Covid-19 #: Final-Project

Done by: Esraa Khaled Ahmed Fouad Omar # ID : 20399123

February 27, 2023

Contents

1 Data

As we know our data contains 14 major variables that will be having an impact on whether someone has recovered or not, the description of each variable are as follows.

1. Country: where the person resides
2. Location: which part in the Country
3. Age: Classification of the age group for each person, based on WHO Age Group Standard
4. Gender: Male or Female
5. Visited _Wuhan: whether the person has visited Wuhan, China or not
6. From_Wuhan: whether the person is from Wuhan, China or not
7. Symptoms: there are six families of symptoms that are coded in six fields.
8. Time_before _symptoms_appear:
9. Result: death (1) or recovered (0)

We need to know more about our data .

(a) Let's see sample of our data

	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

Figure 1: Covid-19 _dataset

As you see from figure 1, all Features are numerical and our target is binary 0 and 1.

(b) Let's describe our data to know more about the distribution of data

	location	country	gender	age	vis_wuhan	from_wuhan	symptom1	symptom2	symptom3	symptom4	symptom5	symptom6	diff_sym_hos	result
count	695.000000	695.000000	695.000000	695.000000	695.000000	695.000000	695.000000	695.000000	695.000000	695.000000	695.000000	695.000000	695.000000	695.000000
mean	75.965468	16.477698	0.686331	49.507338	0.205755	0.128058	11.700719	27.277698	18.129496	11.801439	2.991367	0.998561	1.234532	0.152518
std	38.605392	7.468802	0.606901	16.506778	0.404544	0.334395	4.332664	8.165031	3.168729	1.316378	0.141768	0.037932	2.571953	0.359781
min	0.000000	0.000000	0.000000	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-5.000000	0.000000
25%	45.000000	8.000000	0.000000	37.000000	0.000000	0.000000	6.000000	31.000000	19.000000	12.000000	3.000000	1.000000	0.000000	0.000000
50%	83.000000	18.000000	1.000000	49.400000	0.000000	0.000000	14.000000	31.000000	19.000000	12.000000	3.000000	1.000000	0.000000	0.000000
75%	108.000000	23.000000	1.000000	60.000000	0.000000	0.000000	14.000000	31.000000	19.000000	12.000000	3.000000	1.000000	1.000000	0.000000
max	138.000000	33.000000	2.000000	96.000000	1.000000	1.000000	24.000000	31.000000	19.000000	12.000000	3.000000	1.000000	15.000000	1.000000

Figure 2: Distribution of data set

As you can see the data isn't on the a similar scale. The Location and Age have approximately the same scale but they have a large scale compared to other features. Also, the country, symptom1, symptom2, symptom3 and diff_sym_hos are on the same scale but still they are large scale compared to the rest features. It's a **first problem** in our data and we will try to solve it later in the following section. This improves the performance of the model.

(c) Checking the Missing data is important part. We should validate that there is no missing in the data. As shown in the below figure, There is no missing in our data.

```
print('The number of missing values in each column ' )
print(Covid_data.isna().sum()) # Thanks to Allah, There is no missing in our data
```

```
The number of missing values in each column
location        0
country         0
gender          0
age             0
vis_wuhan       0
from_wuhan      0
symptom1        0
symptom2        0
symptom3        0
symptom4        0
symptom5        0
symptom6        0
diff_sym_hos    0
result          0
dtype: int64
```

Figure 3: The number of missing values in each column

(d) when we check the duplication in the data we found a bout 168 duplicated rows in the data. we will handle in the following part. It's a **second problem**

```
print('The number of missing values in each column ' )
print(Covid_data.isna().sum()) # Thanks to Allah, There is no missing in our data
print('The number of duplications in data  = ', Covid_data.duplicated().sum() ) # Unfortunately, we have about 168 duplicated rov
df = Covid_data.drop_duplicates() # Let's dor duplicated  row from our data
print(" the shape of the data after droping duplicated row  = ", df.shape)
```

```
The number of missing values in each column
location        0
country         0
gender          0
age             0
vis_wuhan       0
from_wuhan      0
symptom1        0
symptom2        0
symptom3        0
symptom4        0
symptom5        0
symptom6        0
diff_sym_hos    0
result          0
dtype: int64
The number of duplications in data  =  168
 the shape of the data after droping duplicated row  =  (695, 14)
```

Figure 4: The number of duplicated rows in the data

(e) We notice that there is some of negative values in the diff_sym _hos and we know that this column result time .So, it is not valid to be negative values in it. It 's the **third problem**. Also, we will see how we could handle it.

```
print("Count negaitive values in diff_sym_hos column = " , (df.diff_sym_hos < 0).any().any().sum()
```

```
Count negaitive values in diff_sym_hos column =  1
```

Figure 5: Count negative values in diff_sym _hos column

(f) The most important part is checking the imbalance in the target column. As this effects our model later make bias to the class that has a highest count.So, let's check it in our target column "result". Unfortunately, there is an imbalance in the result column. Class 1 only has **106** samples and Class 0 has **587** samples. It's a **fourth problem**.we will handle it in the following section.

```
print("the frequency of each unique value in a result column " , df['result'].value_counts())
ax = sns.countplot(x='result',data=df, palette="PuRd")
plt.show()
```

```
the frequency of each unique value in a result column  0    587
1    106
Name: result, dtype: int64
```

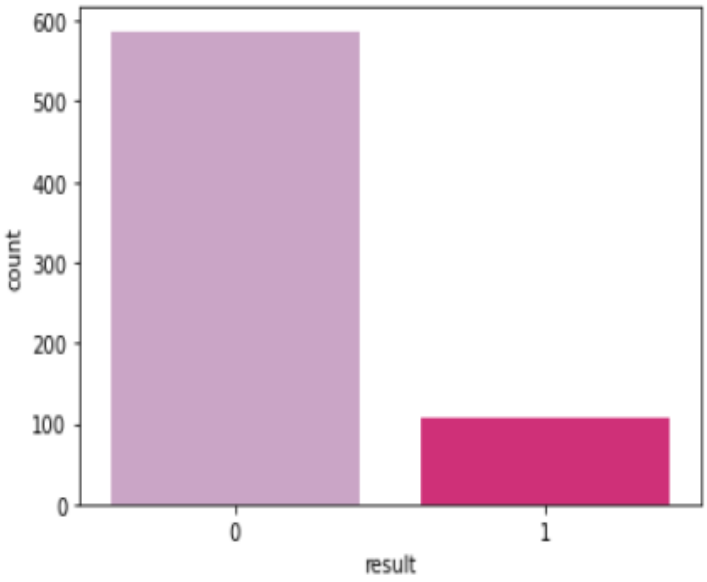

Figure 6: The Frequency of each unique value in result column

2 Prepare data

In this section, we will handle all the problems we mentioned in the previous.

(a) **Duplication** : We drop duplicated rows.

(b) **Negative Values** : we remove 2 rows that have negative values

(c) **imbalance** : we solve this problem using the most common method .we use smote to over sample the minority class.

(d) **Features are not on the same scale**
we use some visualization to know more about the relation between features and result. from the below visualization, we found that age ,location ,and Country are continuous values and almost follow Gaussian distribution and the rest is discrete values. So,we scale age ,location ,and Country using standardization and encode symptom1, symptom2, symptom3, symptom4, symptom5, symptom6, and time using one hot encoding.

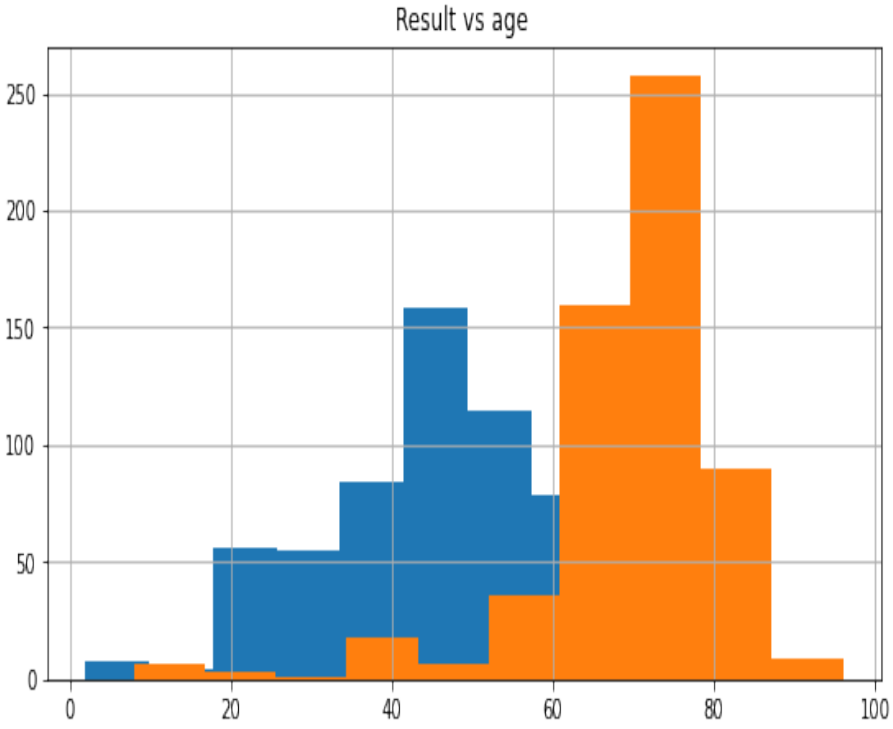

Figure 7: Age

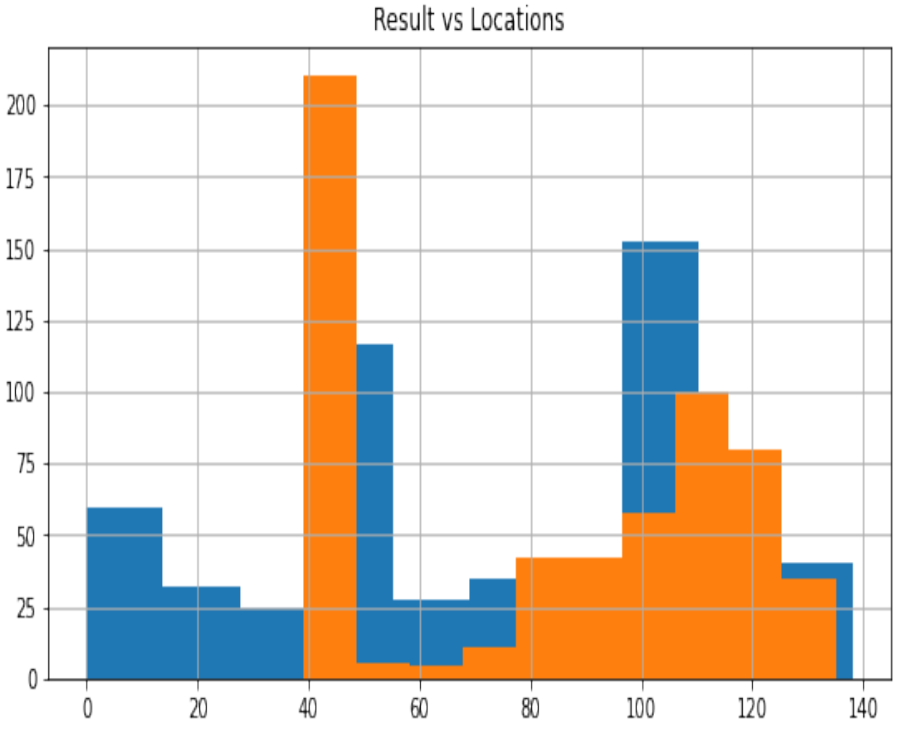

Figure 8: Locations

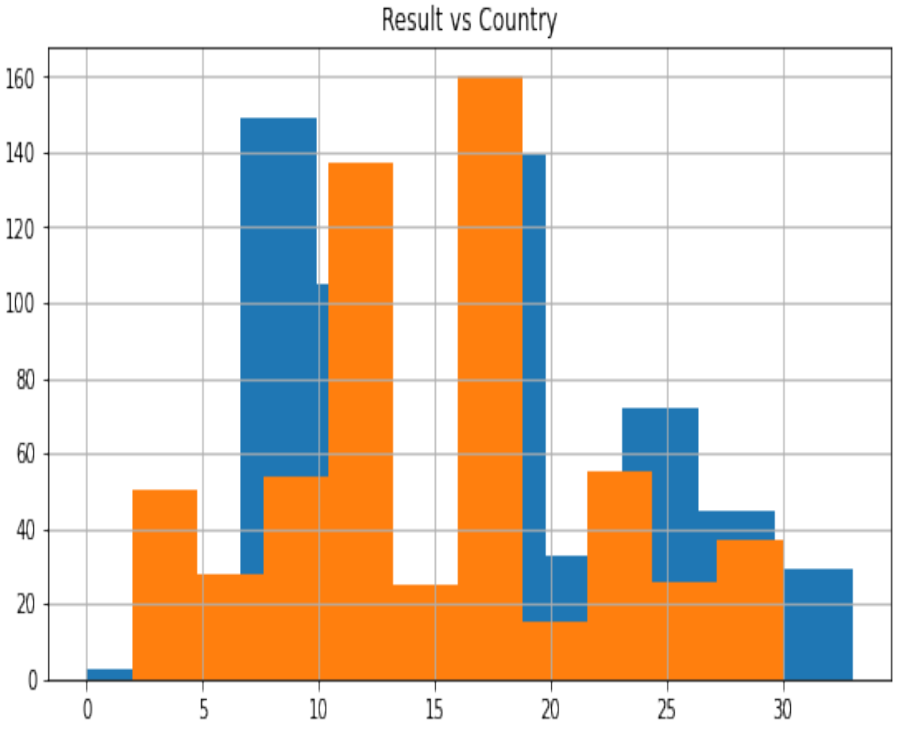

Figure 9: Country

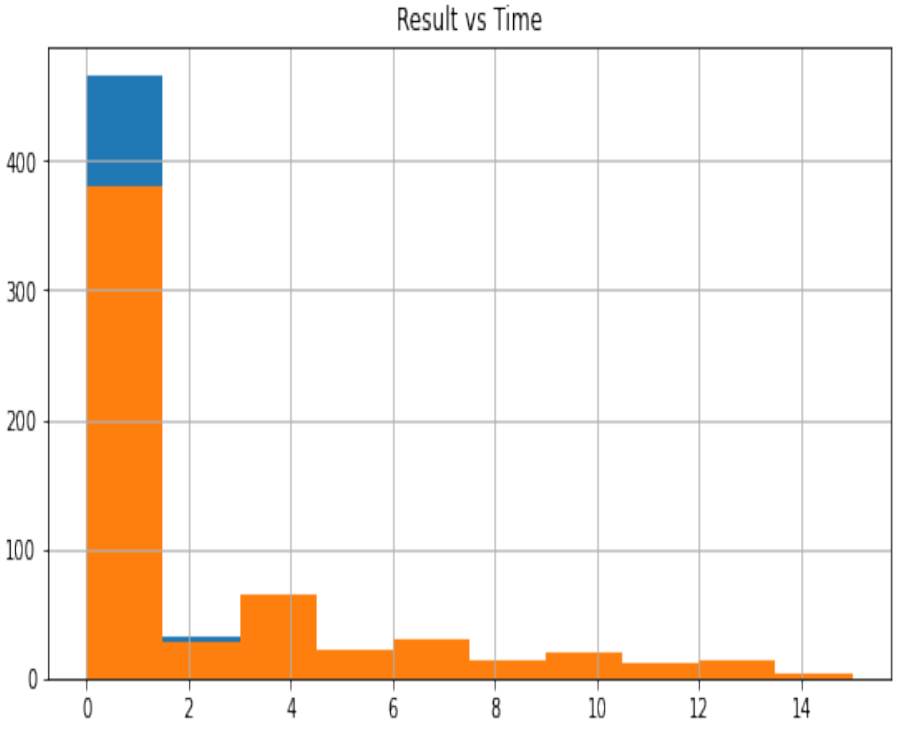

Figure 10: Time

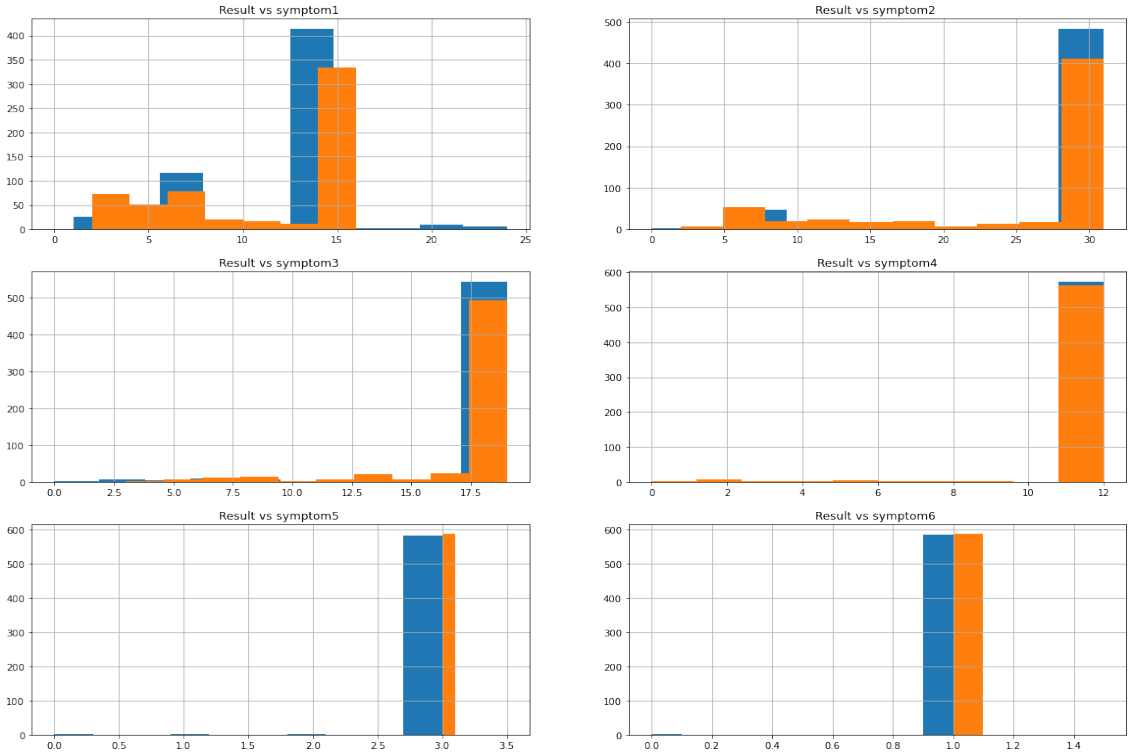

Figure 11: symptoms

3 Split Data

we split data into train and test. The train set represents 90 % of actual data and the test set represents 10 % of actual data. Then,we split the train into 2 partitions: train set and validation test internally in GridSearchCV to tune optimal hyper parameters.

4 Tune Optimal Hyper parameters

In our project, we try 5 classifiers : KNN, Logistic regression, Decision tree, Naive Bayes ,and SVM to choose the best one. Each model has different hyper parameters. we need to search for the best combination of hyper parameters for each model. There is a lot of ways to do that. Here, we use the grid search and cross validation using GridSearchCV to try different combination of hyper parameters and return the best one. Then, use them to train model.

Let's see sample of hyper parameters for each model that we use.The rest of hyper parameters are in the notebook you can see them.

Classifiers	Hyper-Parameters	Values For Each Hyper-Parameter
KNN		
	• n_neighbors :Number of neighbors to use. possible values are any integer value • p : Power parameter for the Minkowski metric. possible values are [0 , 1 , 2] • weights : Weight function used in prediction. possible values are [‘uniform’, ‘distance’].	• n_neighbors: range(1, 21, 2) • p : [3 , 2 , 1] • weights : [uniform, distance]
Logistic Regression	• penalty: Specify the norm of the penalty .Possible Values are [l1, l2, elasticnet, none]. • solver : Algorithm to use in the optimization problem. Possible values are [lbfgs, liblinear,newton-cg,newton-cholesky, sag, saga] • C : Inverse of regularization strength; must be a positive float.Like in support vector machines, smaller values specify stronger regularization.Possible values are any integer values. • tol: Tolerance for stopping criteria.Possible values are any float values. • class_weight : Weights associated with classes in the form class_label: weight. If not given, all classes are supposed to have weight one.Possible values are [dict , balanced, none]	• penalty: [l1, l2, elasticnet, none] • solver : [lbfgs, liblinear,newton-cg, newton-cholesky,sag, saga] • C : range(1, 21, 1) • tol: [x / 100 for x in range(0, 10)] • class_weight :[dict , balanced, none]
Naive Bayes	• var_smoothing: Portion of the largest variance of all features that is added to variances for calculation stability.Possible values are any float values (default=$1e^9$).	• var_smoothing:

Table 1: Hyper parameters used in this project

4.1 Best Hyper-parameters

After tuning each model using the previous combinations of different hyper-parameters, we got the best hyper-parameters for each model that achieve highest **f1score**.As you know that our problem is medical and we care that our model to be sensitive to class 1. So, we choose f1score as it combine recall and precision. we need model with high recall for class 1 .Also, we can't ignore that we need acceptable precision. f1score trades off between recall and precision.

for each model, we try different cases :

1. **Case 1** : Assume that the data is cleaned as mentioned in the statement and tune model in the data without any applying prepossessing method.
2. **Case 2** : Prepare data and apply the methods that were mentioned before and tune model in the prepared data **except** solving imbalance in the target column
3. **Case 3** : Prepare data and apply all the method that were mentioned before and tune model in the prepared data.

For each case, we will visualize the precision, recall ,f1score and ROC/AUC curve for each model.

4.1.1 KNN

When tune knn model on 3 different cases, we got the following :

1. **Case 1 :**

```
Best Params:  {'n_neighbors': 5, 'p': 3, 'weights': 'distance'}
Best Score:  0.7955187303069202
Test set accuracy: 0.9080459770114943
Train set accuracy: 1.0
AUC: 0.8343301435406698  precision :  0.7142857142857143  recall :  0.45454545454545453 FScore :  0.5555555555555556
```

Figure 12: Best Hyper parameters for Knn (case1)

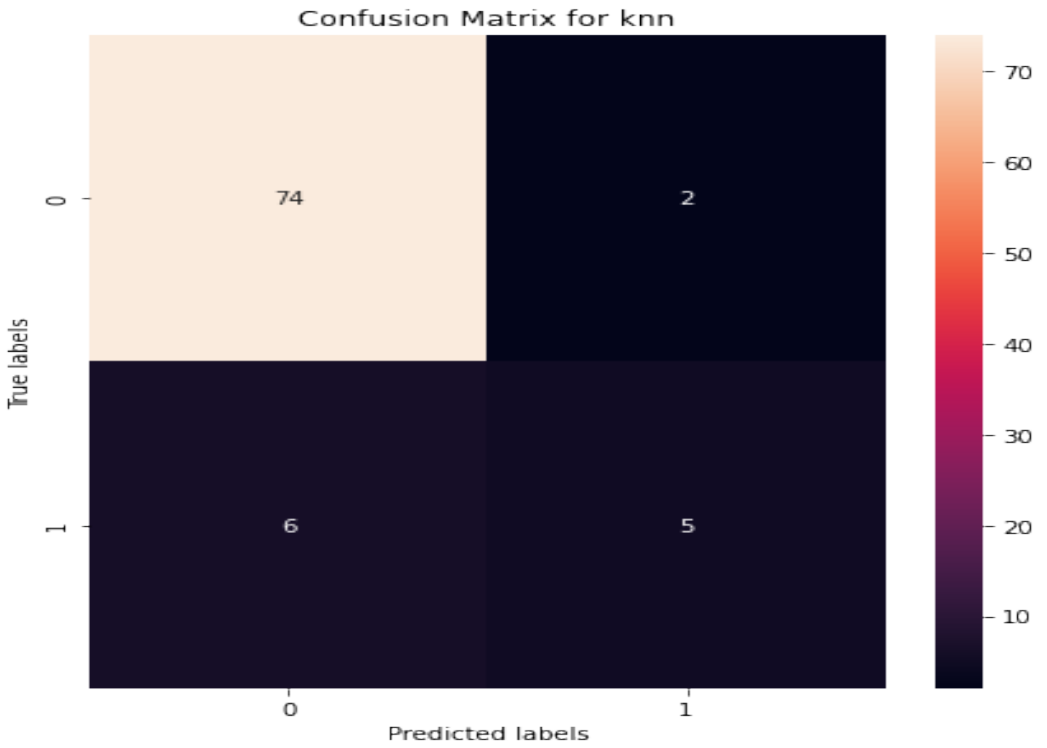

Figure 13: Confusion matrix for Knn (case1)

As expected, the model went worse. The true Fact is GIGO. As you can see without any prepossessing in our data, we got a poor classifier. The model is over fitting as test error is large.The difference between accuracy in train and test is large as shown in figure (13) . Also, recall and f1score are very low. we can't choose this model to solve our problem .

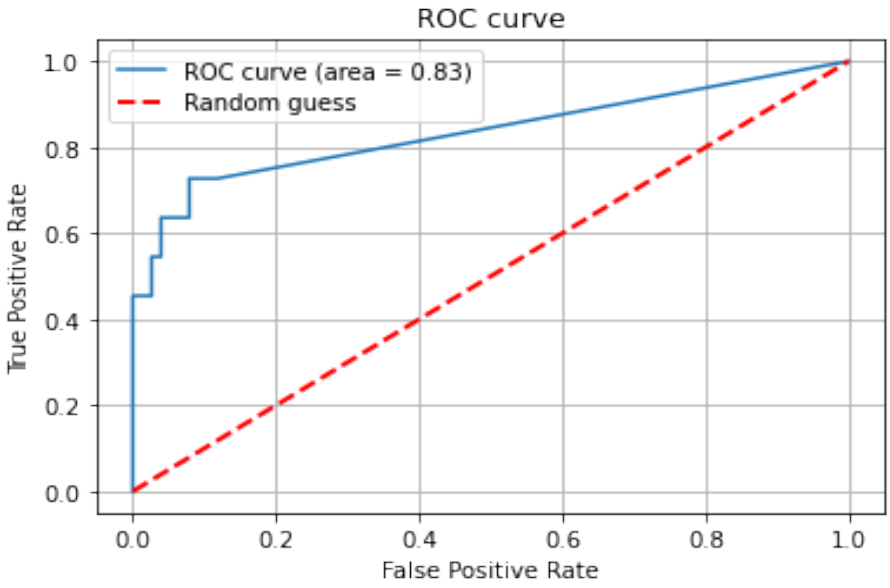

Figure 14: ROC Curve for Knn (case1)

classification_report of knn

	precision	recall	f1-score	support
0	0.93	0.97	0.95	76
1	0.71	0.45	0.56	11
accuracy			0.91	87
macro avg	0.82	0.71	0.75	87
weighted avg	0.90	0.91	0.90	87

Figure 15: Classification matrix for Knn (case1)

Classification report shows us that the model performs well in class 0 but fails in class 1 .Due to imbalance in the data, our model bias towards the high Cardinality class (class 0).

2. **Case 2 :**

```
Best Params:  {'n_neighbors': 7, 'p': 3, 'weights': 'distance'}
Best Score:  0.7671713668782306
Test set accuracy: 0.9285714285714286
Train set accuracy: 1.0
AUC: 0.9784283513097073  precision :  1.0  recall :  0.5454545454545454 FScore :  0.7058823529411764
```

Figure 16: Best Hyper parameters for Knn (case2)

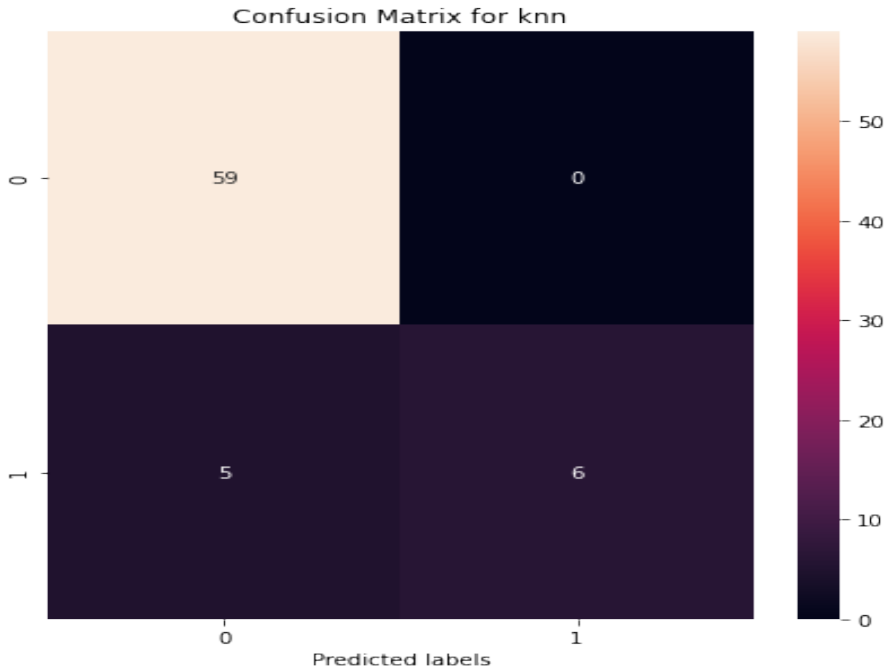

Figure 17: Confusion matrix for Knn (case2)

As expected, the model went worse. The imbalance has a bad impact in the model as shown in the figure(16). The model is over fitting as test error is large.The difference between accuracy in train and test is large as shown in figure (16) . Also, recall and f1score are still very low. we can't choose this model to solve our problem. Case 1 and 2 can't solve our problem. Let's see our new model after applying prepossessing.

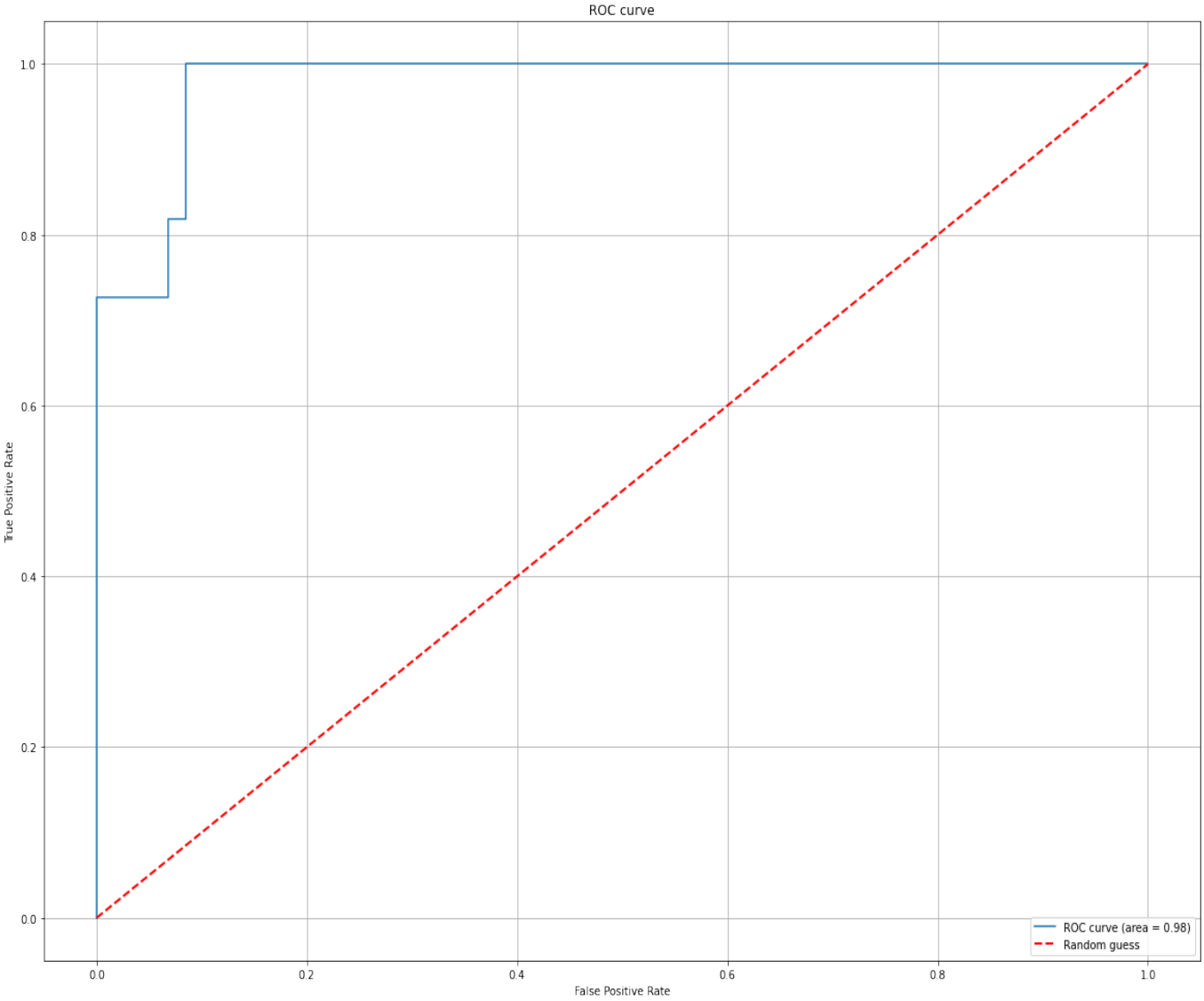

Figure 18: ROC Curve for Knn (case2)

```
classification_report of knn
              precision    recall  f1-score   support

           0       0.92      1.00      0.96        59
           1       1.00      0.55      0.71        11

    accuracy                           0.93        70
   macro avg       0.96      0.77      0.83        70
weighted avg       0.93      0.93      0.92        70
```

Figure 19: Classification matrix for Knn (case2)

Classification report shows us that the model performs well in class 0 but fails in class 1 .Due to imbalance in the data, our model bias towards the high Cardinality class (class 0). we can't deny that it is better than case 1 but still is not the best one.

3. **case3**

```
Best Params:  {'n_neighbors': 11, 'p': 2, 'weights': 'distance'}
Best Score:  0.9399642661065553
Test set accuracy: 0.9067796610169492
Train set accuracy: 1.0
AUC: 0.9640907785119218  precision :  0.875  recall :  0.9491525423728814 FScore :  0.9105691056910569
```

Figure 20: Best Hyper parameters for Knn (case 3)

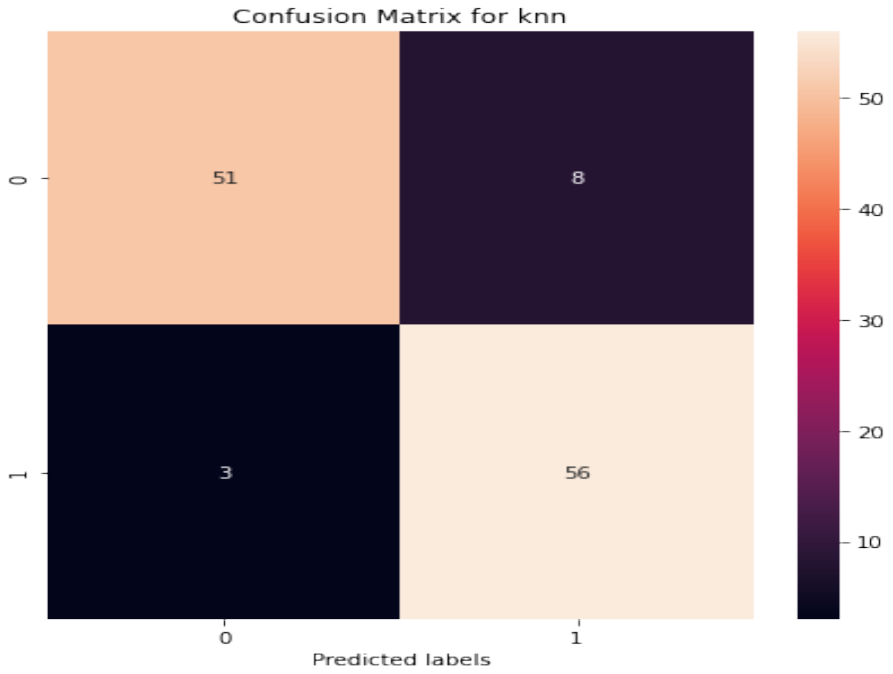

Figure 21: Confusion matrix for Knn (case 3)

The over fitting still exists.we can justify that :

(a) the data is very low . we have only 693 row after removing duplicates and negative values

(b) after applying prepossessing, we generated a lot of features. Increasing Features may cause over fitting.

(c) the method we use to solve imbalance may be not suitable.

However, we can say that this case is the best compared to the previous ones.As you can see, it has highest recall (94 %). It's acceptable for our problem. Also, f1score is high (90 %).

We can't choose our model based on auc as you see, all the model has high aus but doesn't mean that they are the best classifiers.

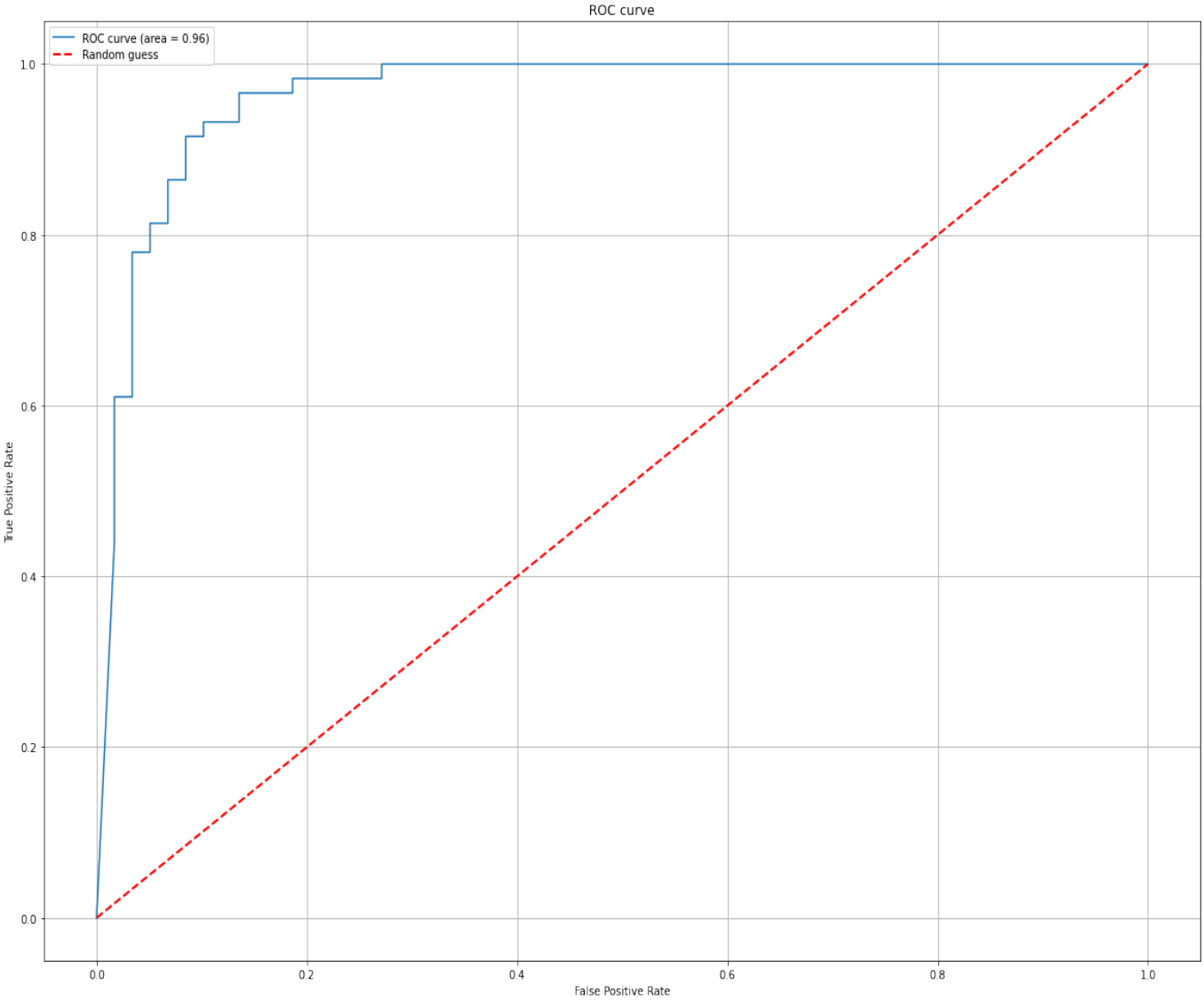

Figure 22: ROC Curve for Knn (case 3)

classification_report of knn

	precision	recall	f1-score	support
0	0.94	0.86	0.90	59
1	0.88	0.95	0.91	59
accuracy			0.91	118
macro avg	0.91	0.91	0.91	118
weighted avg	0.91	0.91	0.91	118

Figure 23: Classification matrix for Knn (case 3)

Classification report shows us that the model is relatively good not bais into certain class.

Let's move on the other different classifier to see if knn (case 3) still is the best or not.

4.1.2 Logistic Regression

When tune Logistic Regression model on 3 different cases, we got the following :

1. **Case 1 :**

```
Best Params:  {'C': 11, 'class_weight': 'dict', 'penalty': 'l2', 'solver': 'newton-cg', 'tol': 0.09}
Best Score:  0.7755555555555556
Test set accuracy: 0.9080459770114943
Train set accuracy: 0.9536082474226805
AUC: 0.9677033492822967  precision :  0.7142857142857143  recall :  0.45454545454545453 FScore :  0.5555555555555556
```

Figure 24: Best Hyper parameters for Logistic Regression (case1)

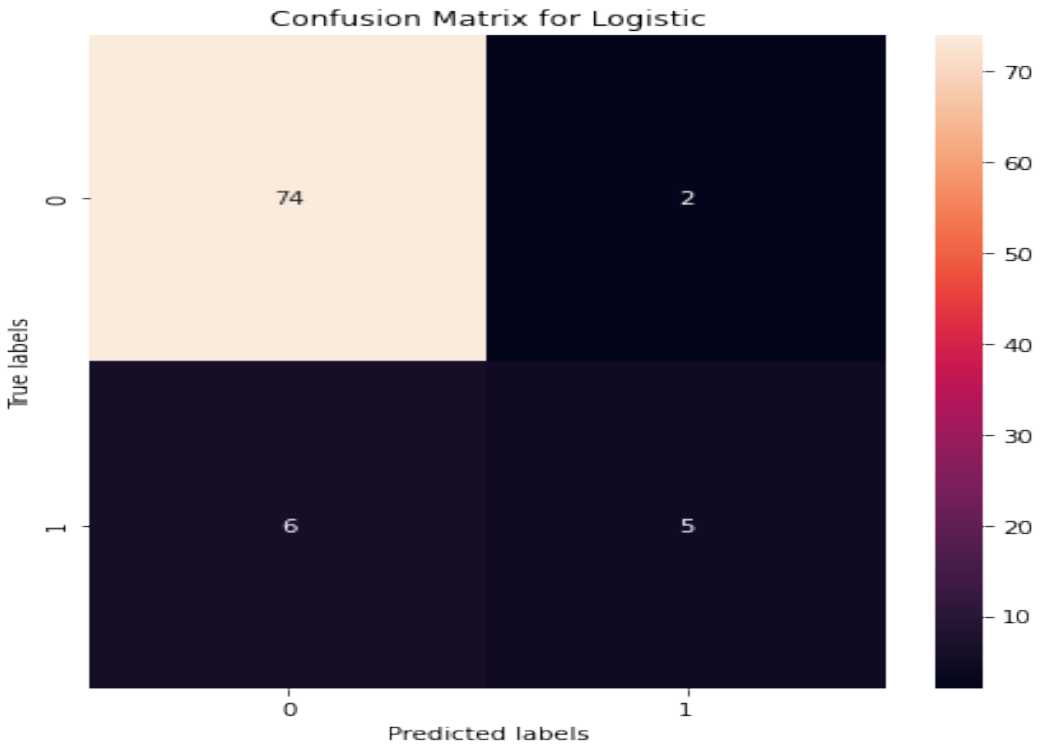

Figure 25: Confusion matrix for Logistic Regression (case1)

Here, we can say that the logistic model succeeded in decreasing the over fitting as shown from above figure. the model could solve this over fitting through l2 penalty and tolerance(stopping criteria). But the model still doesn't achieve an acceptable recall or f1score.

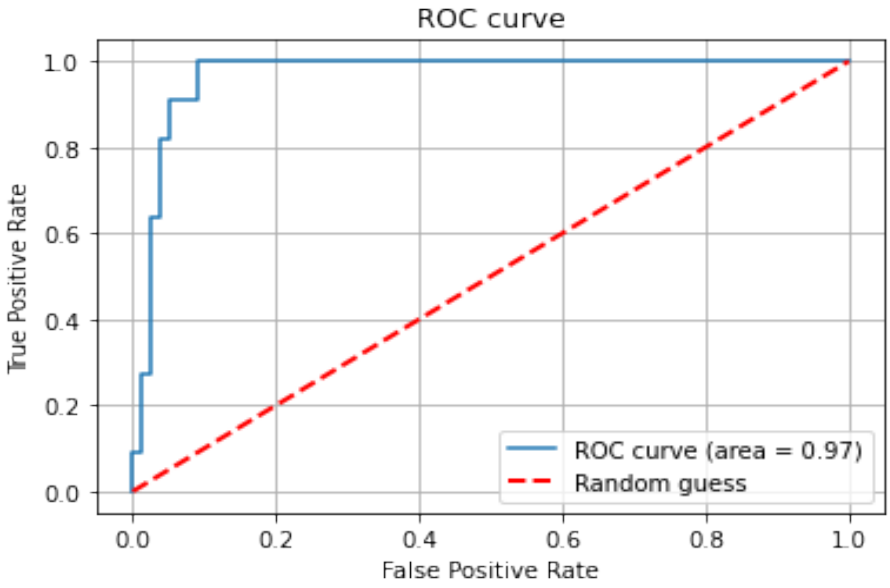

Figure 26: ROC Curve for Logistic Regression (case1)

classification_report of Logistic

	precision	recall	f1-score	support
0	0.93	0.97	0.95	76
1	0.71	0.45	0.56	11
accuracy			0.91	87
macro avg	0.82	0.71	0.75	87
weighted avg	0.90	0.91	0.90	87

Figure 27: Classification matrix for Logistic Regression (case1)

Classification report shows us that the model performs well in class 0 but fails in class 1 .Due to imbalance in the data, our model bias towards the high Cardinality class (class 0).Knn (Case 3) is still the best one.

2. **Case 2 :**

```
Best Params:  {'C': 3, 'class_weight': 'none', 'penalty': 'none', 'solver': 'sag', 'tol': 0.08}
Best Score:  0.8053373946147486
Test set accuracy: 0.9
Train set accuracy: 0.9518459069020867
AUC: 0.9738058551617874  precision :  0.6666666666666666  recall :  0.7272727272727273 FScore :  0.6956521739130435
```

Figure 28: Best Hyper parameters for Logistic Regression (case2)

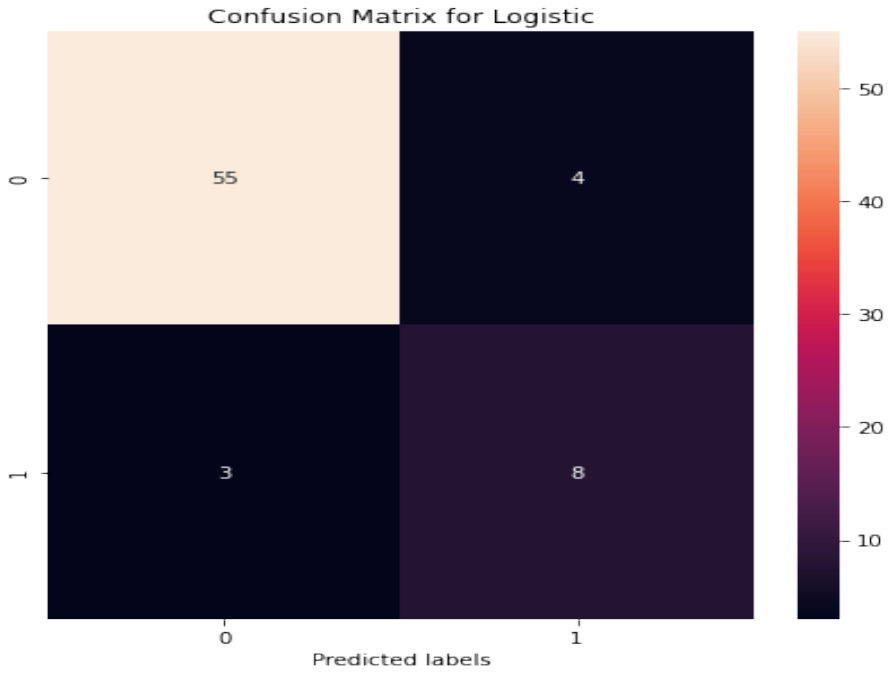

Figure 29: Confusion matrix for Logistic Regression (case2)

No significant difference between this case and previous cases.Only the recall and f1score are relatively improved.

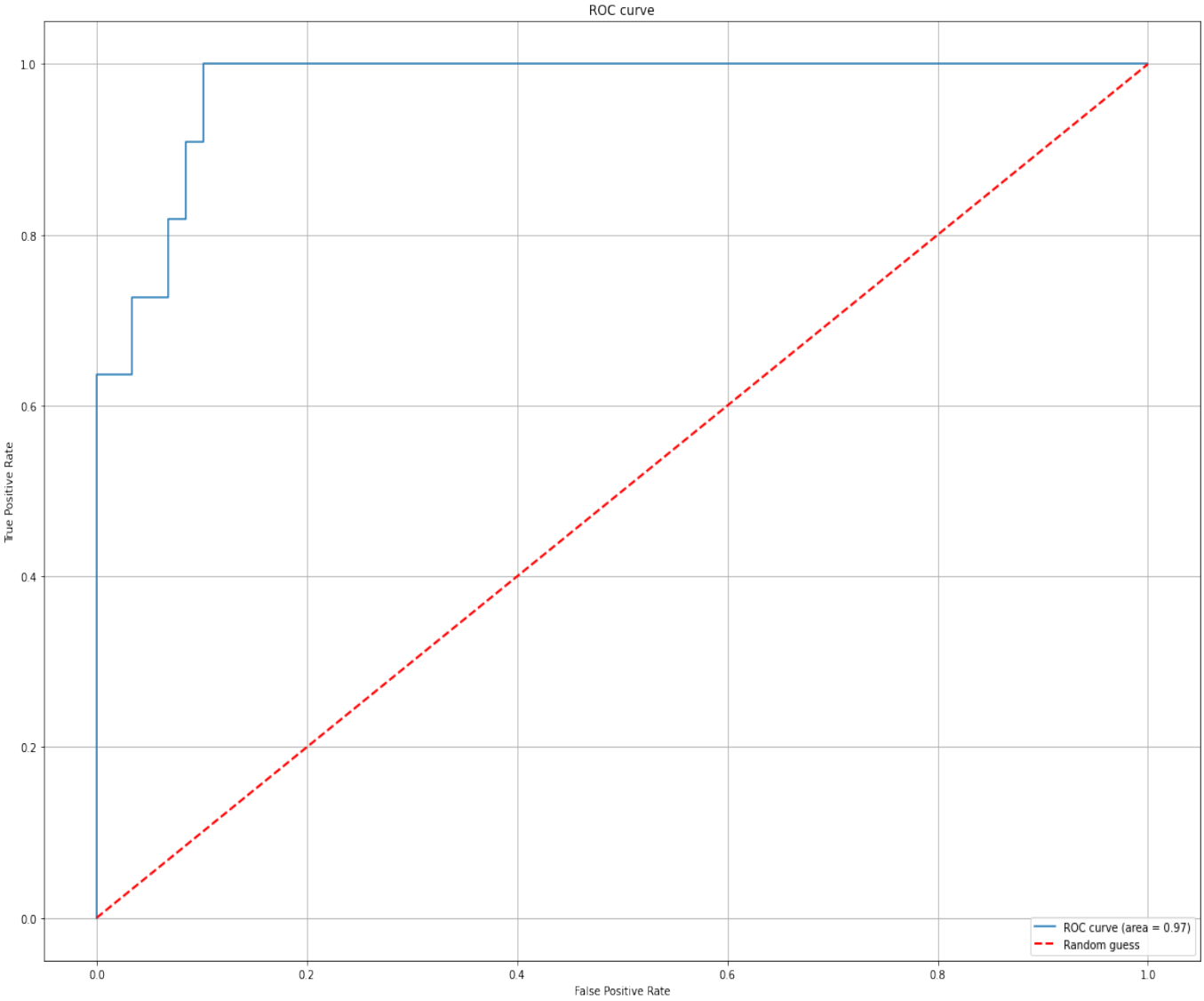

Figure 30: ROC Curve for Knn (case2)

classification_report of Logistic

	precision	recall	f1-score	support
0	0.95	0.93	0.94	59
1	0.67	0.73	0.70	11
accuracy			0.90	70
macro avg	0.81	0.83	0.82	70
weighted avg	0.90	0.90	0.90	70

Figure 31: Classification matrix for Logistic Regression (case2)

Classification report shows us that the model performs well in class 0 but fails in class 1 .Due to imbalance in the data, our model bias towards the high Cardinality class (class 0). we can't deny that it is better than case 1 but still is not the best one.

3. **case3**

```
Best Params:  {'C': 8, 'class_weight': 'none', 'penalty': 'none', 'solver': 'sag', 'tol': 0.09}
Best Score:  0.9344399038078576
Test set accuracy: 0.8898305084745762
Train set accuracy: 0.9441287878787878
AUC: 0.9143924159724217  precision :  0.8833333333333333  recall :  0.8983050847457628 FScore :  0.8907563025210085
```

Figure 32: Best Hyper parameters for Logistic Regression (case3)

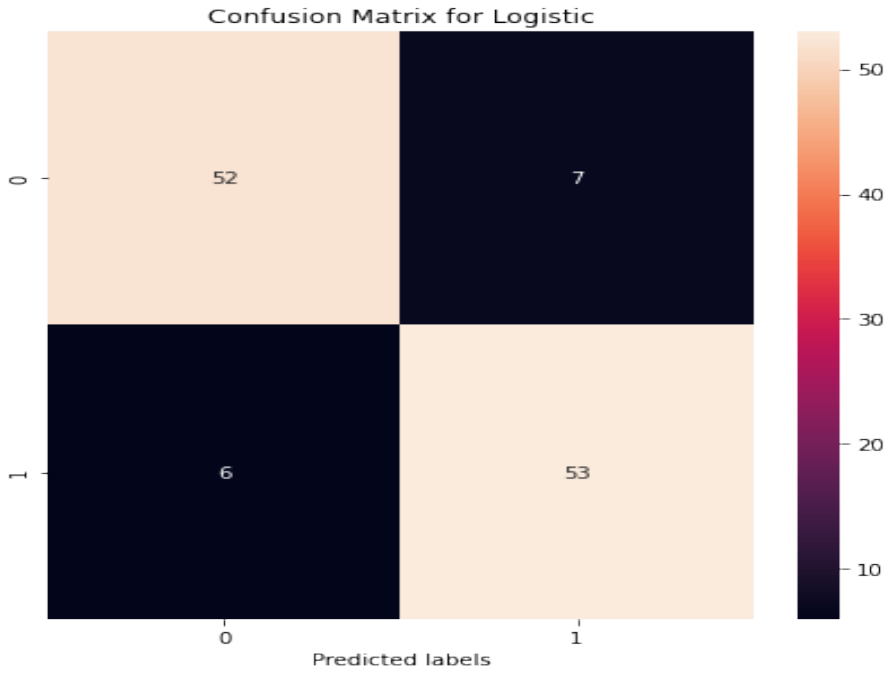

Figure 33: Confusion matrix for Logistic Regression (case3)

Recall ,precision ,f1score and auc are improved. we can say that case 3 (Logistic Regression) is the best compared to the previous ones of Logistic Regression cases.

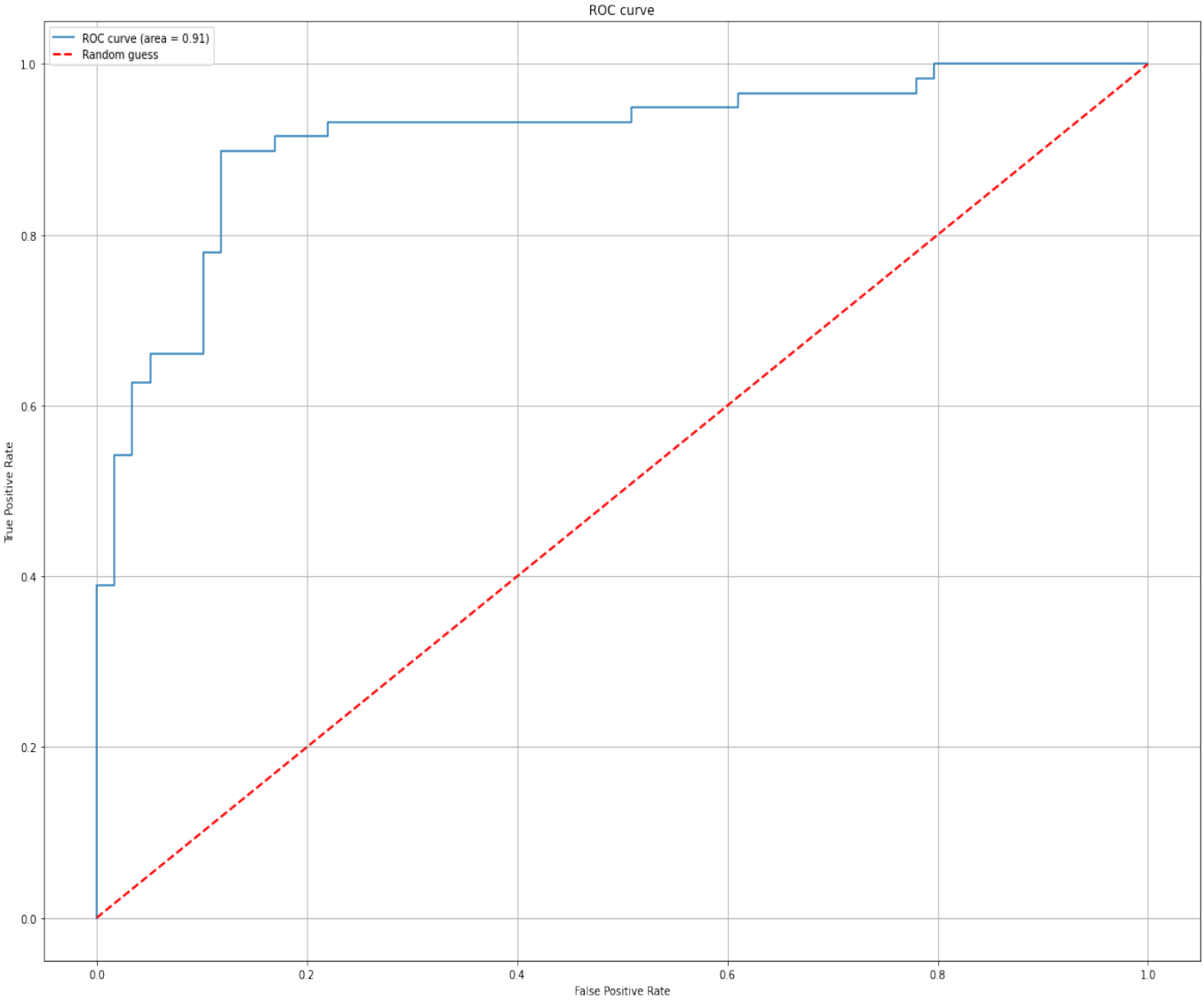

Figure 34: ROC Curve for Logistic Regression (case3)

classification_report of Logistic

	precision	recall	f1-score	support
0	0.90	0.88	0.89	59
1	0.88	0.90	0.89	59
accuracy			0.89	118
macro avg	0.89	0.89	0.89	118
weighted avg	0.89	0.89	0.89	118

Figure 35: Classification matrix for Logistic Regression (case3)

Classification report shows us that the model is relatively good not bais into certain class. But sill **Knn (case 3)** has higher recall than **Logistic Regression(case 3)**.Let's move on.

4.1.3 Decision Tree

When tune Decision Tree model on 3 different cases, we got the following :

1. **Case 1 :**

```
Best Params:  {'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 8, 'max_features': 'auto', 'max_leaf_nodes': 20, 'min
_samples_leaf': 2, 'min_samples_split': 10, 'splitter': 'best'}
Best Score:  0.8479741917908813
Test set accuracy: 0.896551724137931
Train set accuracy: 0.9368556701030928
AUC: 0.7858851674641149  precision :  0.5714285714285714  recall :  0.7272727272727273 FScore :  0.64
```

Figure 36: Best Hyper parameters for Decision Tree (case1)

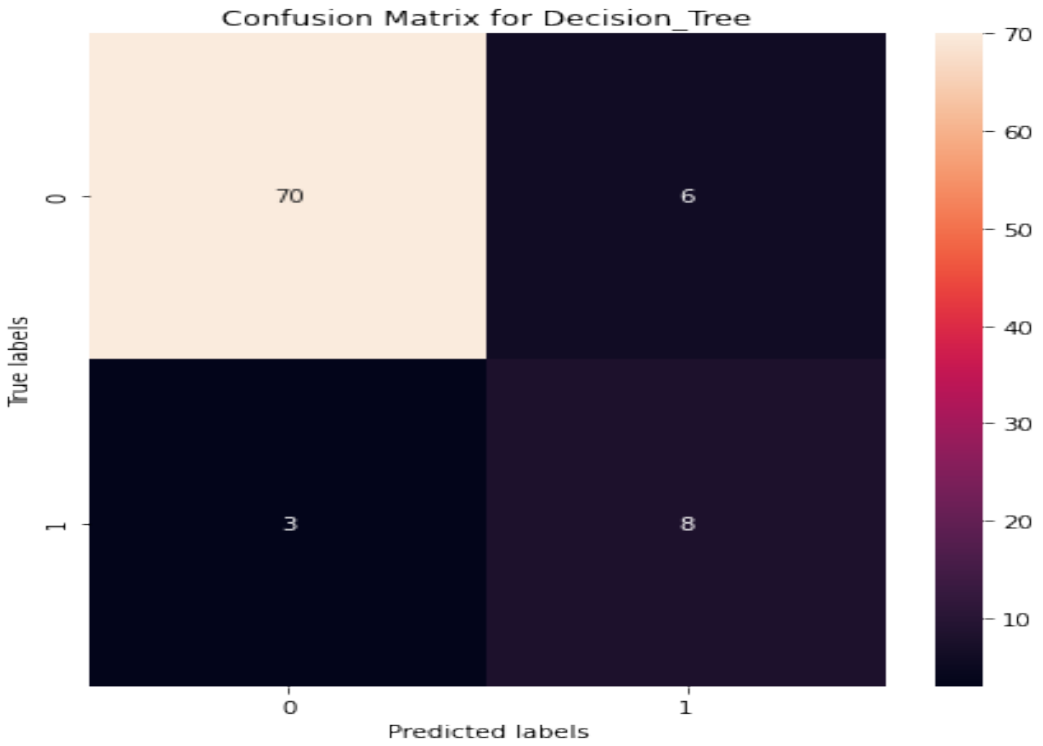

Figure 37: Confusion matrix for Decision Tree (case1)

As you can see, case 1 has a good recall compared to case 1 in knn and logistic.But we can't still accept it in our problem.Also, f1score and precision still are very low.That is very normal data need to prepossessing.

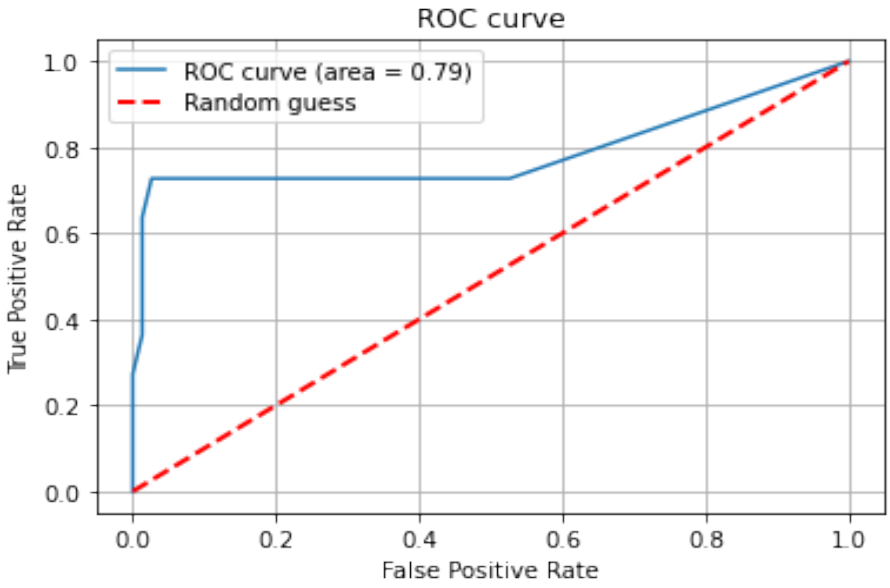

Figure 38: ROC Curve for Decision Tree (case1)

classification_report of Decision_Tree

	precision	recall	f1-score	support
0	0.96	0.92	0.94	76
1	0.57	0.73	0.64	11
accuracy			0.90	87
macro avg	0.77	0.82	0.79	87
weighted avg	0.91	0.90	0.90	87

Figure 39: Classification matrix for Decision Tree (case1)

Classification report shows us that the model performs well in class 0 but fails in class 1 .Due to imbalance in the data, our model bias towards the high Cardinality class (class 0).

2. **Case 2 :**

```
Best Params:  {'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 6, 'max_features': 'sqrt', 'max_leaf_nodes': 10, 'min_samples_leaf': 1, 'min_s
amples_split': 50, 'splitter': 'best'}
Best Score:  0.6931764705882353
Test set accuracy: 0.4
Train set accuracy: 0.5152487961476726
AUC: 0.652542372881356  precision :  0.19607843137254902  recall :  0.9090909090909091 FScore :  0.3225806451612903
```

Figure 40: Best Hyper parameters for Decision Tree (case2)

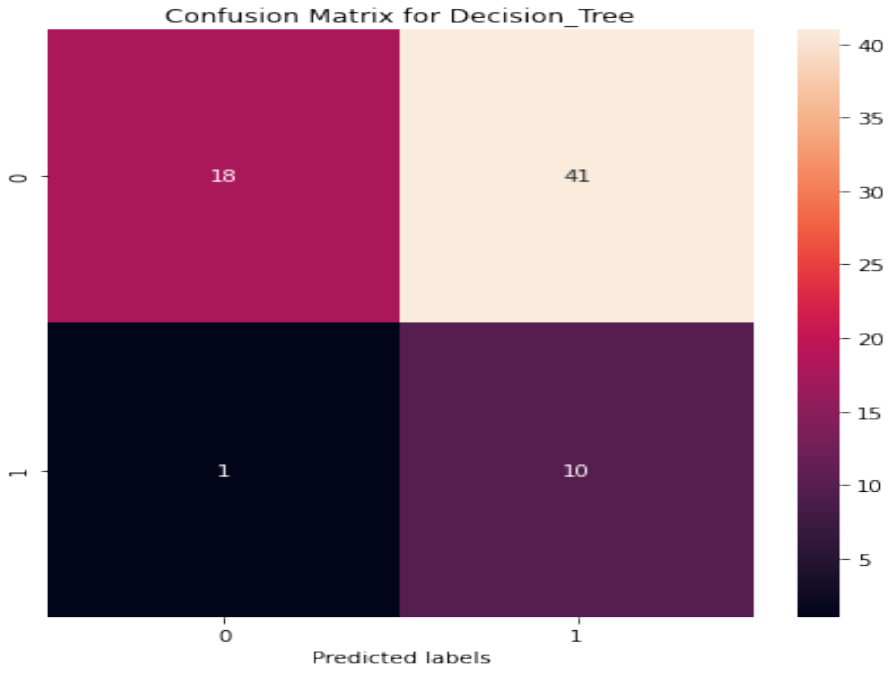

Figure 41: Confusion matrix for Decision Tree (case2)

The model went worse.Although the recall is relative high but f1score is very low. The model is under fitting.we can justify that As:

- we have a large number of feature(118). So, the depth of tree will be large. But we choose possible values of max _depth are in range from 1 to 10.

Also, increasing depth may cause over fitting.

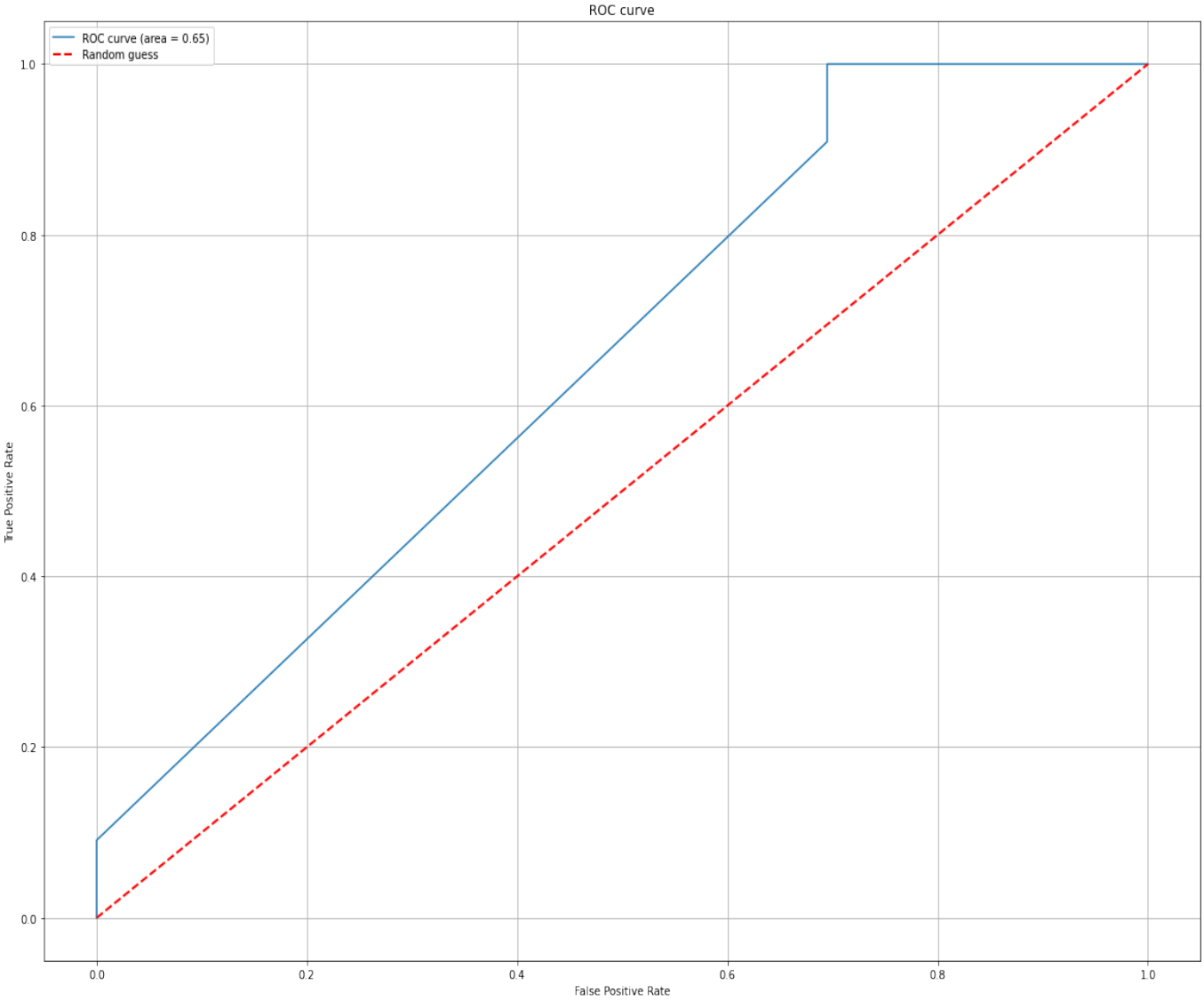

Figure 42: ROC Curve for Decision Tree (case2)

classification_report of Decision_Tree

	precision	recall	f1-score	support
0	0.95	0.31	0.46	59
1	0.20	0.91	0.32	11
accuracy			0.40	70
macro avg	0.57	0.61	0.39	70
weighted avg	0.83	0.40	0.44	70

Figure 43: Classification matrix for Decision Tree (case2)

Classification report shows us that the model bias towards class 1.

3. **case3**

```
Best Params:  {'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 6, 'max_features': 'auto', 'max_leaf_nodes': 10, 'min_samples_leaf': 1, 'mi
n_samples_split': 5, 'splitter': 'best'}
Best Score:  0.9108429692387437
Test set accuracy: 0.5932203389830508
Train set accuracy: 0.5823863636363636
AUC: 0.6308532031025568  precision :  0.7894736842105263  recall :  0.2542372881355932 FScore :  0.3846153846153846
```

Figure 44: Best Hyper parameters for Decision Tree (case3)

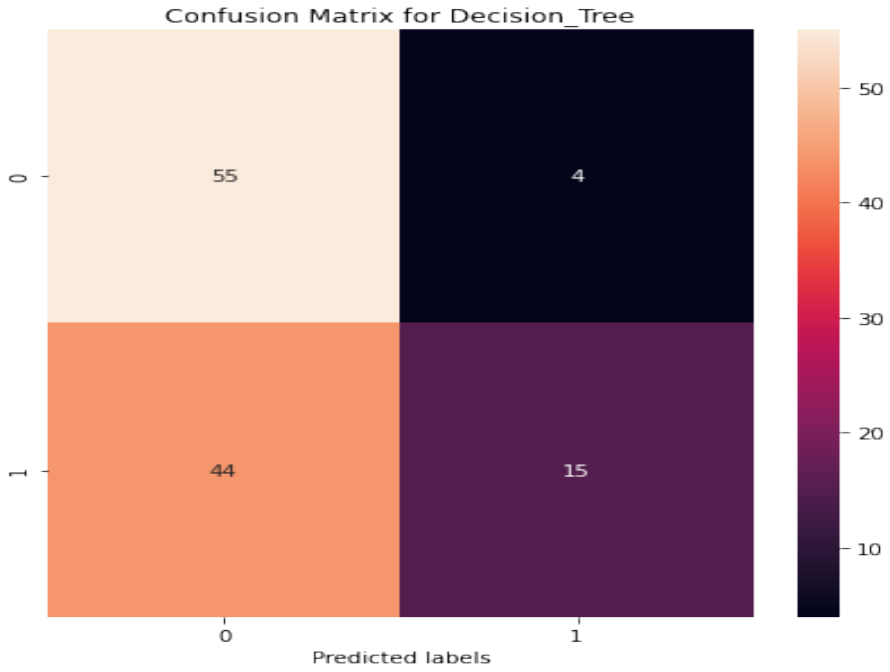

Figure 45: Confusion matrix for Decision Tree (case3)

The under fitting still exists.No significant difference in accuracy and f1score.
Note that:
the class_weight = balanced .So, the model in case 2 and case 3 solve the problem of imbalance. But the recall is decreased and precision is increased.That may be due to splitting of the data in cross validation.

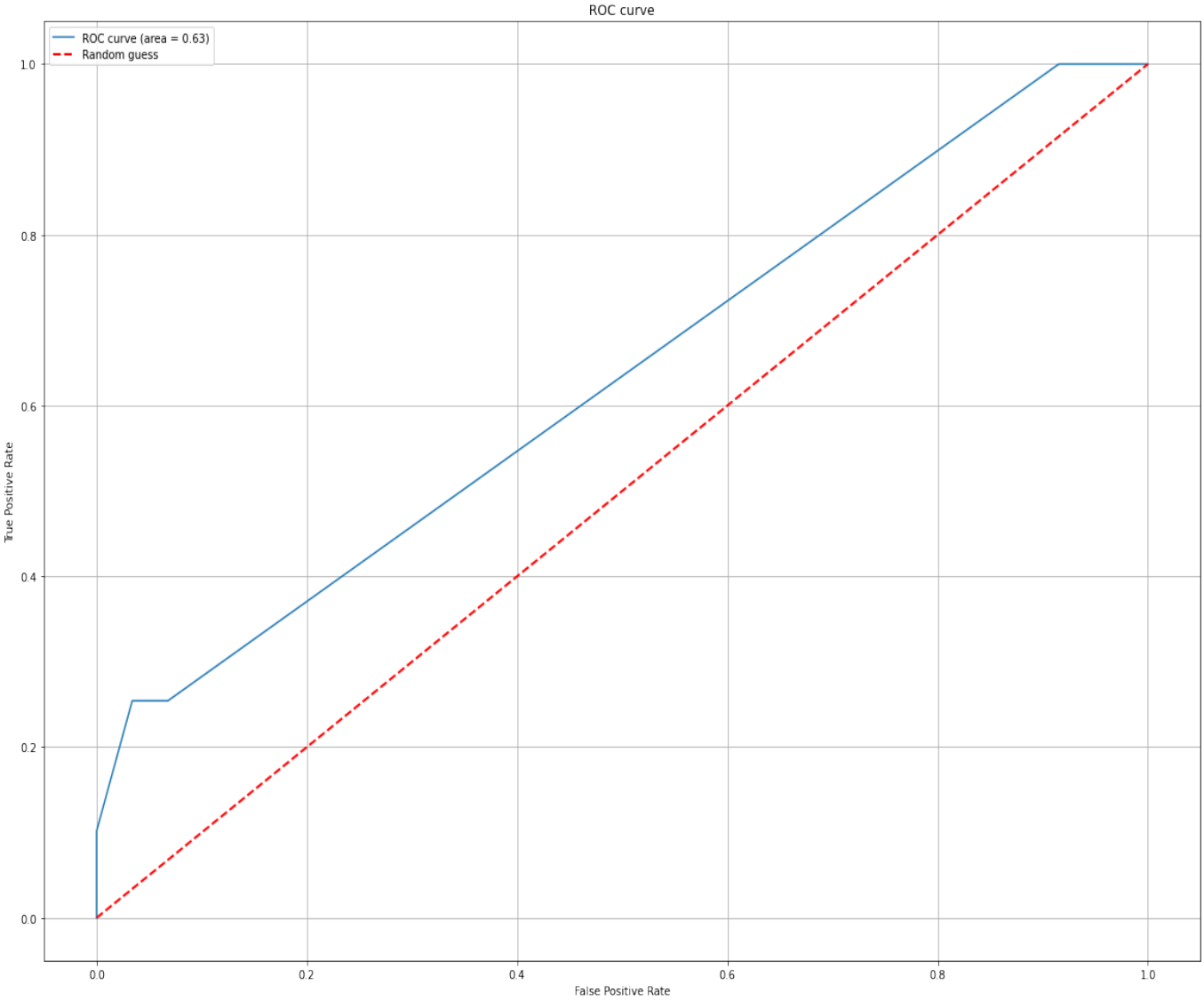

Figure 46: ROC Curve for Desicion Tree (case3)

classification_report of Decision_Tree

	precision	recall	f1-score	support
0	0.56	0.93	0.70	59
1	0.79	0.25	0.38	59
accuracy			0.59	118
macro avg	0.67	0.59	0.54	118
weighted avg	0.67	0.59	0.54	118

Figure 47: Classification matrix for Desicion Tree (case3)

Classification report shows us that the model bias towards class 0.

4.1.4 Naive Bayes

When tune Naive Bayes model on 3 different cases, we got the following :

1. **Case 1 :**

```
Best Params:  {'var_smoothing': 3.5111917342151275e-06}
Best Score:  0.5966815244201451
Test set accuracy: 0.8850574712643678
Train set accuracy: 0.875
AUC: 0.9294258373205742  precision :  0.5384615384615384  recall :  0.6363636363636364 FScore :  0.5833333333333334
```

Figure 48: Best Hyper parameters for Knn (case1)

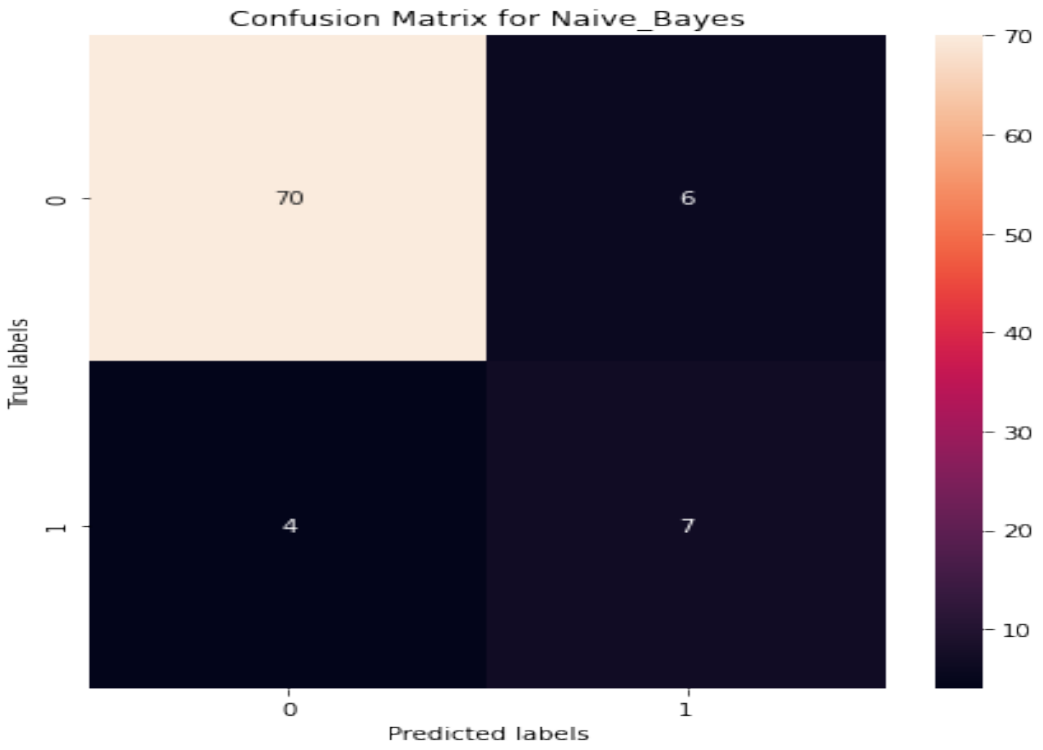

Figure 49: Confusion matrix for Naive Bayes (case1)

There is no over fitting. The difference between train and test accuracy isn't significant.Recall and f1score are very low. we can't choose this model to solve our problem .

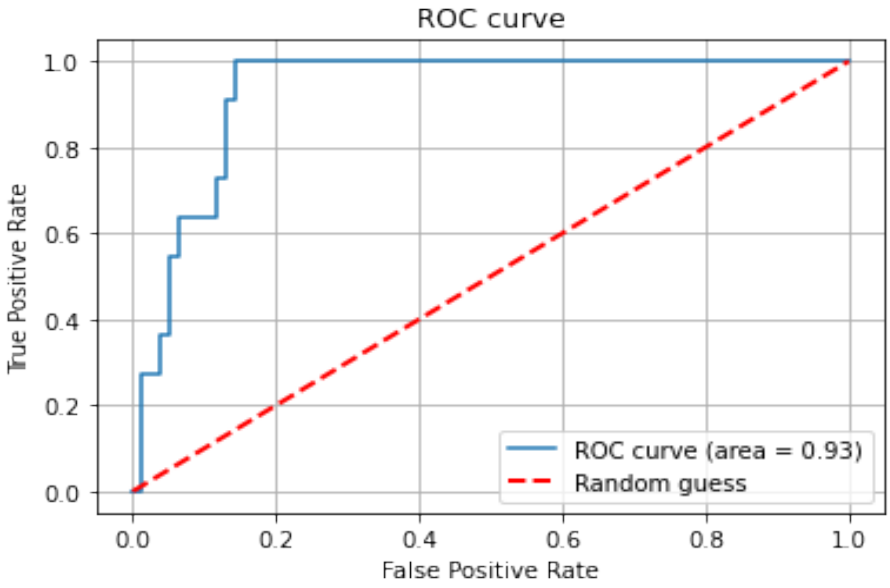

Figure 50: ROC Curve for Naive Bayes (case1)

classification_report of Naive_Bayes

	precision	recall	f1-score	support
0	0.95	0.92	0.93	76
1	0.54	0.64	0.58	11
accuracy			0.89	87
macro avg	0.74	0.78	0.76	87
weighted avg	0.89	0.89	0.89	87

Figure 51: Classification matrix for Naive Bayes (case1)

Classification report shows us that the model performs well in class 0 but fails in class 1 .Due to imbalance in the data, our model bias towards the high Cardinality class (class 0).

2. **Case 2 :**

```
Best Params:  {'var_smoothing': 0.0015199110829529332}
Best Score:  0.4328504643709971
Test set accuracy: 0.7428571428571429
Train set accuracy: 0.8218298555377207
AUC: 0.7935285053929122  precision :  0.36  recall :  0.8181818181818182 FScore :  0.5
```

Figure 52: Best Hyper parameters for Naive Bayes (case2)

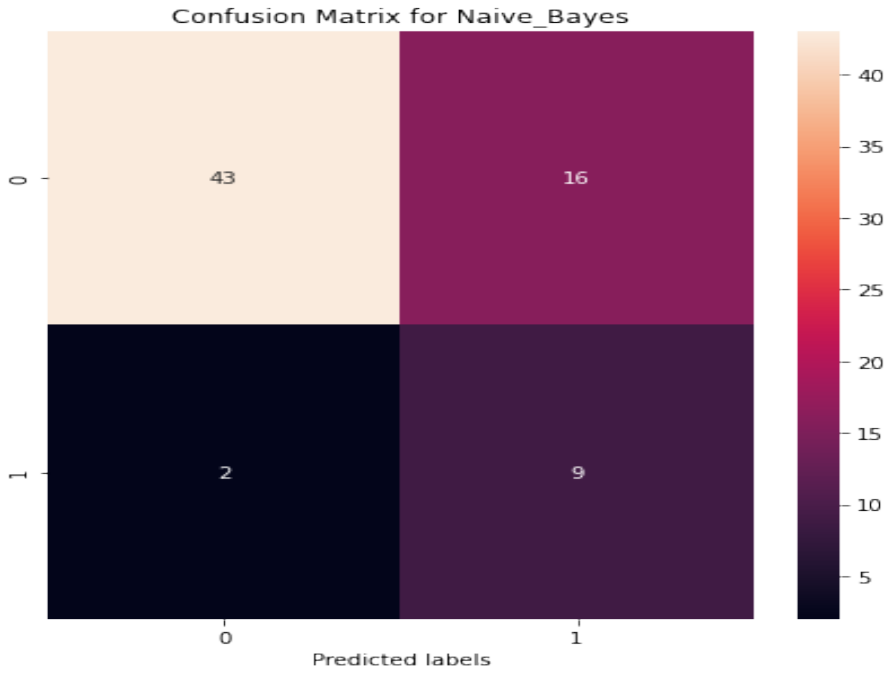

Figure 53: Confusion matrix for Naive Bayes (case2)

The model is over fitting as we said before that increasing features and imbalance have a bad impact in the model. But we can see improving in recall and f1score. .

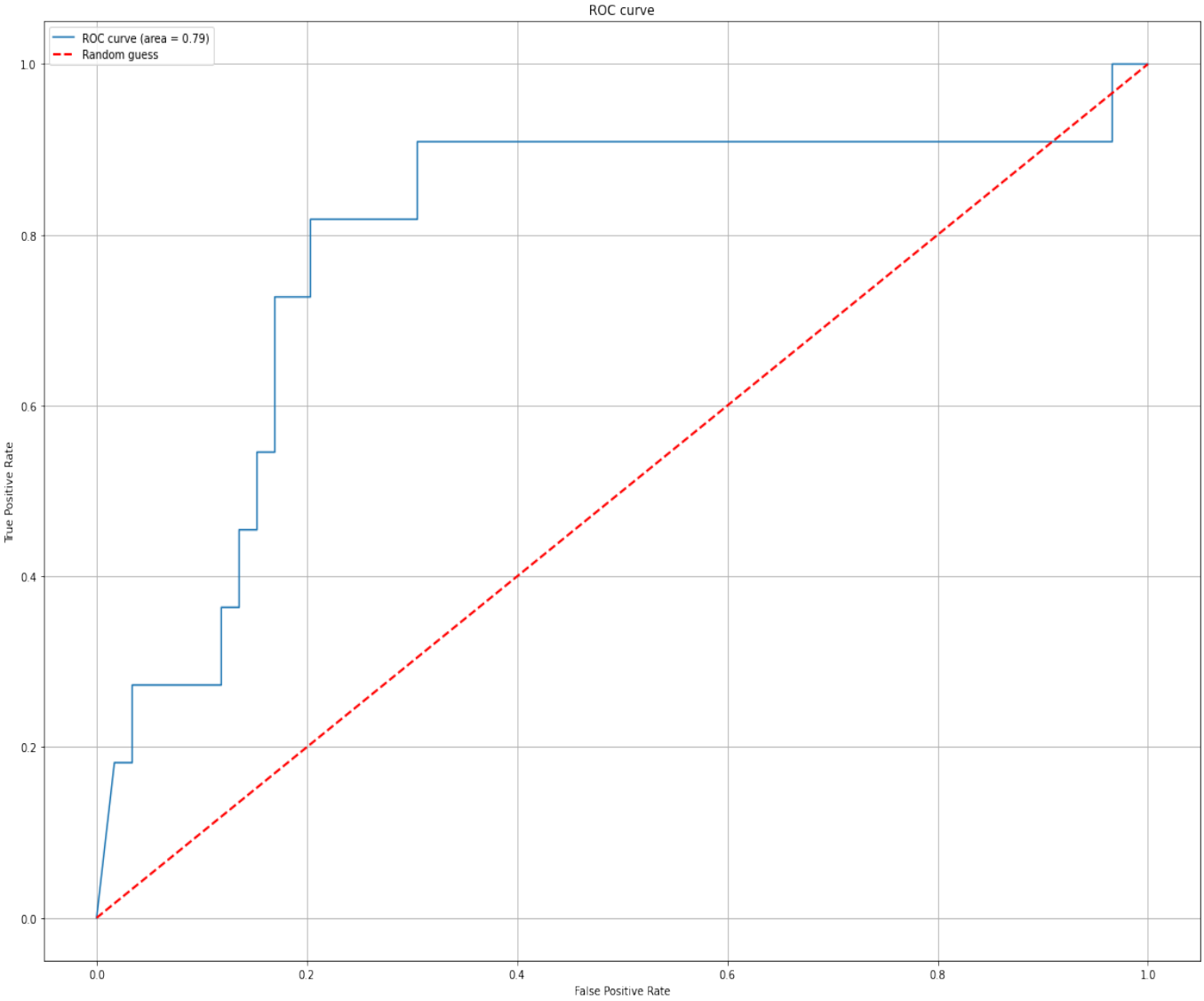

Figure 54: ROC Curve for Naive Bayes (case2)

classification_report of Naive_Bayes

	precision	recall	f1-score	support
0	0.96	0.73	0.83	59
1	0.36	0.82	0.50	11
accuracy			0.74	70
macro avg	0.66	0.77	0.66	70
weighted avg	0.86	0.74	0.78	70

Figure 55: Classification matrix for Naive Bayes (case2)

Classification report shows us that the model performs well in class 0 but fails in class 0. Model has high false positive.precision is very low.

3. **case3**

```
Best Params:  {'var_smoothing': 0.533669923120631}
Best Score:  0.8960902565432542
Test set accuracy: 0.8728813559322034
Train set accuracy: 0.8996212121212122
AUC: 0.8939959781671933  precision :  0.9074074074074074  recall :  0.8305084745762712 FScore :  0.8672566371681415
```

Figure 56: Best Hyper parameters for Naive Bayes (case3)

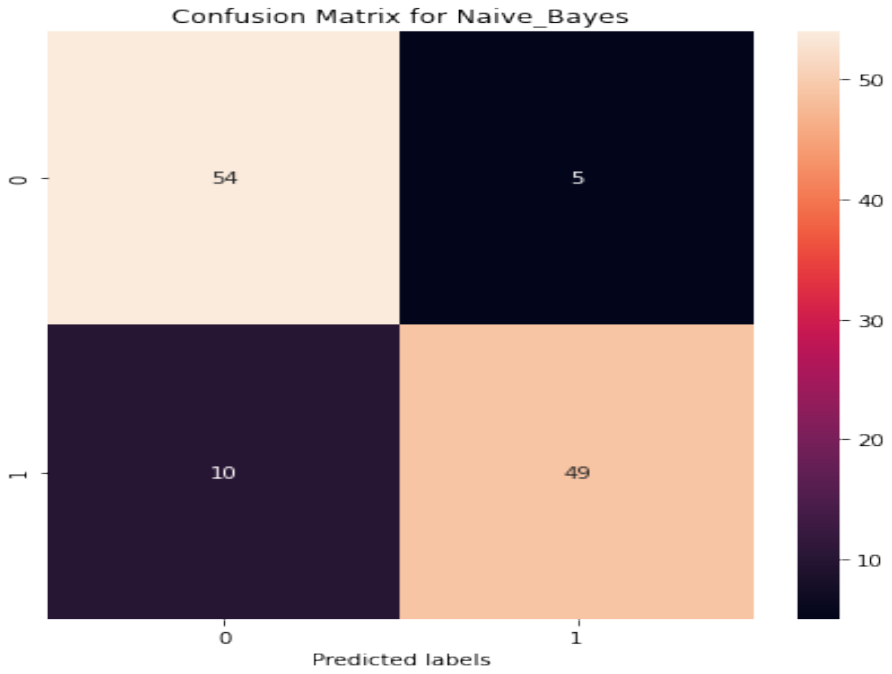

Figure 57: Confusion matrix for Naive Bayes (case3)

The over fitting is solved after solving imbalance.Also, the precision and f1score is increased.There is no significant improving in recall (from 80% to 83 %)

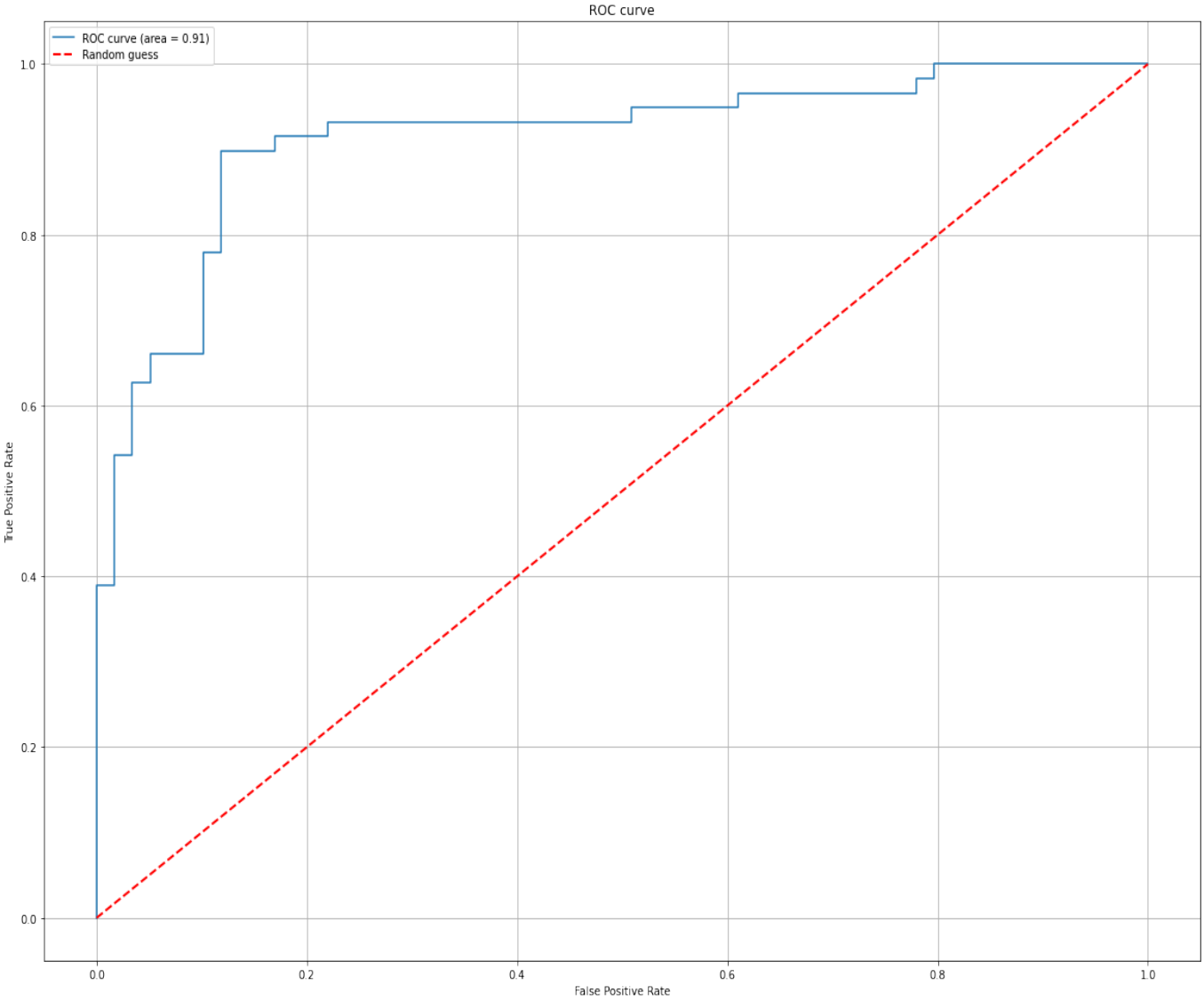

Figure 58: ROC Curve for Naive Bayes (case3)

classification_report of Naive_Bayes

	precision	recall	f1-score	support
0	0.84	0.92	0.88	59
1	0.91	0.83	0.87	59
accuracy			0.87	118
macro avg	0.88	0.87	0.87	118
weighted avg	0.88	0.87	0.87	118

Figure 59: Classification matrix for Naive Bayes (case3)

Classification report shows us that the model is relatively good not bais into certain class. we can say that:

- Knn (case 3) has 1st ranking
- Logistic Regression (case 3) has 2nd ranking
- Naive Bayes (case 3) has third ranking.

4.1.5 SVM

When tune SVM model on 3 different cases, we got the following :

1. **Case 1 :**

```
Best Params:  {'C': 1, 'class_weight': 'balanced', 'degree': 4, 'gamma': 'auto', 'kernel': 'poly'}
Best Score:  0.8497032048830718
Test set accuracy: 0.9540229885057471
Train set accuracy: 1.0
AUC: 0.9641148325358851  precision :  1.0  recall :  0.6363636363636364 FScore :  0.7777777777777778
```

Figure 60: Best Hyper parameters for SVM (case1)

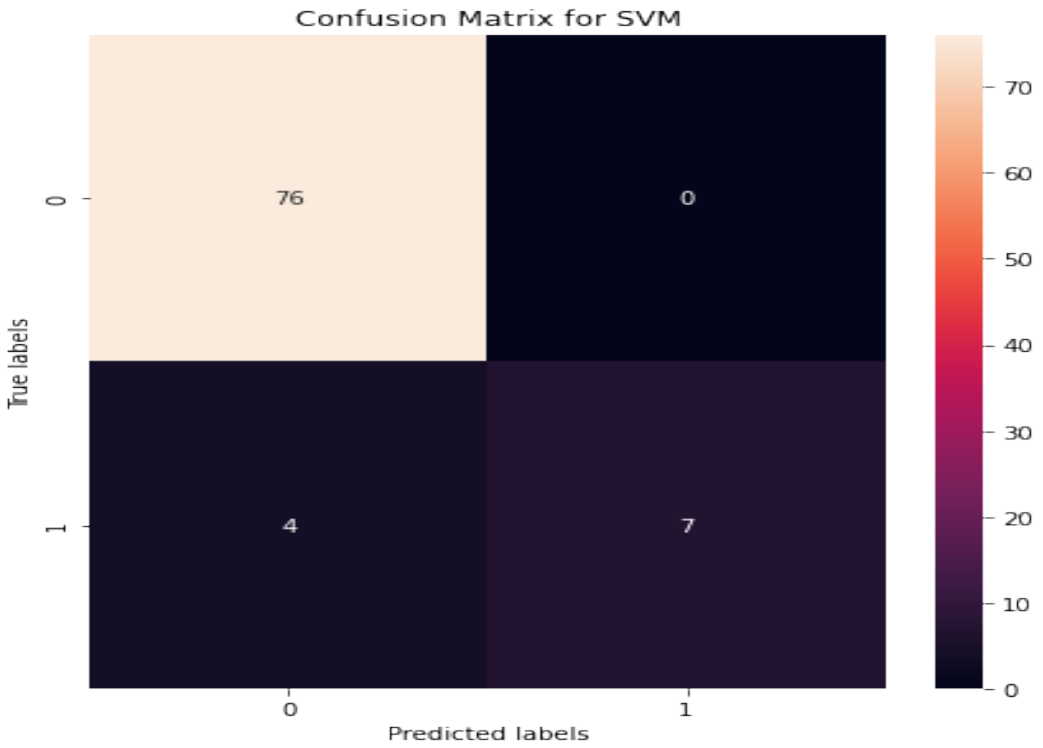

Figure 61: Confusion matrix for SVM (case1)

As you can see, SVM is solved imbalance problem in the data (class_weight = balanced). But still model need cleaned data to get good recall and f1score. There is a little bit of over fitting .The features aren't in the same range. So, Some features will take a large weight than other

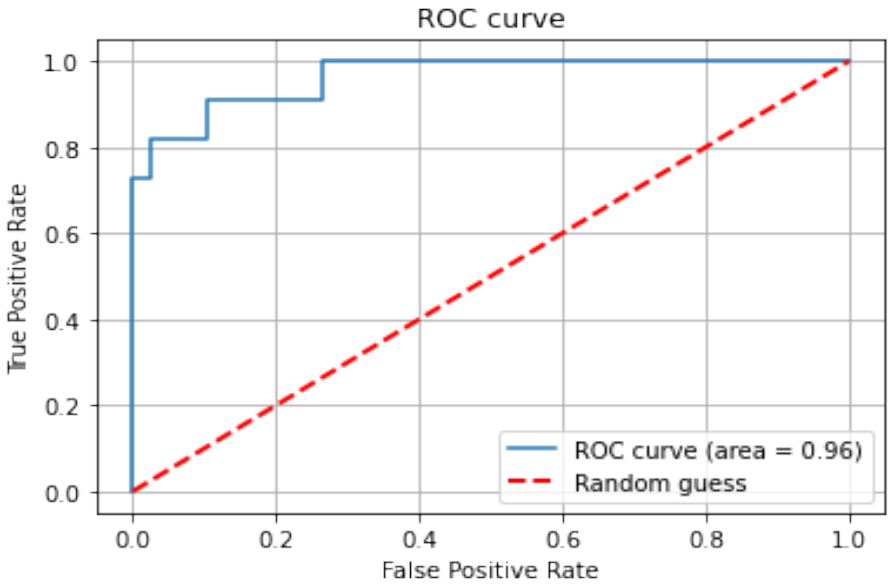

Figure 62: ROC Curve for SVM (case1)

classification_report of SVM

	precision	recall	f1-score	support
0	0.95	1.00	0.97	76
1	1.00	0.64	0.78	11
accuracy			0.95	87
macro avg	0.97	0.82	0.88	87
weighted avg	0.96	0.95	0.95	87

Figure 63: Classification matrix for SVM (case1)

Classification report shows us that the model performs well in class 0 but fails in class 1.

2. **Case 2 :**

```
Best Params:  {'C': 9, 'class_weight': 'balanced', 'degree': 2, 'gamma': 'scale', 'kernel': 'rbf'}
Best Score:  0.8798669302953268
Test set accuracy: 1.0
Train set accuracy: 0.9919743178170144
AUC: 1.0  precision :  1.0  recall :  1.0 FScore :  1.0
```

Figure 64: Best Hyper parameters for SVM (case2)

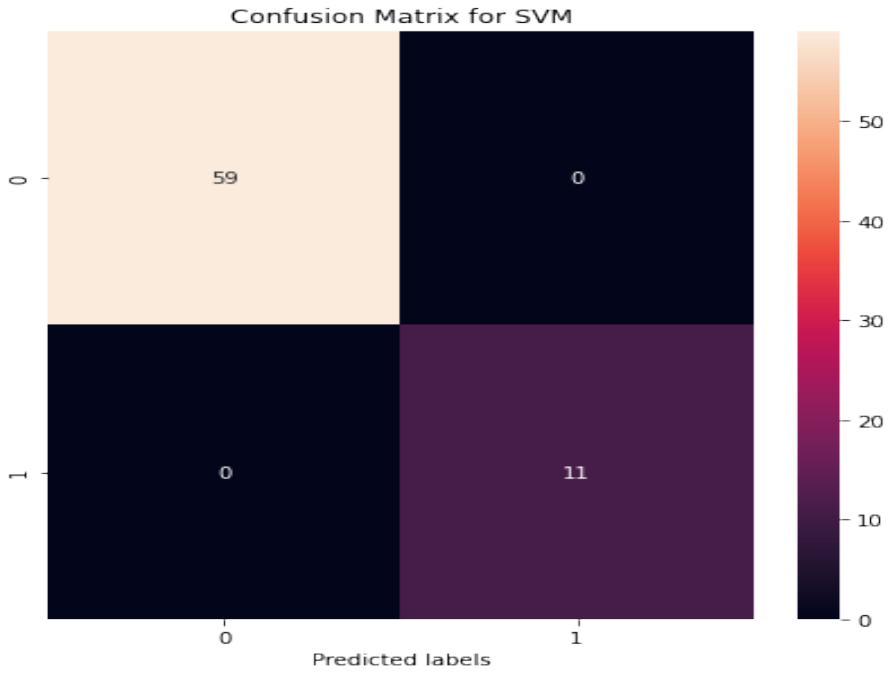

Figure 65: Confusion matrix for SVM (case2)

The model is awesome.The model performs well. With some prepossessing we got this pretty model.As you can see, SVM is solved imbalance problem in the data (class_weight = balanced).There is no over fitting. It is soft margin SVM. As we studied in our class, it is not preferred as a single outlier can determine the boundary, which makes the classifier overly sensitive to noise in the data. we need to choose smaller C to maximize our margin and minimize violation.

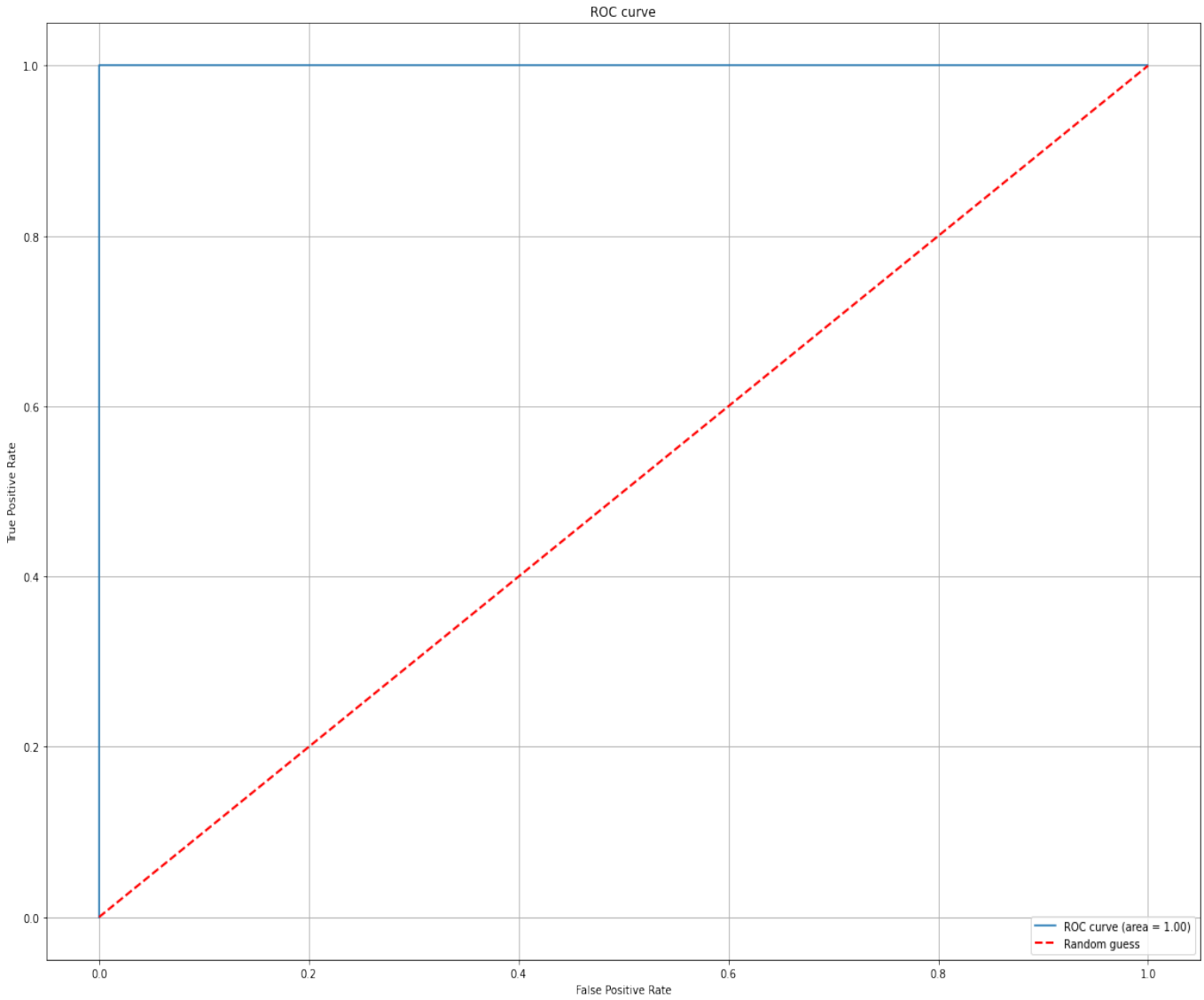

Figure 66: ROC Curve for SVM (case2)

classification_report of SVM

	precision	recall	f1-score	support
0	1.00	1.00	1.00	59
1	1.00	1.00	1.00	11
accuracy			1.00	70
macro avg	1.00	1.00	1.00	70
weighted avg	1.00	1.00	1.00	70

Figure 67: Classification matrix for SVM (case2)

Classification report shows us that the model performs well in both classes.

3. **case3**

```
Best Params:  {'C': 5, 'class_weight': 'balanced', 'degree': 2, 'gamma': 'scale', 'kernel': 'rbf'}
Best Score:  0.9710624458942464
Test set accuracy: 0.9661016949152542
Train set accuracy: 0.9876893939393939
AUC: 0.9971272622809537  precision :  0.9508196721311475  recall :  0.9830508474576272 FScore :  0.9666666666666667
```

Figure 68: Best Hyper parameters for SVM (case3)

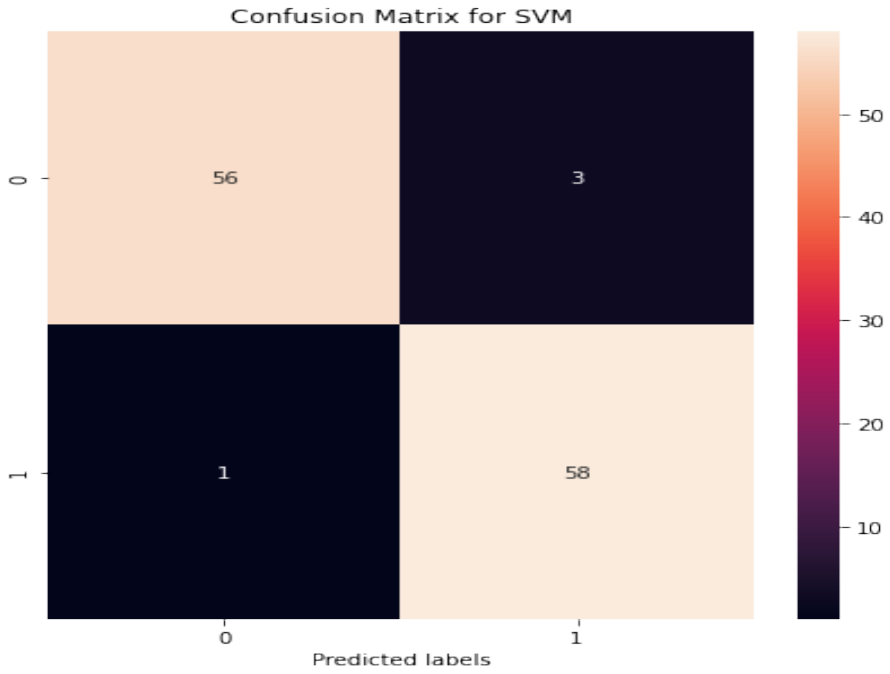

Figure 69: Confusion matrix for SVM (case3)

Although, Accuracy, precision ,recall and f1score decrease a little bit but still acceptable for us. we prefer this case than case 2. It is hard margin and this what we need.Notice that C deceases from 9 to 5 and all other hyper parameters are the same.

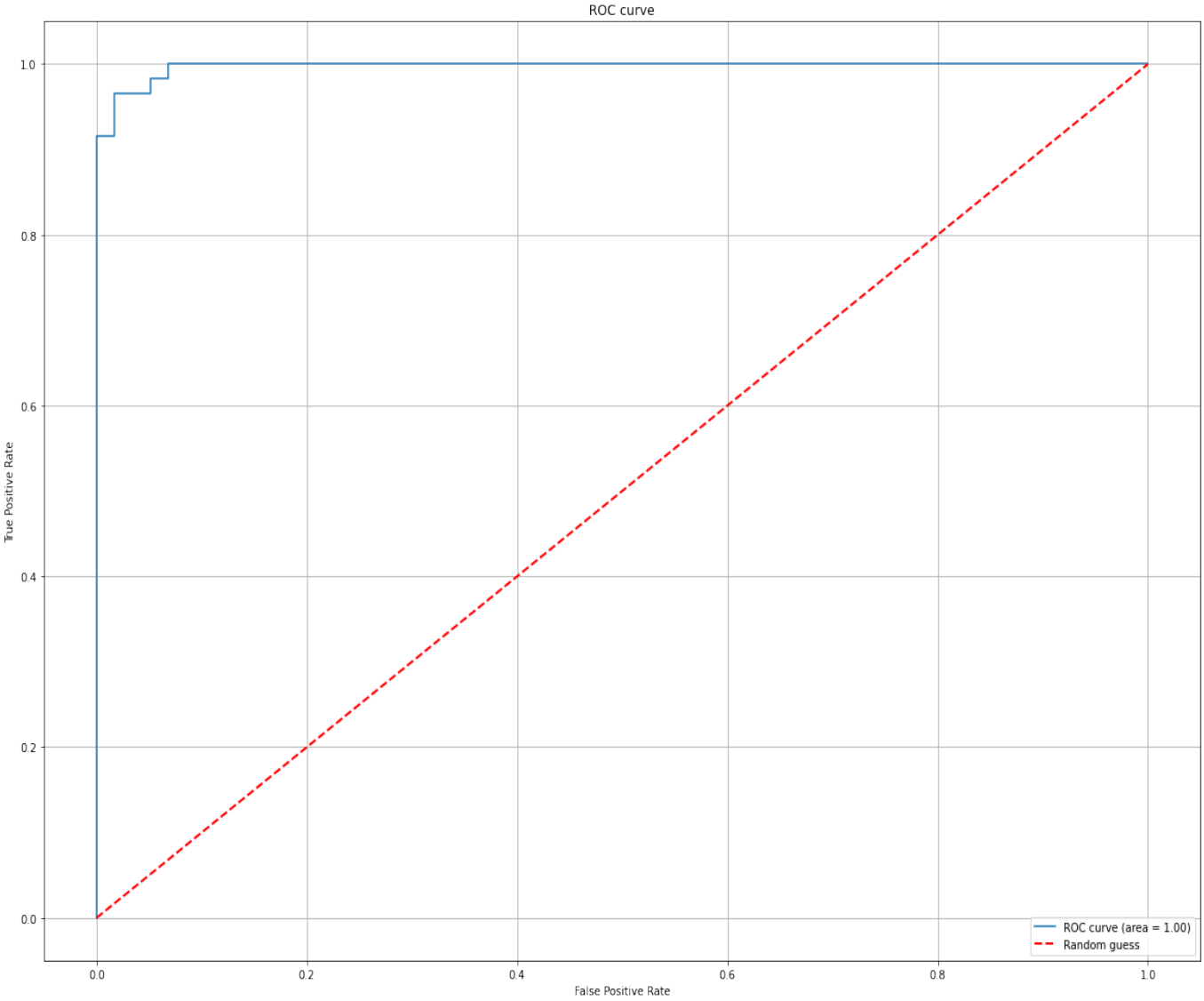

Figure 70: ROC Curve for SVM (case3)

classification_report of SVM

	precision	recall	f1-score	support
0	0.98	0.95	0.97	59
1	0.95	0.98	0.97	59
accuracy			0.97	118
macro avg	0.97	0.97	0.97	118
weighted avg	0.97	0.97	0.97	118

Figure 71: Classification matrix for SVM (case3)

Classification report shows us that the model is perfect classifier.

5 Comparison between Models

Let's visualize our model to summarize our insights

1. **Case 1** :

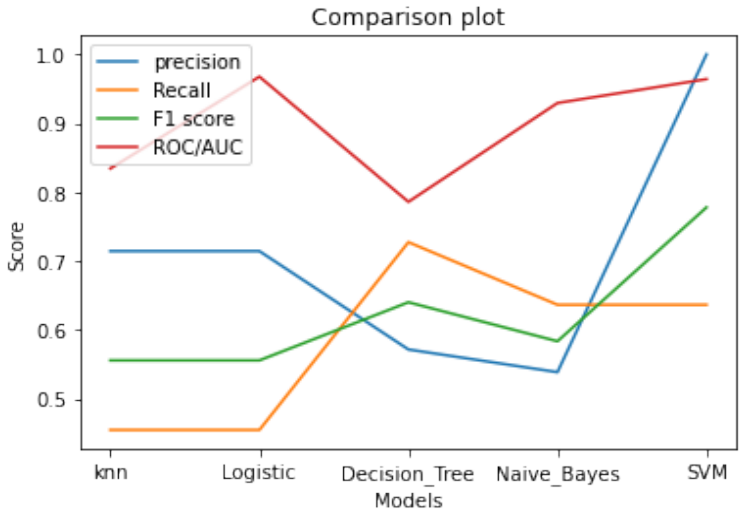

Figure 72: Evaluation matrices for each model in case (1)

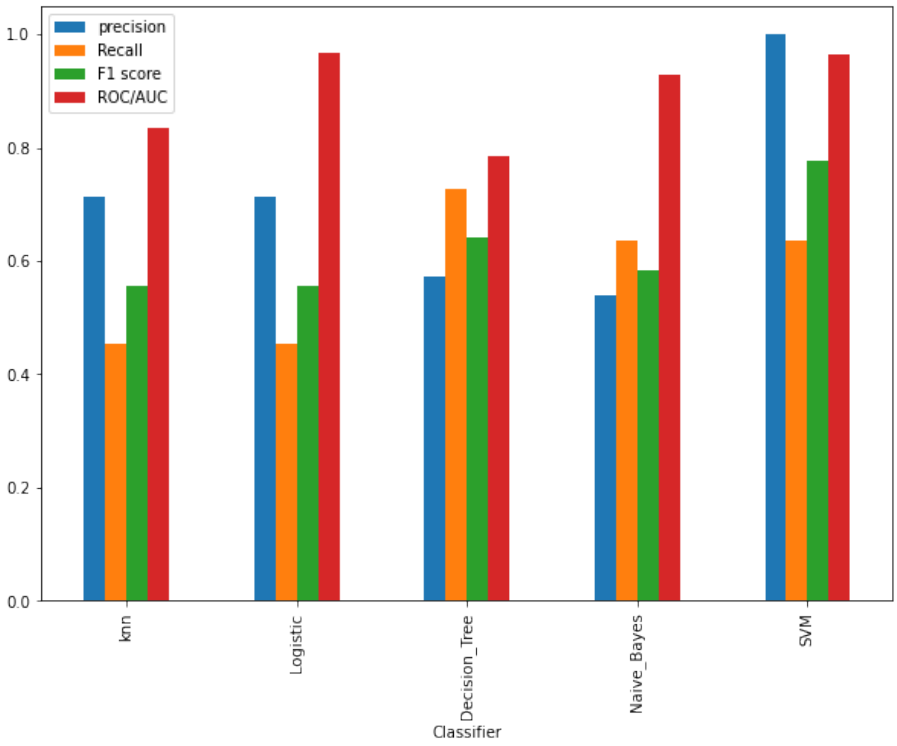

Figure 73: bar plot Evaluation matrices for each model in case (1)

our Visualization emphasize our insights that no model in case (1) is suitable for our problem.Recall and f1score are very low. This reflects that prepossessing in our data is crucial.

2. **Case 2** :

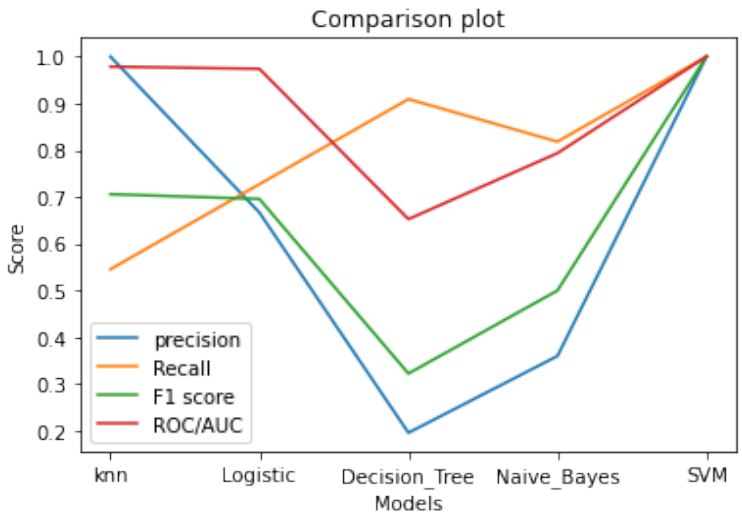

Figure 74: Evaluation matrices for each model in case (2)

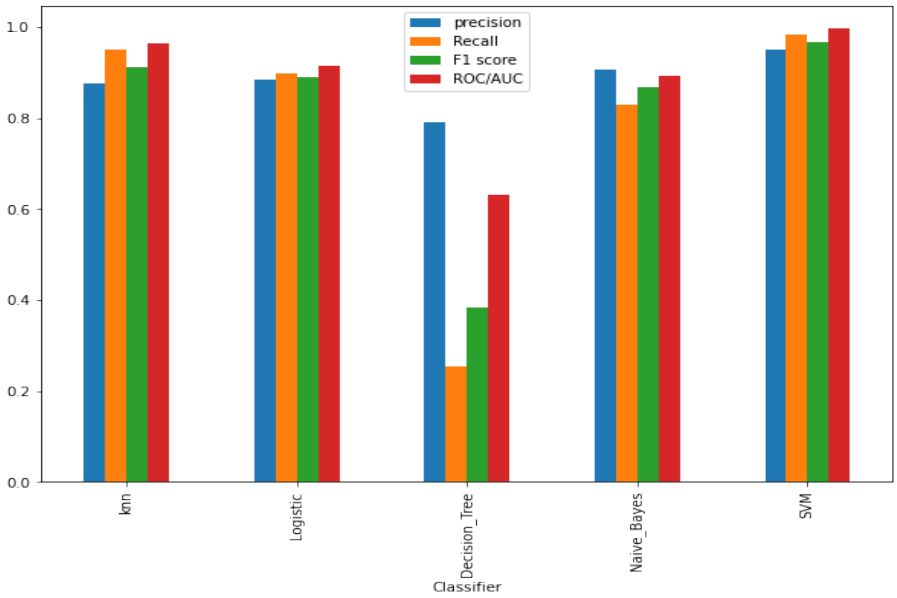

Figure 75: Evaluation matrices for each model in case (2)

In this case, the only one that has high recall is SVM but also we reject this model and we justify why in the previous section.

3. **Case 3** :
   As you can see it the best case, we got models with high recall and f1score.

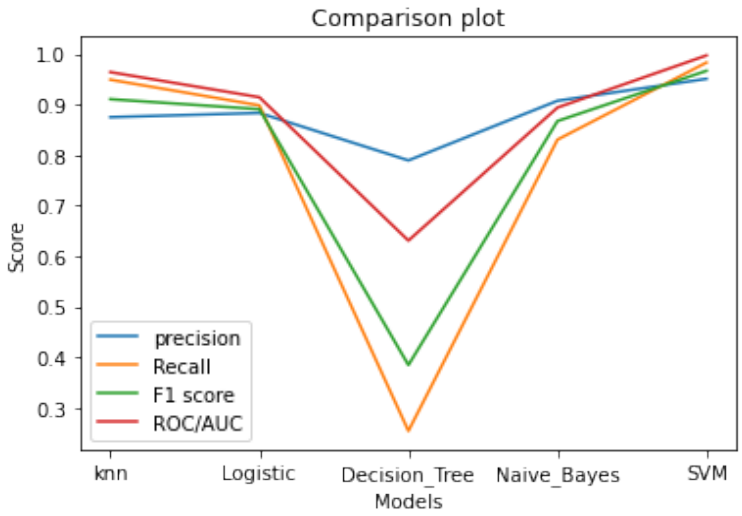

Figure 76: Evaluation matrices for each model in case (3)

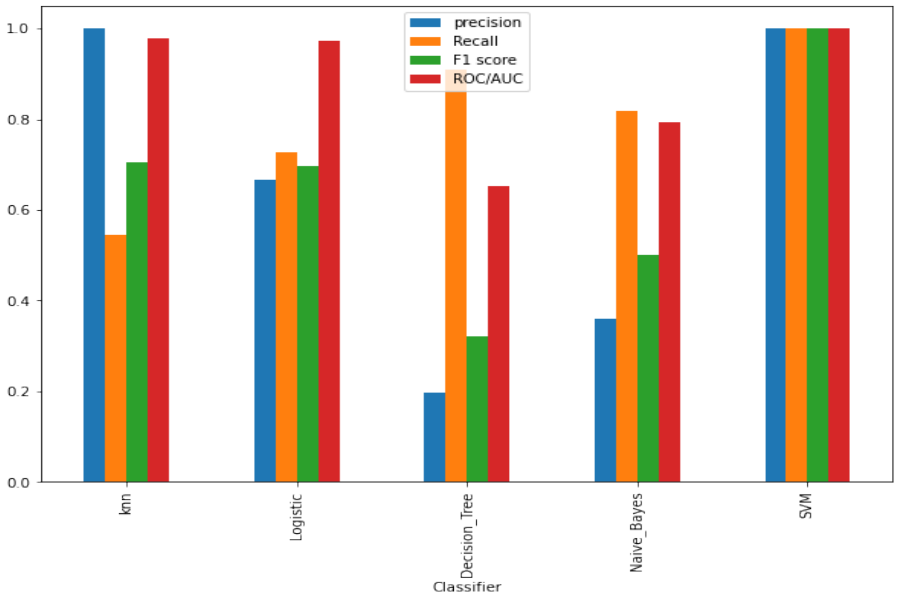

Figure 77: Evaluation matrices for each model in case (3)

On the top of these models is SVM.

- 1st rank is SVM
- 2nd rank is Logistic regression as it has lower over fitting than KNN
- 3rd rank is KNN

- 4th rank is Naive Bayes
- 5th rank is Decision Tree