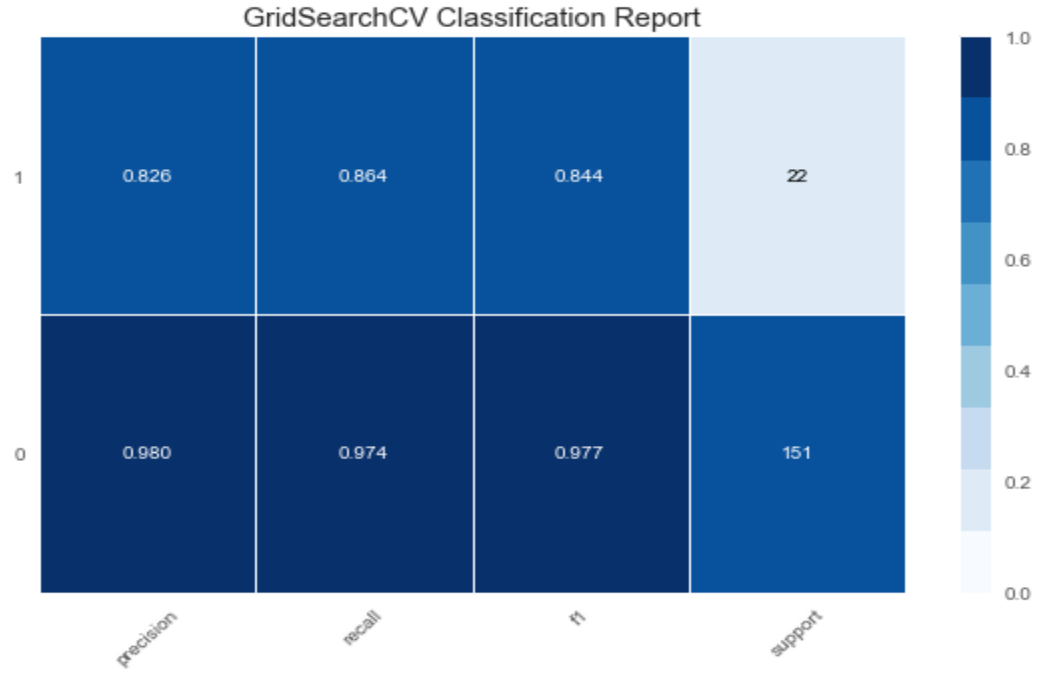
   iv. **Decision Tree**: Max depth, Minimum samples split, Criterion.

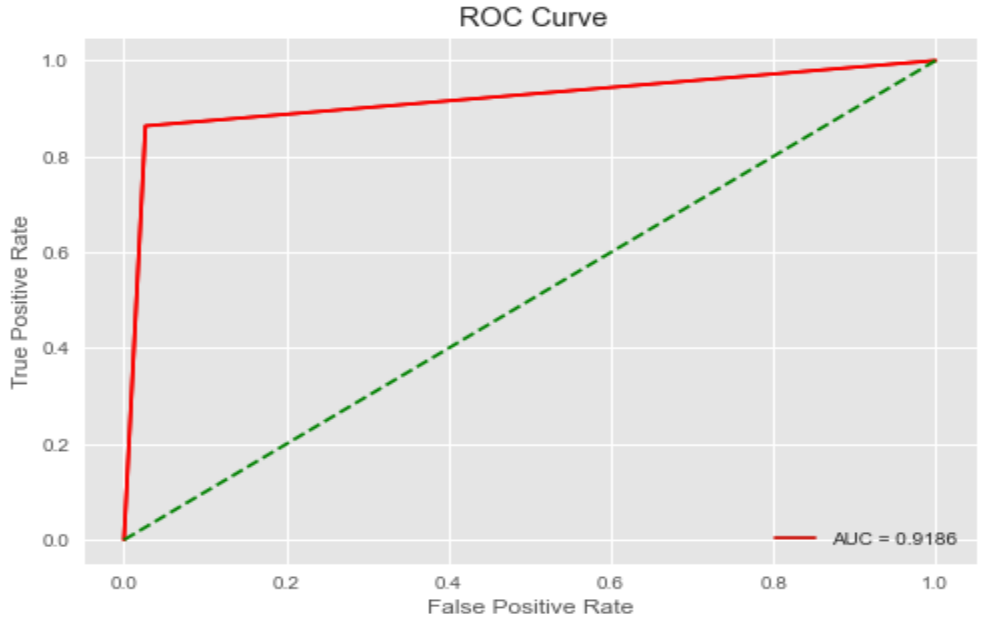
   v. **Support Vector Machine:** C , Kernal , Gamma

4. Model Performance Evaluation.

a. K- Nearest Neighbor.

```
Best parameter for K-Nearest Neighbors
{'n_neighbors': 1, 'weights': 'uniform'}
```





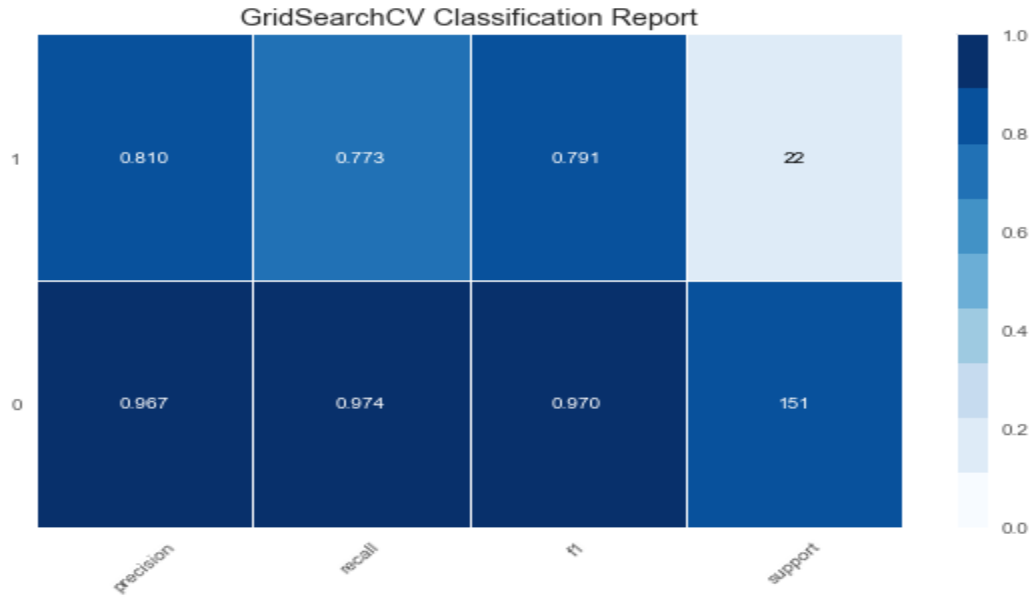
Findings:

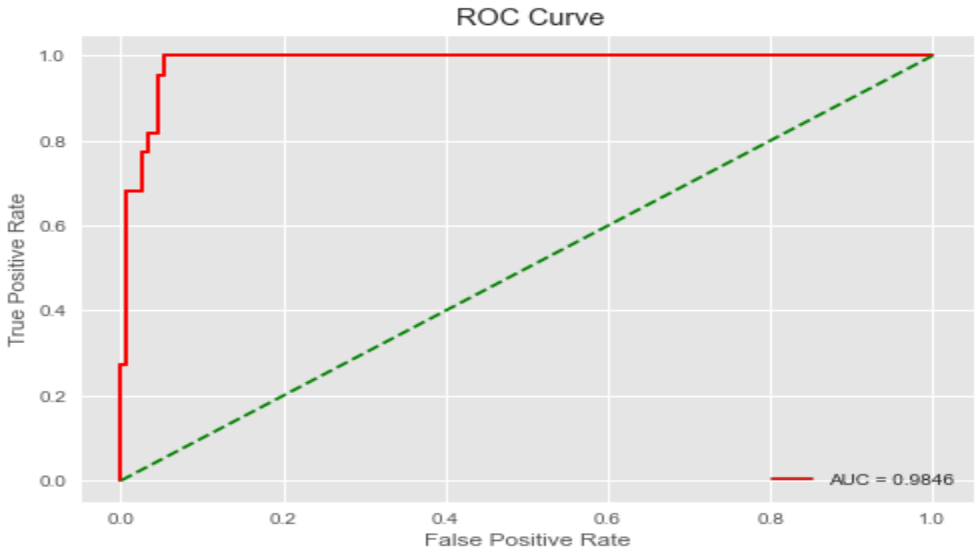
i. Precision ,F1 and Recall were good on class 0 with on average 97% , but had poorer performance on class 1 with average 84%.
ii. AUC gave Accuracy of 91% which is good.

---

b. Logistic Regression.

```
Best parameter for Logistic Regression
{'C': 10, 'penalty': 'l2', 'solver': 'sag'}
```





Findings:

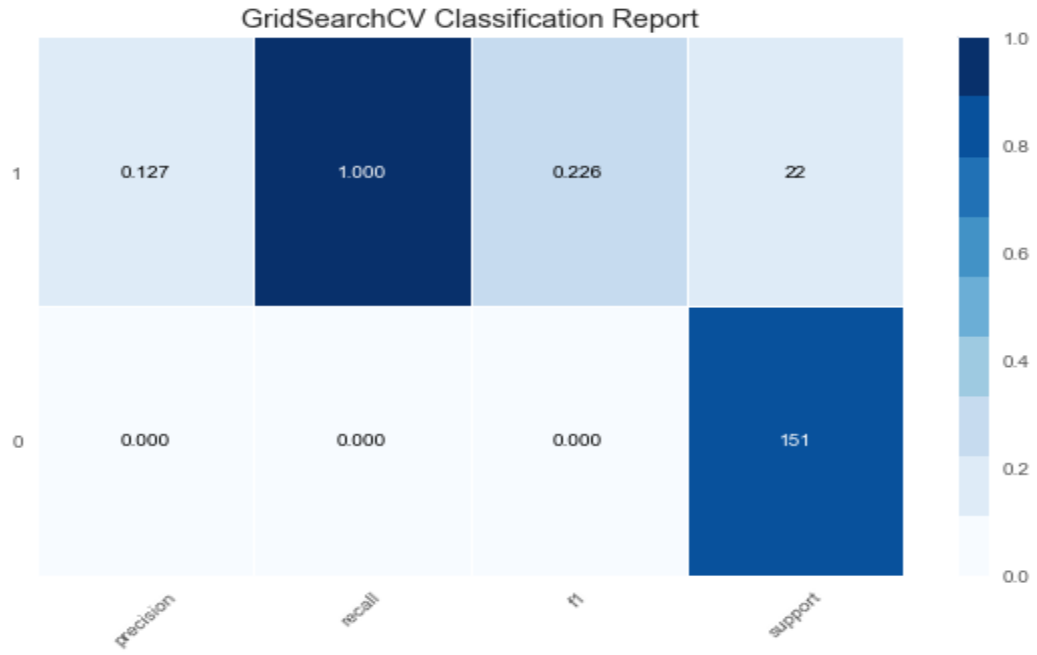
i. Similar to KNN it gave good performance on class 0 , but gave a poorer performance on class 1.

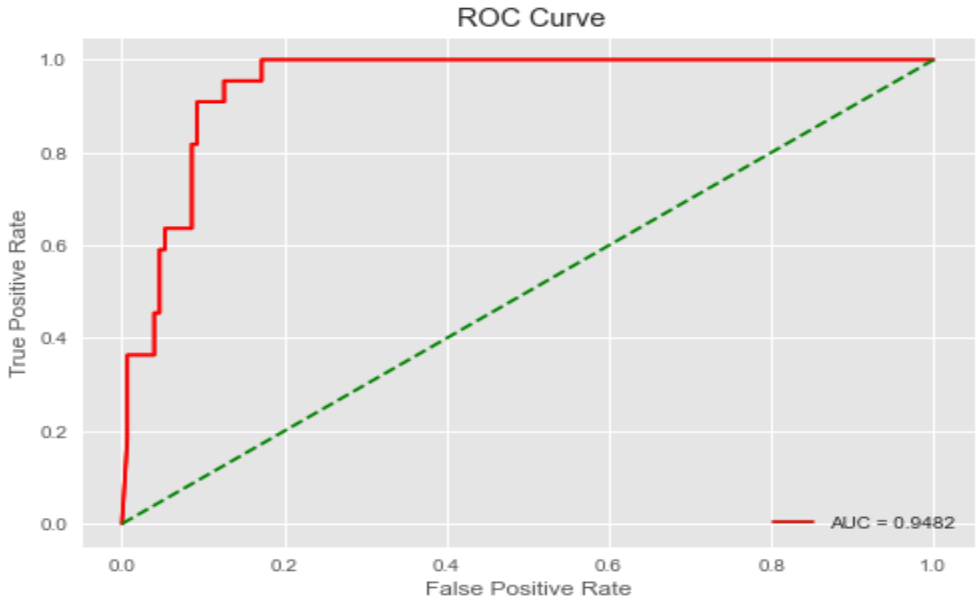
ii. The AUC showed Excellent accuracy of 98.4%.

---

c. Naive Bayes.

```
Best parameter for Naïve Bayes
{'var_smoothing': 1e-09}
```





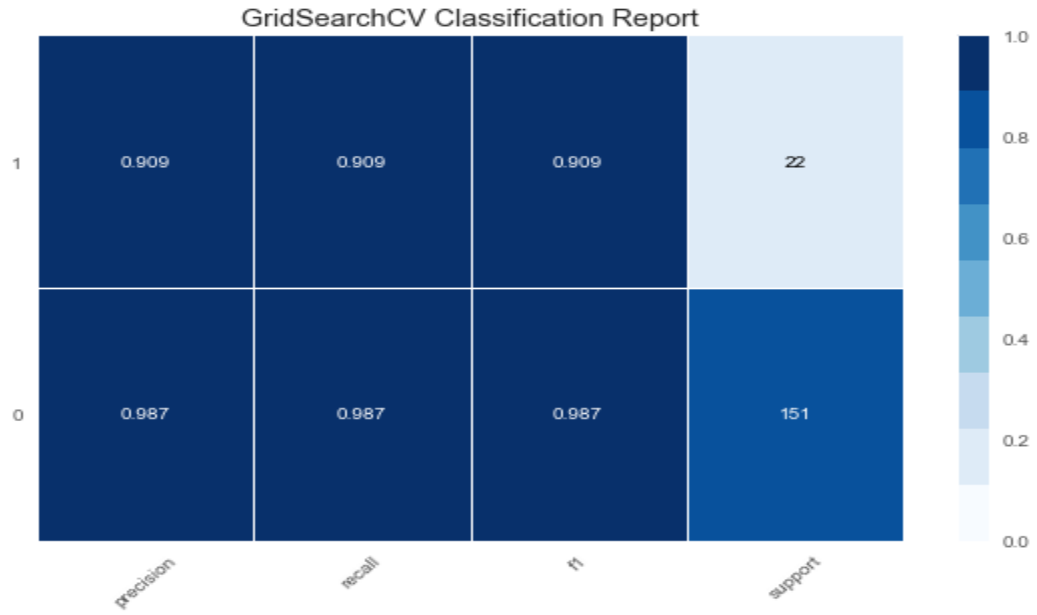
Findings:

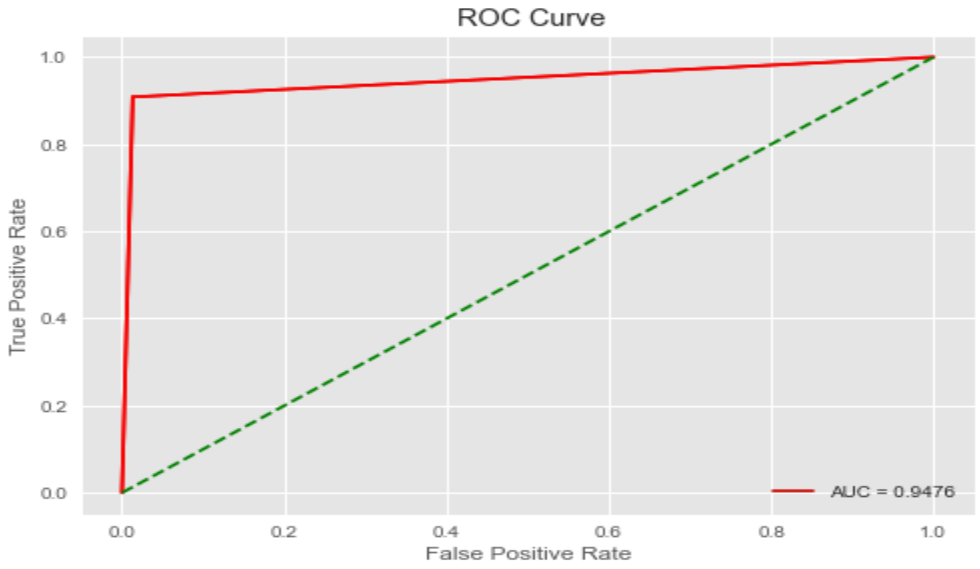
i. The model gave and overall very poor performance in both classes.
ii. AUC showed Accuracy of 94%
iii. Naive Bayes is not suitable for this Problem.

---

d. Decision Tree.

```
Best parameter for Decision Trees
{'criterion': 'entropy', 'max_depth': 7, 'min_samples_split': 2}
```





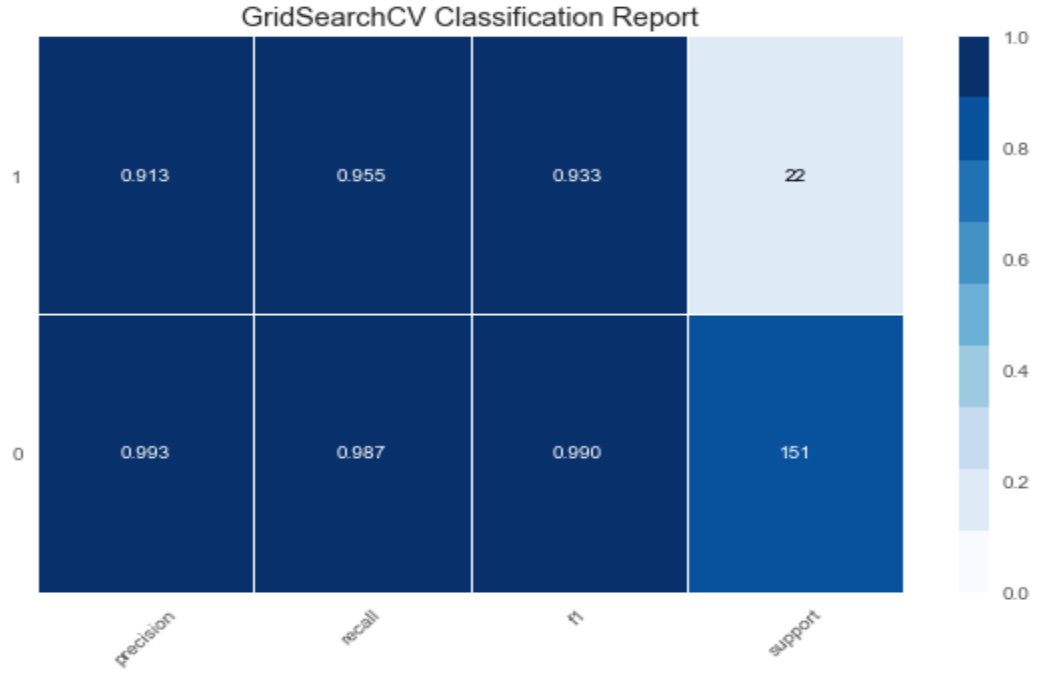
Findings:

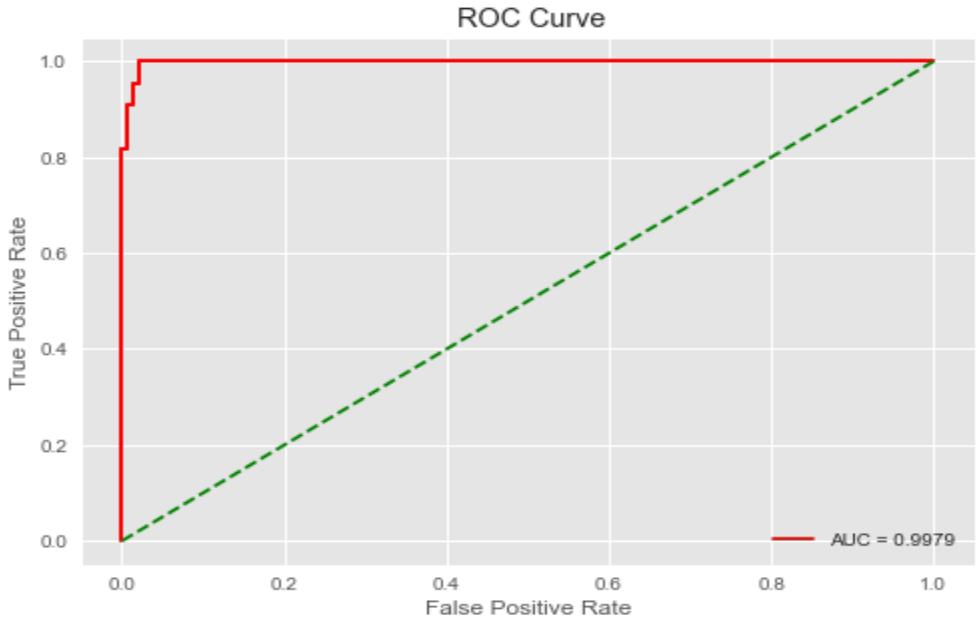
i. The model gave very high performance on both classes.

ii. AUC showed an accuracy of 94.7%.

e. Support Vector Machine.

```
Best parameter for SVM
{'C': 1000, 'gamma': 0.01, 'kernel': 'rbf'}
```





Findings:

i. The model gave very high performance on both classes.
ii. AUC showed very high accuracy of 99.7%.

---

5. Conclusion.

- SVM and Decision Tree are the best two models.
- SVM has a slightly higher performance in both classes
- SVM has a higher AUC accuracy with 99.7% than Decision Tree with 94.7%.
- SVM is the best model for this problem.