Project Topic
Predicting Round Winner In CS:GO

Name - Sanchay Gawande

1○ Description of the problem and the data
2○ Parameter tuning with charts
3○ Generation and tuning of alternative models
4○ Learning curve analysis
5○ Performance and error analysis

1○ Description of the problem and the data

Goal : Predicting which one of the two teams will win the round in CS:GO Game.

Here in this project I am going to build 2 models (Logistic regression and Neural Network) for the data set to predict the winner of the round using the features and see how well the models can do for predicting the winner of the game.

Data : The dataset consists of round snapshots from about 700 demos from high level tournament play in 2019 and 2020. Warmup rounds and restarts have been filtered, and for the remaining live rounds a round snapshot has been recorded every 20 seconds until the round is decided. Following its initial publication, It has been pre-processed and flattened to improve readability and make it easier for algorithms to process. The total number of snapshots is 122410. Snapshots are i.i.d and should be treated as individual data points, not as part of a match.

The data file is in the zip file I have submitted named as -
csgo_round_snapshots.csv

Data Analysis :

This dataset contains 122410 rows and 97 columns.

	time_left	ct_score	t_score	map	bomb_planted	ct_health	t_health	ct_armor	t_armor	ct_money	...	t_grenade_flashbang	ct_grenade_smokegrenade
0	175.00	0	0	de_dust2	False	500	500	0	0	4000	...	0	0
1	156.03	0	0	de_dust2	False	500	500	400	300	600	...	0	0
2	96.03	0	0	de_dust2	False	391	400	294	200	750	...	0	0
3	76.03	0	0	de_dust2	False	391	400	294	200	750	...	0	0
4	174.97	1	0	de_dust2	False	500	500	192	0	18350	...	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
122405	15.41	11	14	de_train	True	200	242	195	359	100	...	2	1
122406	174.93	11	15	de_train	False	500	500	95	175	11500	...	2	1
122407	114.93	11	15	de_train	False	500	500	495	475	1200	...	4	3
122408	94.93	11	15	de_train	False	500	500	495	475	1200	...	5	0
122409	74.93	11	15	de_train	False	375	479	395	466	1100	...	3	0

122410 rows × 97 columns

Every column in the dataset can be considered as a feature, but there are 97 columns that is why I printed correlation matrix and used feature importance to see which features are contributing for the output and ended up with a total of 20 useful features.

Feature Importance Graph:

Feature Correlation Matrix :

	map	bomb_planted	ct_health	t_health	ct_armor	t_armor	ct_helmets	t_helmets	ct_defuse_kits	ct_players_alive	...	ct_grenade_flashbang	t_grenade_flashbang	t_grenade_
0	de_dust2	False	500	500	0	0	0	0	0	5	...	0	0	
1	de_dust2	False	500	500	400	300	0	0	1	5	...	0	0	
2	de_dust2	False	391	400	294	200	0	0	1	4	...	0	0	
3	de_dust2	False	391	400	294	200	0	0	1	4	...	0	0	
4	de_dust2	False	500	500	192	0	0	0	1	5	...	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
122405	de_train	True	200	242	195	359	2	4	1	2	...	1	2	
122406	de_train	False	500	500	95	175	1	2	1	5	...	1	2	
122407	de_train	False	500	500	495	475	3	5	1	5	...	4	4	
122408	de_train	False	500	500	495	475	3	5	1	5	...	1	5	
122409	de_train	False	375	479	395	466	2	5	1	4	...	0	3	

122410 rows × 21 columns

Then I went on to see how many samples we have for CT and T.
CT - 60004
T - 62406
Which is a good sample for both the classes as our model can learn properly based on the number of samples we have for both classes.

After reducing the features the dataset contained **Three non numerical data** shown below.

	map	bomb_planted	round_winner
0	de_dust2	False	CT
1	de_dust2	False	CT
2	de_dust2	False	CT
3	de_dust2	False	CT
4	de_dust2	False	CT
...	...	...	...
122405	de_train	True	T
122406	de_train	False	T
122407	de_train	False	T
122408	de_train	False	T
122409	de_train	False	T

122410 rows × 3 columns

I converted their non-numeric data into Int format using label encoder and replaced the CT and T for 0 and 1 and the same for False and True.
(0: 'de_cache' 1: 'de_dust2', 2: 'de_inferno', 3: 'de_mirage', 4: 'de_nuke',
5: 'de_overpass',6: 'de_train', 7: 'de_vertigo')

After that my dataset was looking like this:

	map	bomb_planted	ct_health	t_health	ct_armor	t_armor	ct_helmets	t_helmets	ct_defuse_kits	ct_players_alive	...	ct_grenade_flashbang	t_grenade_flashbang	t_grenade_smol
0	1	0	500	500	0	0	0	0	0	5	...	0	0	
1	1	0	500	500	400	300	0	0	1	5	...	0	0	
2	1	0	391	400	294	200	0	0	1	4	...	0	0	
3	1	0	391	400	294	200	0	0	1	4	...	0	0	
4	1	0	500	500	192	0	0	0	1	5	...	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
122405	6	1	200	242	195	359	2	4	1	2	...	1	2	
122406	6	0	500	500	95	175	1	2	1	5	...	1	2	
122407	6	0	500	500	495	475	3	5	1	5	...	4	4	
122408	6	0	500	500	495	475	3	5	1	5	...	1	5	
122409	6	0	375	479	395	466	2	5	1	4	...	0	3	

122410 rows × 21 columns

I did not do feature scaling as the data it was not need after removing the extra features in the dataset

Applying Machine learning algorithms

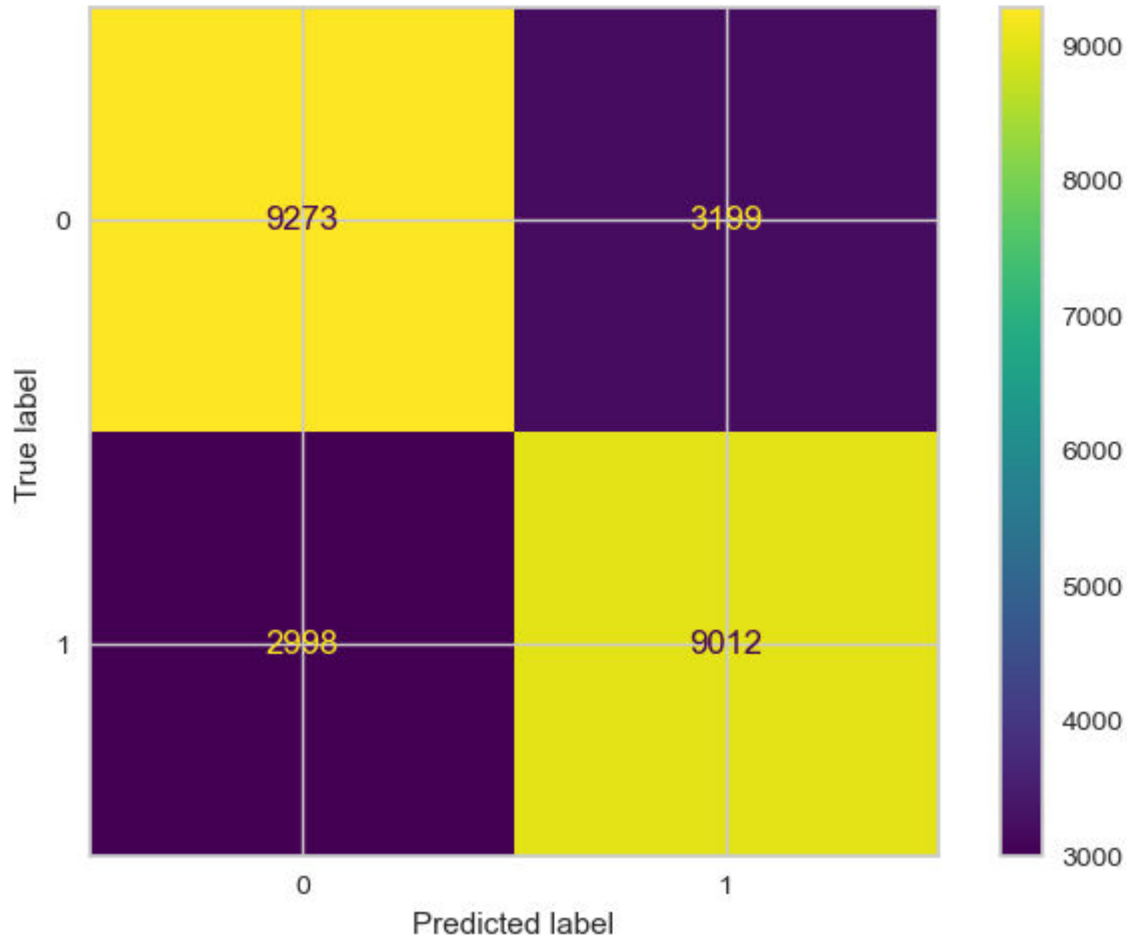
1. **Logistic regeneration**
2. **Neural Networks**

1. Logistic regeneration Model

First I splitted my data into Train Test Validation Split as nearly :

Train	60%	train shape:(78342, 20)
Validation	20%	val shape: (19586, 20)
Test	20%	test shape: (24482, 20)

First, I trained my data with default parameters and this i got theses results :



This is the confusion matrix I got. You can see in total 3199 numbers of samples of label 0 are mispredicted as 1 and 2998 numbers of samples of label 1 are mispredicted as 0. This is the result I got, without the hyperparameter tuning with the default parameters.

2○ Parameter tuning with charts

Hyperparameter Tuning

I went on to do the hyperparameter tuning, where I changed the parameters like solvers, multiclass, changing the maximum iterations and applied regularization function (lambda) as l1 and l2 used in sklearn. I have experimented with these parameters by changing them, and seeing if we can get a better score, or not after the experimentation, I got the good result, and went on with the best results I got for the Logistic Regression model after hyper tuning the perimeters. The chart of different parameters is given below.

hypertuned_model1 = LR(solver="liblinear", multi_class="ovr", max_iter=700, penalty="l1", random_state=1)
hypertuned_model1.fit(train, y_train)
hypertuned_model_output1 = hypertuned_model1.predict(val)

hypertuned_model2 = LR(solver="liblinear", multi_class="ovr", max_iter=700, penalty="l2", random_state=1)
hypertuned_model2.fit(train, y_train)
hypertuned_model_output2 = hypertuned_model2.predict(val)

hypertuned_model3 = LR(solver="liblinear", multi_class="auto", max_iter=700, penalty="l1", random_state=1)
hypertuned_model3.fit(train, y_train)
hypertuned_model_output3 = hypertuned_model3.predict(val)

hypertuned_model4 = LR(solver="lbfgs", multi_class="ovr", max_iter=1500, penalty="l2", random_state=1)
hypertuned_model4.fit(train, y_train)
hypertuned_model_output4 = hypertuned_model4.predict(val)

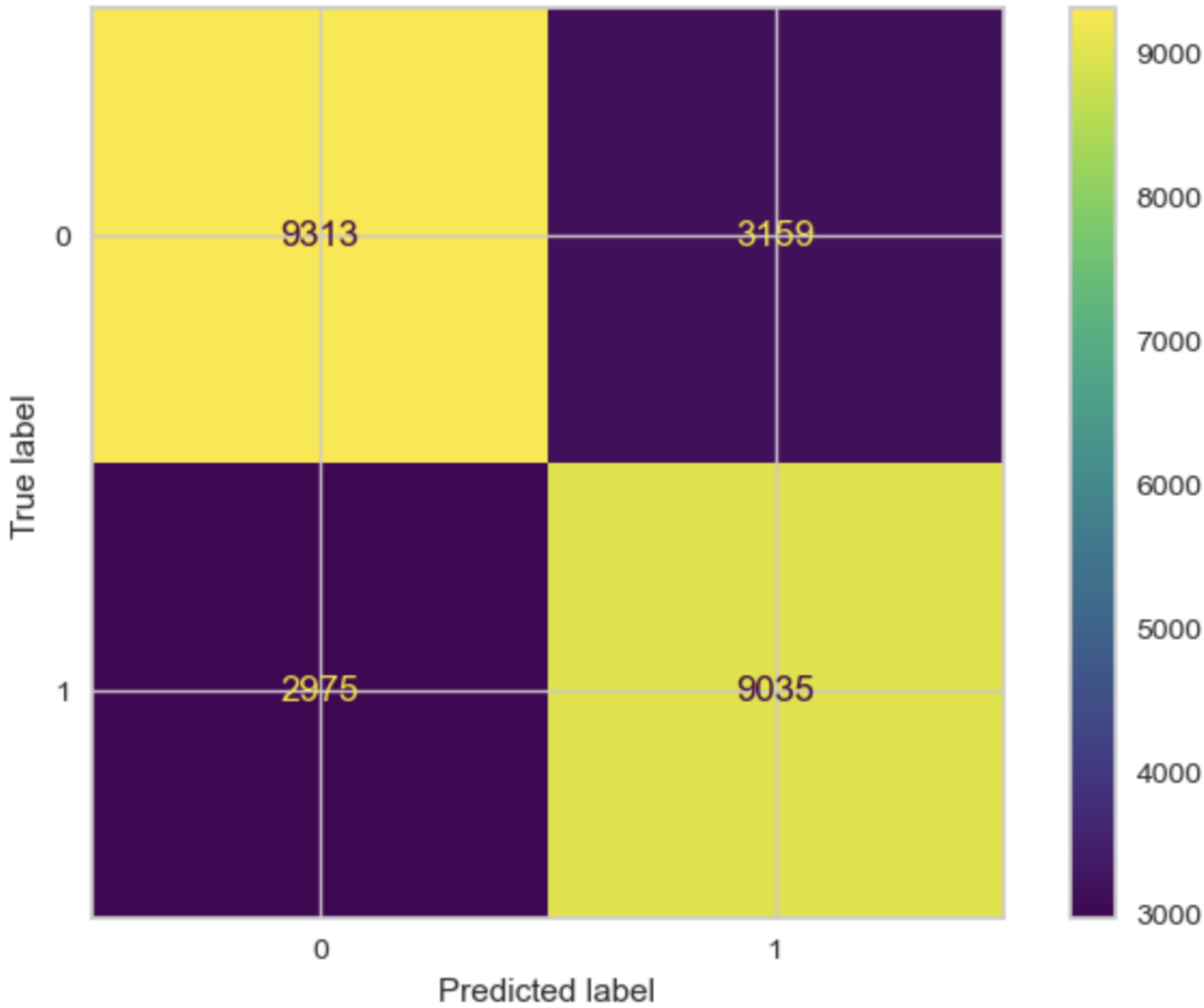
hypertuned_model5 = LR(solver="liblinear", multi_class="ovr", max_iter=700, penalty="l1", random_state=1)
hypertuned_model5.fit(train, y_train)

```
hypertuned_model_output5 = hypertuned_model5.predict(va
```

```
hypertuned_optimal_model = LR(solver="liblinear", multi_class="ovr",
max_iter=1500, penalty="l1", random_state=1)
hypertuned_optimal_model.fit(train, y_train)
hypertuned_optimal_model_output = hypertuned_optimal_model.predict(test)
```

```
auc1: 0.7473036012928227
auc2: 0.7474578147370807
auc3: 0.7473036012928227
auc4: 0.7473674918059505
auc5: 0.7473036012928227
optimal_auc_lr: 0.7440719435035551
```

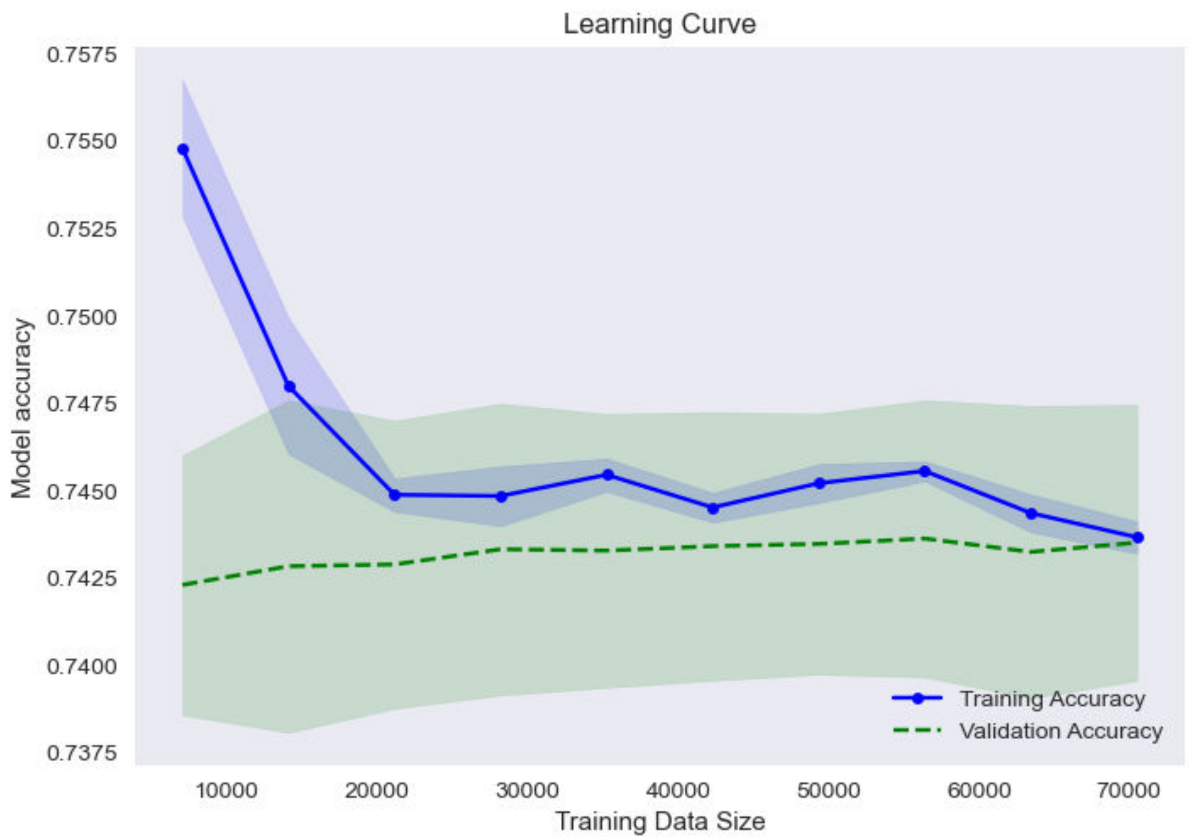
Confusion Matrix After Hyperparameter tuning :



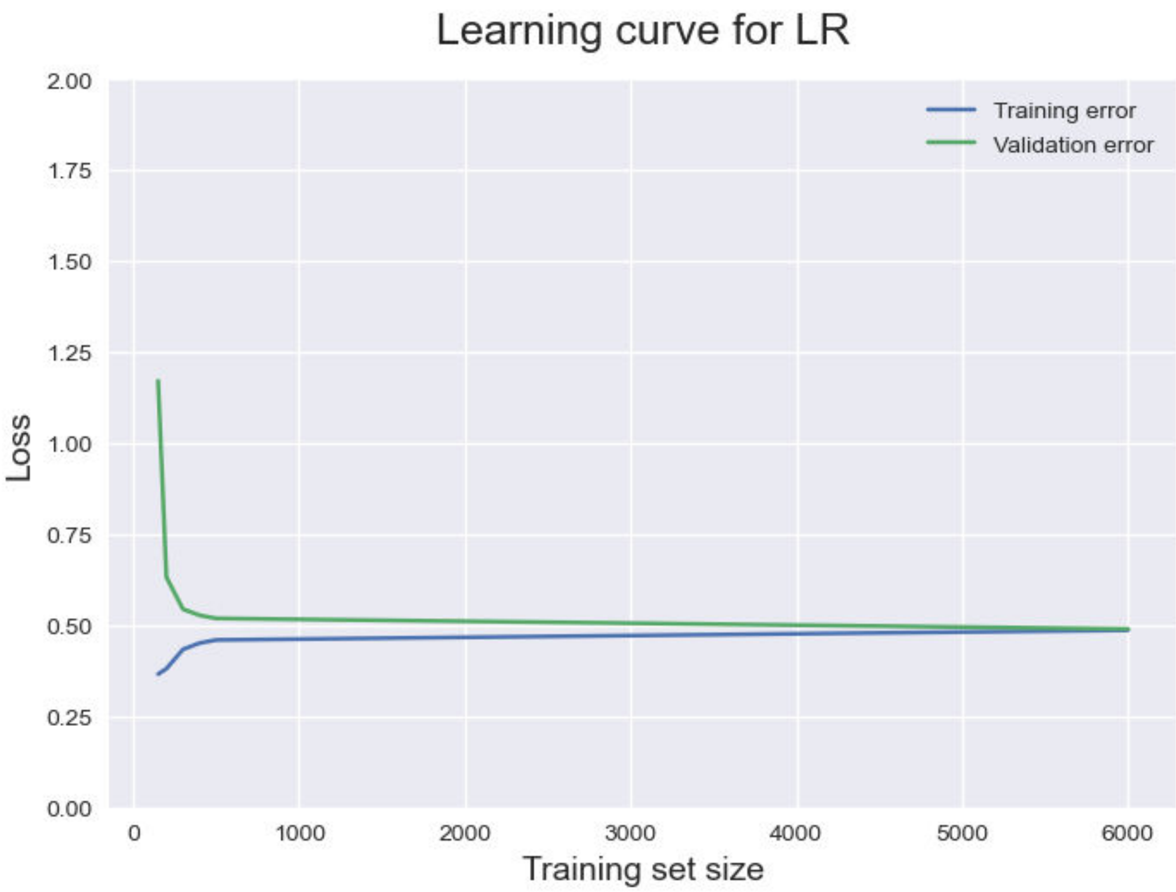
This is the confusion matrix I got after hyperparameter tuning. You can see in total 3159 numbers of samples of label 0 are mispredicted as 1 and 2975 numbers of samples of label 1 are mispredicted as 0. This is the result I got, After the hyperparameter tuning.

4○ Learning curve analysis

Learning curves :



Here in this Learning curve of the model accuracy versus training data size. We can see that as we increase the number of training data the training accuracy goes down and in case of validation accuracy, the accuracy increases slightly with increasing number of Training data set.



Learning curve for the loss here we can say that validation error decreases as we increase the training, size and training, error and validation, error is converging as we increase the number of training set size so it should be considered as a good fit.

ROC Curve For LR Model :

Here we can see that the AUC Area under the curve is 0.84
Here there are two different lines for micro average ROC curve and macro average ROC curve.
Macro average is which gives equal weights to each category while micro averaging gives equal weight to each sample. So if we have nearly the same number of samples for each class, both macro and micro will provide the same score. So in the above ROC Curve, the AUC for macro and micro average is the same, which indicates that we have nearly the same number of samples for each class.

Classification Report :

	precision	recall	f1-score	support
0	0.75	0.74	0.75	12473
1	0.74	0.75	0.74	12009
accuracy			0.74	24482
macro avg	0.74	0.74	0.74	24482
weighted avg	0.74	0.74	0.74	24482

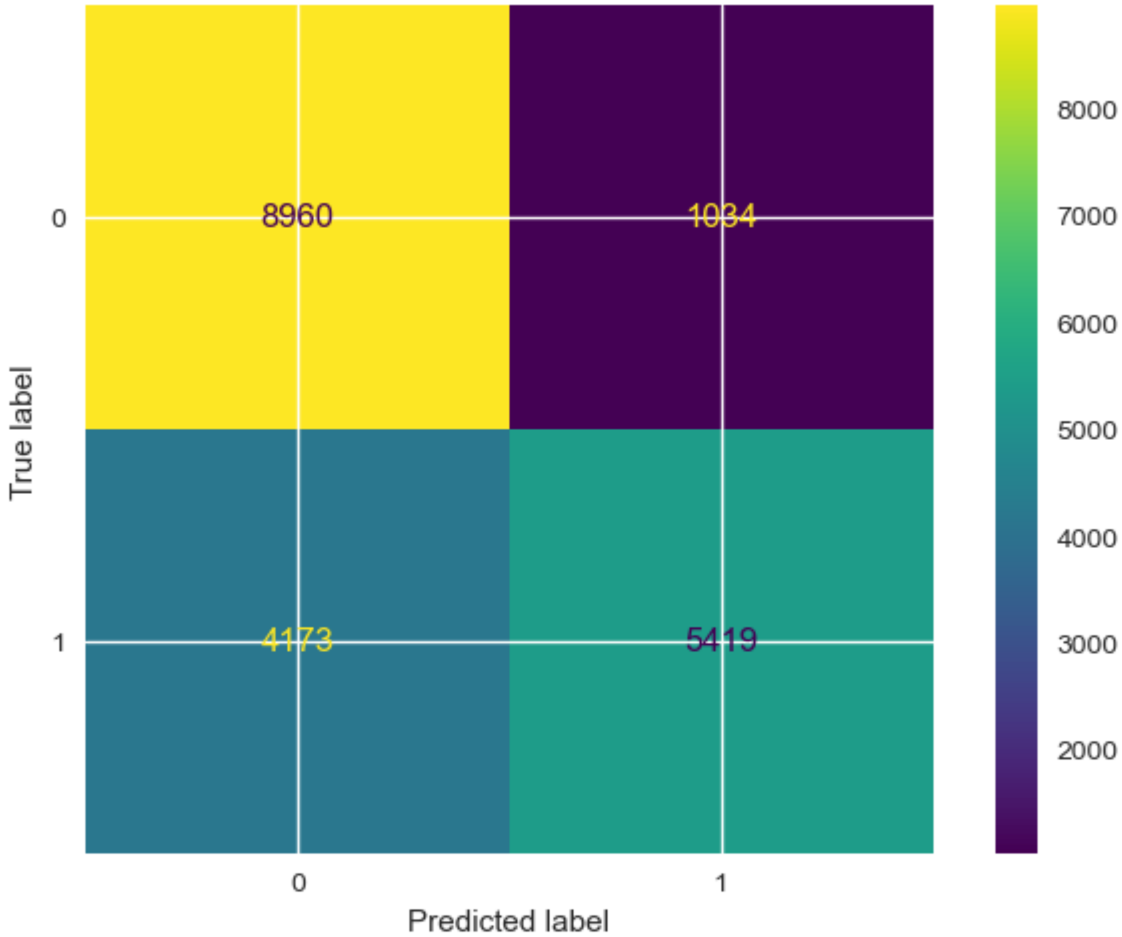
Finally, the accuracy of our logistic regression model comes to 74%. Where are precision for class zero is 0.75 and for class 1 its 0.74 while recall for class 0 is 0.74 and for class 1 its 0.75

3○ Generation and tuning of alternative models

2. Neural Networks Model

Here I used 2 layered Neural network Model and split my data into the same train test validation split 60-20-20 for train, test, validation, respectively.

First, I trained my data with default parameters and this i got theses results :



As I noticed here running the model with default parameters Neural Network model performed worse than logistic regression model as you can see. Here there are 1034 label zero samples that are miss predicted as 1. On the other hand, in case of label 1, the model miss predicted 4173 labels as 0, where it should have been 1.

2○ Parameter tuning with charts

Hyperparameter Tuning :

So I performed hyperparameter tuning where I tweaked the parameter for the neural network model like solvers, hidden layers, maximum iterations, penalty (alpha) which is regularization function. I have experimented with these parameters by changing them, and seeing if we can get a better score, or not after the experimentation, I got the good result, and went on with the best results I got for the Neural network Model after hyper tuning the perimeters. The chart of different parameters is given below.

```
pretuned_clf = MLPClassifier(solver="lbfgs", alpha=0.001, hidden_layer_sizes=(10, 85),
max_iter=350, random_state=1)
pretuned_clf.fit(train, y_train)
pretuned_clf_output = pretuned_clf.predict(val)

pretuned_clf = MLPClassifier(solver="lbfgs", alpha=0.0008, hidden_layer_sizes=(5, 47),
max_iter=350, random_state=1)
pretuned_clf.fit(train, y_train)
pretuned_clf_output = pretuned_clf.predict(val)

pretuned_clf = MLPClassifier(solver="lbfgs", alpha=0.002, hidden_layer_sizes=(10, 25),
max_iter=350, random_state=1)
pretuned_clf.fit(train, y_train)
pretuned_clf_output = pretuned_clf.predict(val)

pretuned_clf = MLPClassifier(solver="adam", alpha=0.2, hidden_layer_sizes=(10, 80),
max_iter=350, random_state=1)
pretuned_clf.fit(train, y_train)
pretuned_clf_output = pretuned_clf.predict(val)

pretuned_clf = MLPClassifier(solver="lbfgs", alpha=0.8, hidden_layer_sizes=(5, 40),
max_iter=350, random_state=1)
pretuned_clf.fit(train, y_train)
pretuned_clf_output = pretuned_clf.predict(val)

pretuned_clf = MLPClassifier(solver="lbfgs", alpha=2, hidden_layer_sizes=(10, 100),
max_iter=350, random_state=1)
pretuned_clf.fit(train, y_train)
pretuned_clf_output = pretuned_clf.predict(val)

pretuned_clf = MLPClassifier(solver="adam", alpha=0.2, hidden_layer_sizes=(10, 80),
max_iter=350, random_state=1)
pretuned_clf.fit(train, y_train)
```

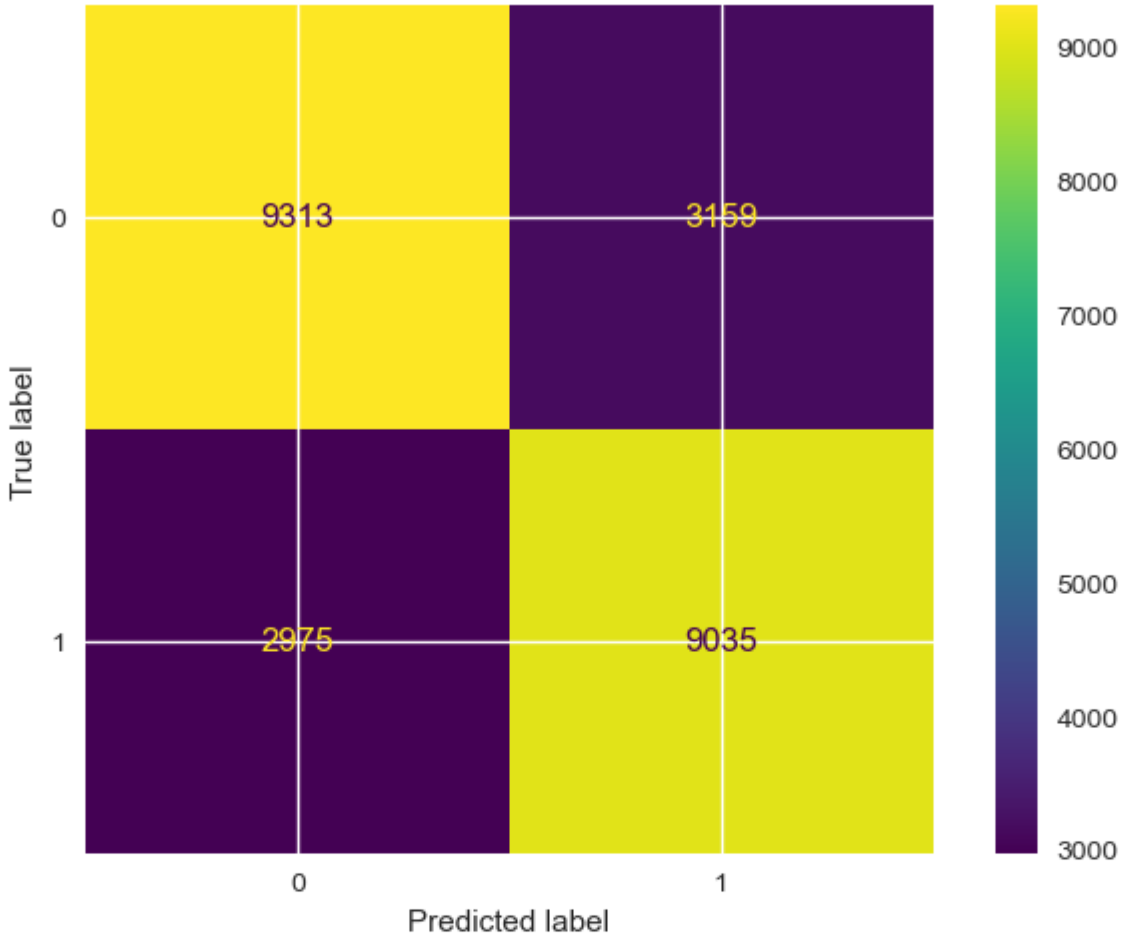
```
pretuned_clf_output = pretuned_clf.predict(val)

pretuned_clf = MLPClassifier(solver="adam", alpha=0.8, hidden_layer_sizes=(5, 40),
max_iter=350, random_state=1)
pretuned_clf.fit(train, y_train)
pretuned_clf_output = pretuned_clf.predict(val)
```

[0.7398513388238771, 0.737195391938726, 0.7386328189575055, 0.7494100167473463, 0.7453865938280743, 0.744859186769421, 0.7494100167473463, 0.7429962124243463]

```
hypertuned_model = MLPClassifier(solver="adam", alpha=0.02, hidden_layer_sizes=(10,
70), max_iter=350, random_state=1)
hypertuned_model.fit(train, y_train)
hypertuned_model_output = hypertuned_model.predict(val)
```

[0.7501753155680224]

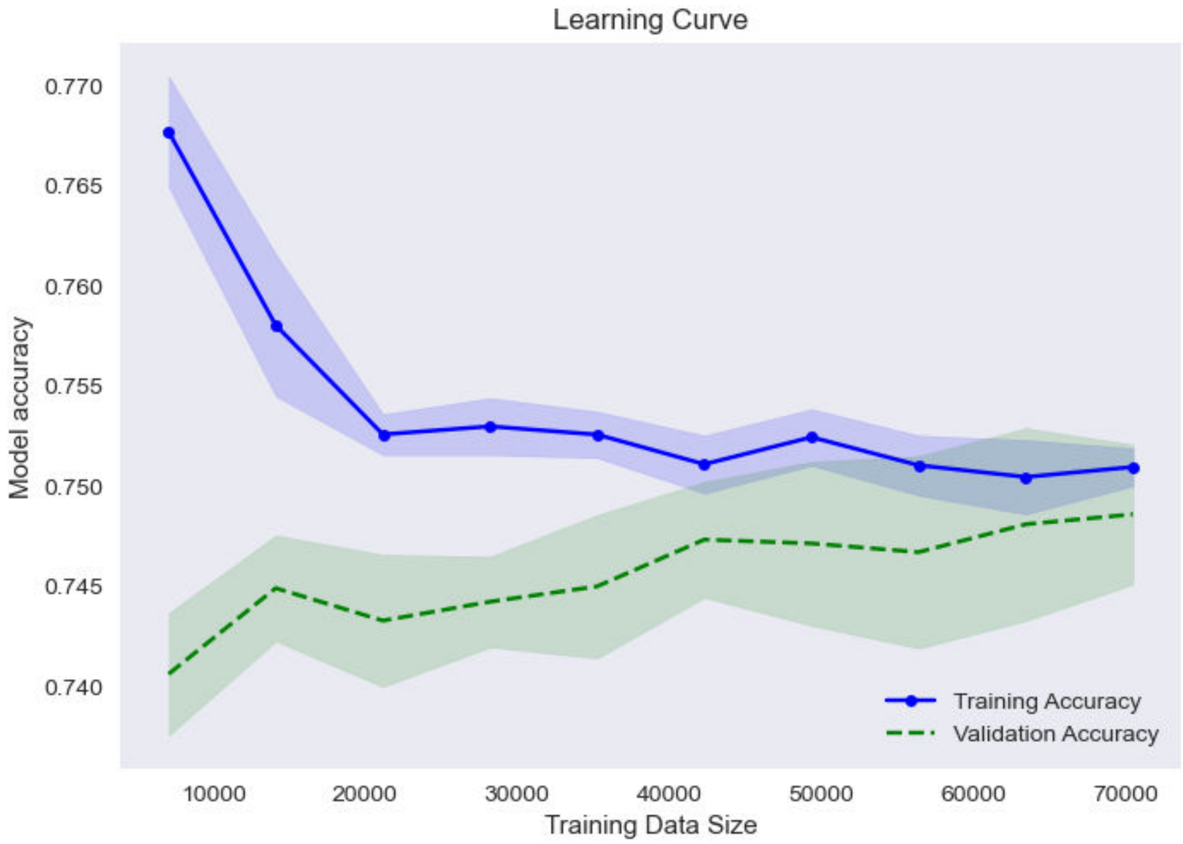
After Hyperparameter Tuning Confusion Matrix :



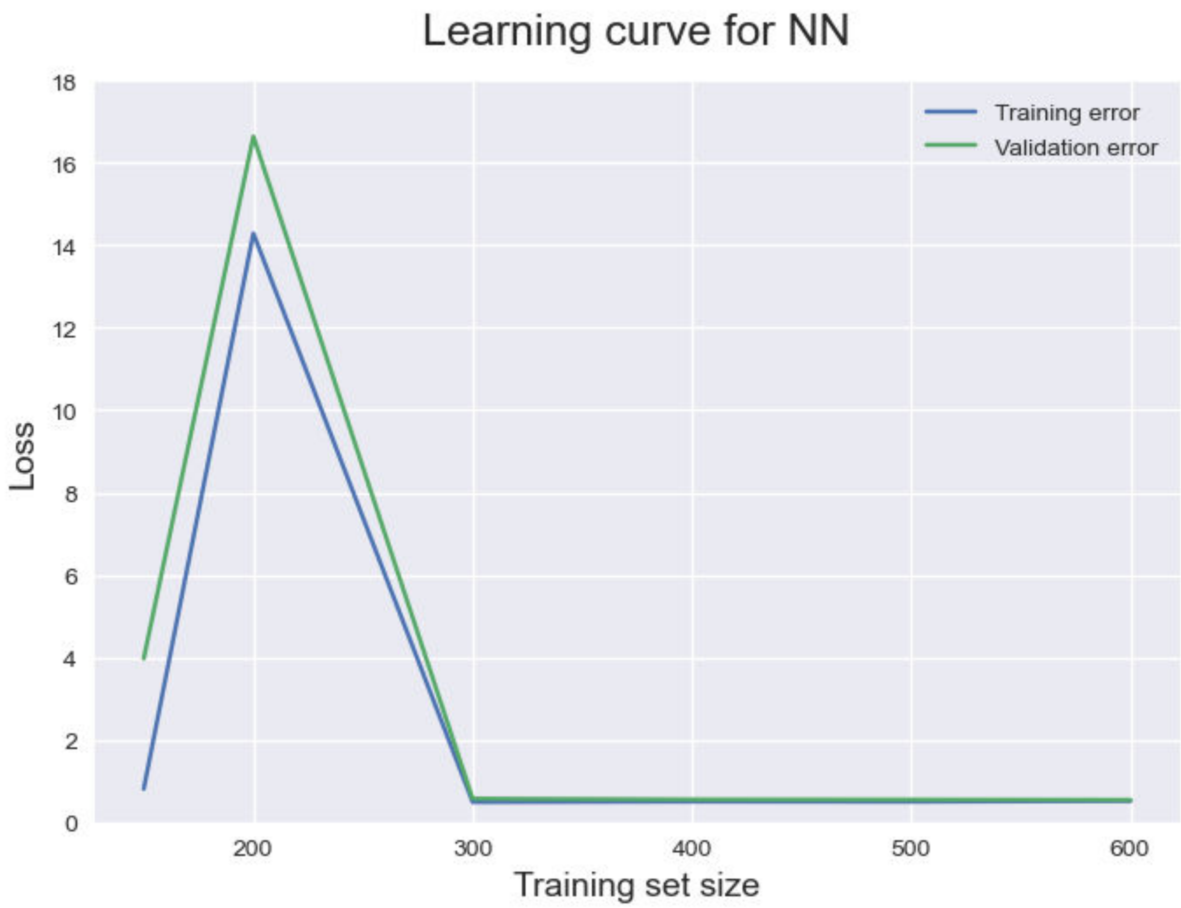
After the hyper parameter tuning, we got better results for predicting labels compared to default parameters of the neural network model.

4○ Learning curve analysis

Learning curves :

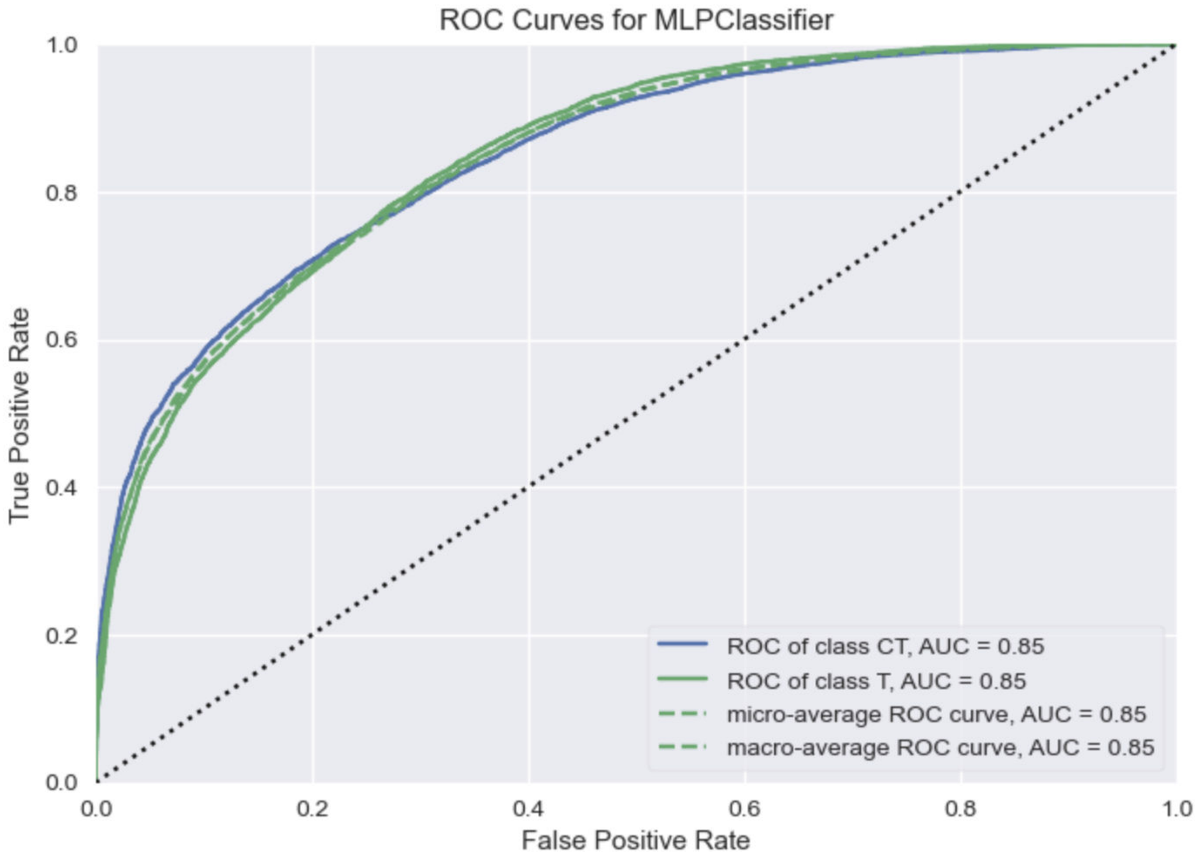


Here in this Learning curve of the model accuracy versus training data size. We can see that as we increase the number of training data the training accuracy goes down and in case of validation accuracy, the accuracy increases(not exponentially) with increasing number of Training data set.



Learning curve for the loss here we can say that validation error And the training error first increased, and then decreased as we increased size of training set, and after 300 training set size both training error and validation error were equal so it is considered as a good fit.

ROC Curve For Neural Network Model :



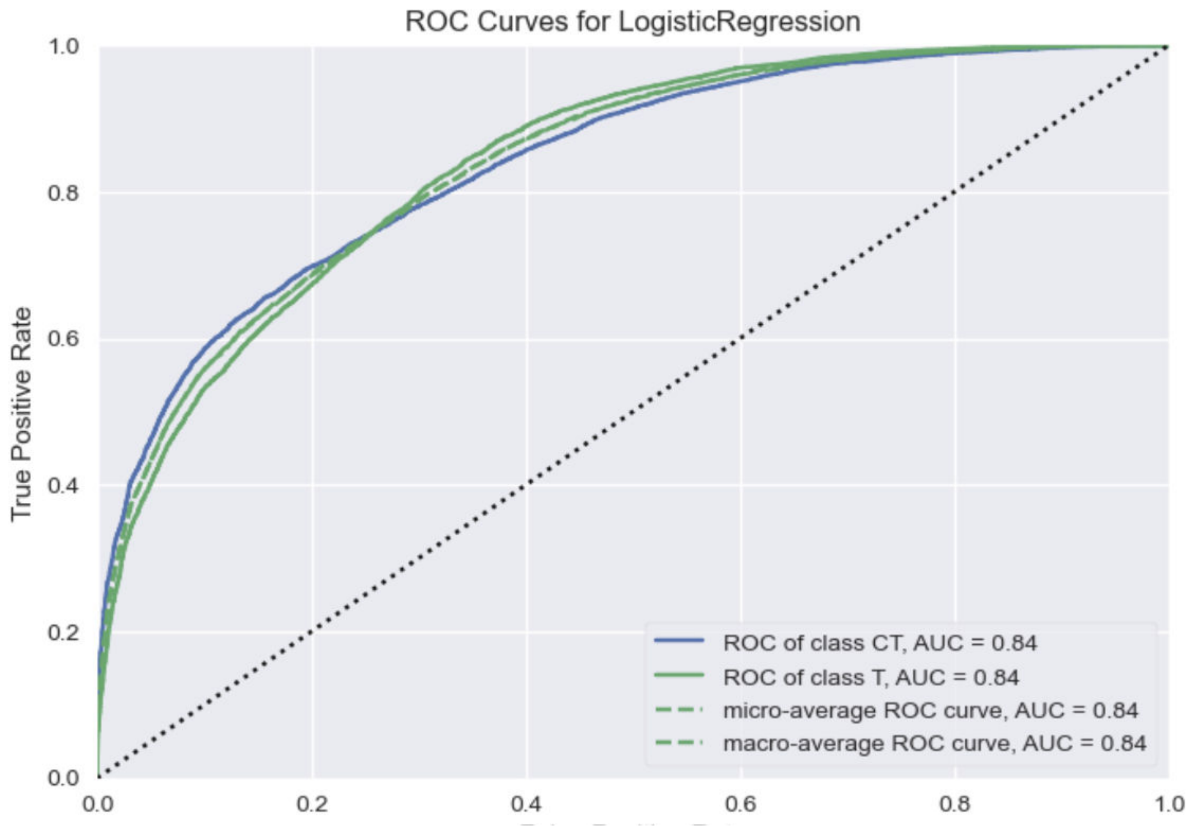
Here we can see that the AUC Area under the curve is 0.85. Which is better than the Logistic Regression Model.
There are two different lines for micro average ROC curve and macro average ROC curve.\Macro average is which gives equal weights to each category while micro averaging gives equal weight to each sample. So if we have nearly the same number of samples for each class, both macro and micro will provide the same score. So in the above ROC Curve, the AUC for macro and micro average is the same, which indicates that we have nearly the same number of samples for each class.

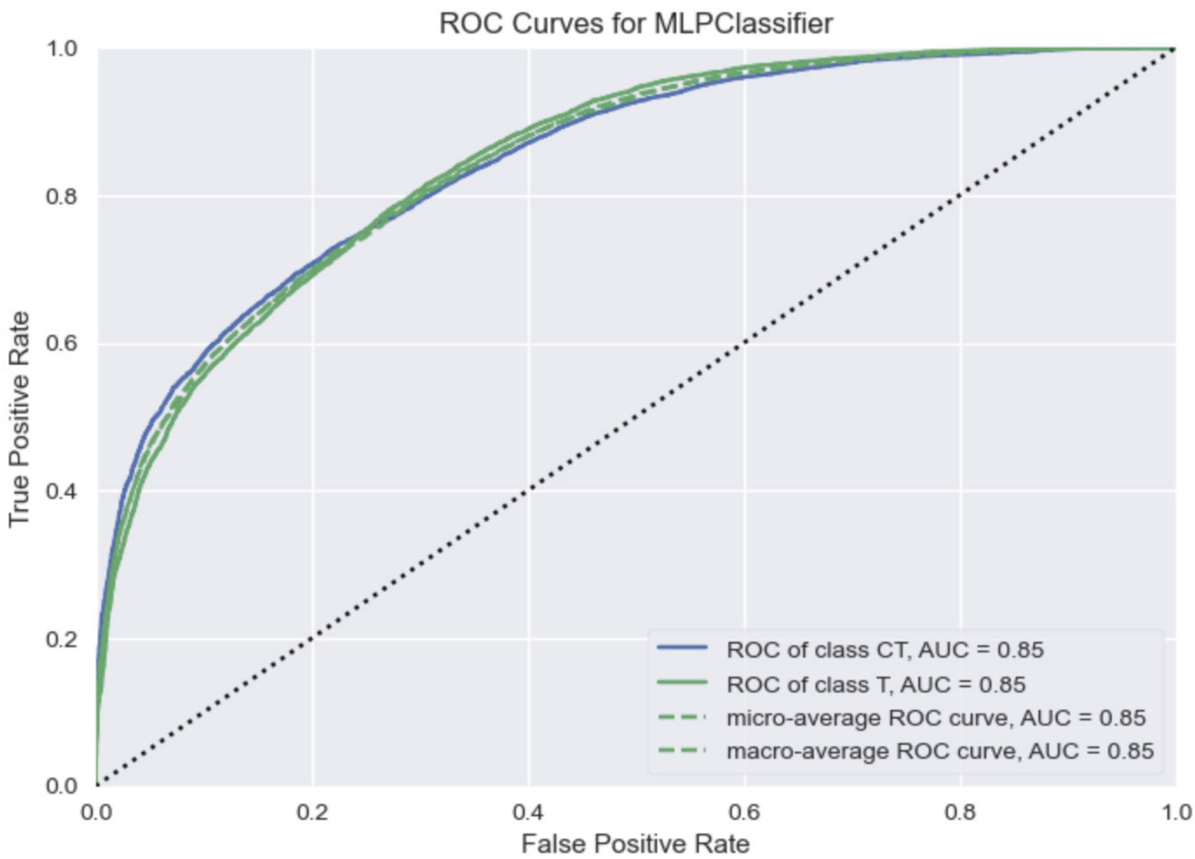
Classification Report:

	precision	recall	f1-score	support
0	0.74	0.78	0.76	9994
1	0.76	0.71	0.73	9592
accuracy			0.75	19586
macro avg	0.75	0.75	0.75	19586
weighted avg	0.75	0.75	0.75	19586

the accuracy of our Neural Network model comes to 75%. Where are precision for class zero is 0.74 and for class 1 its 0.78 while recall for class 0 is 0.76 and for class 1 its 0.71

ROC Curve Comparison :





Comparing the ROC curve for both models, we can see that neural network model has a better curve than logistic regression model the different between the 2 models is not that much but neural network model is performing better than logistic regression model as the area under the curve for logistic regression model is 0.84 and area under the curve for neural network model is 0.85.

5○ Performance and error analysis

As we can see, the neural network model is performing well than logistics regression model accuracy for predicting the winner of the round of logistic regression model it is 0.74 that is 74% and the for the neural network model the accuracy is 0.75 that is 75%. Meaning, our neural network model can predict the outcome 75% of the time.For a game like this in which the win rate of a particular team is roughly 50-50, right getting a almost 75% success rate , correct predictions is actually really good because games are not so one-sided we were definitely not be able to predict 90% of the time we can’t take into account the human error, the human ability to to make a comeback really so based on the stats in most games, we can in most games will be able to predict the winter successfully but there’s going to be that 25% of the games.