Business Data Science
(MIS382N Fall 2019)

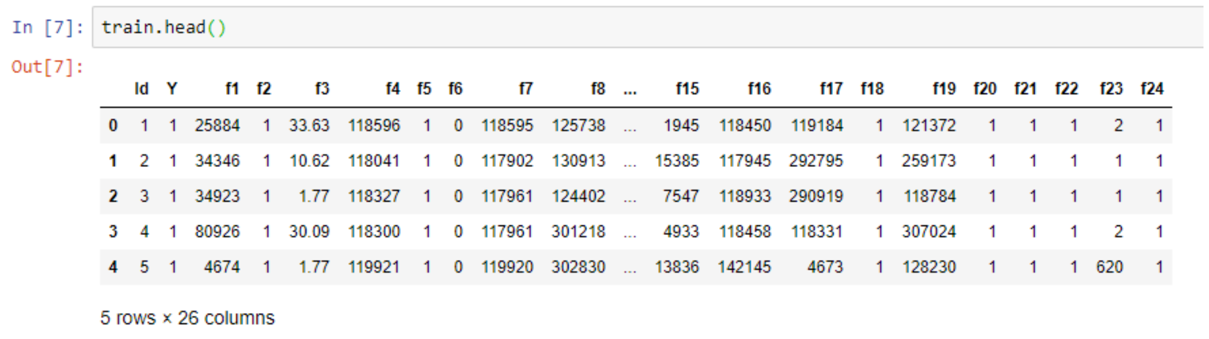
A report on the Kaggle Competition

By:
Shivang Arya

Contents

Data Exploration

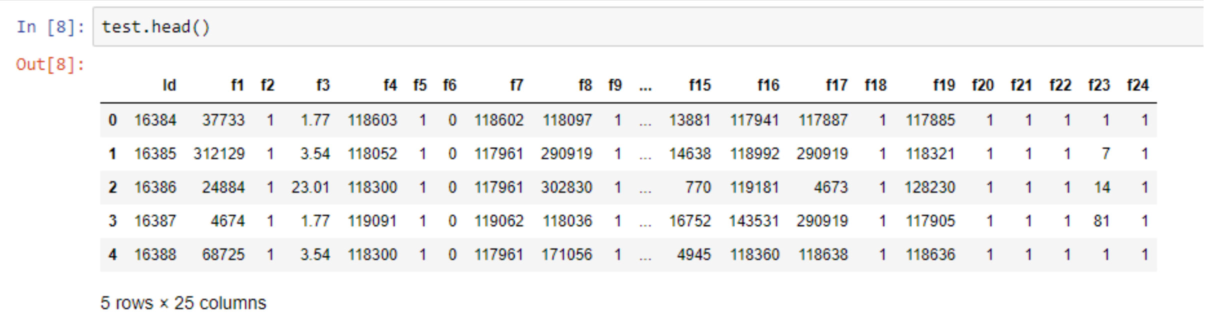
The first step was to check what the data looked like.



```
In [7]: train.head()
Out[7]:
```

	Id	Y	f1	f2	f3	f4	f5	f6	f7	f8	...	f15	f16	f17	f18	f19	f20	f21	f22	f23	f24
0	1	1	25884	1	33.63	118596	1	0	118595	125738	...	1945	118450	119184	1	121372	1	1	1	2	1
1	2	1	34346	1	10.62	118041	1	0	117902	130913	...	15385	117945	292795	1	259173	1	1	1	1	1
2	3	1	34923	1	1.77	118327	1	0	117961	124402	...	7547	118933	290919	1	118784	1	1	1	1	1
3	4	1	80926	1	30.09	118300	1	0	117961	301218	...	4933	118458	118331	1	307024	1	1	1	2	1
4	5	1	4674	1	1.77	119921	1	0	119920	302830	...	13836	142145	4673	1	128230	1	1	1	620	1

5 rows × 26 columns

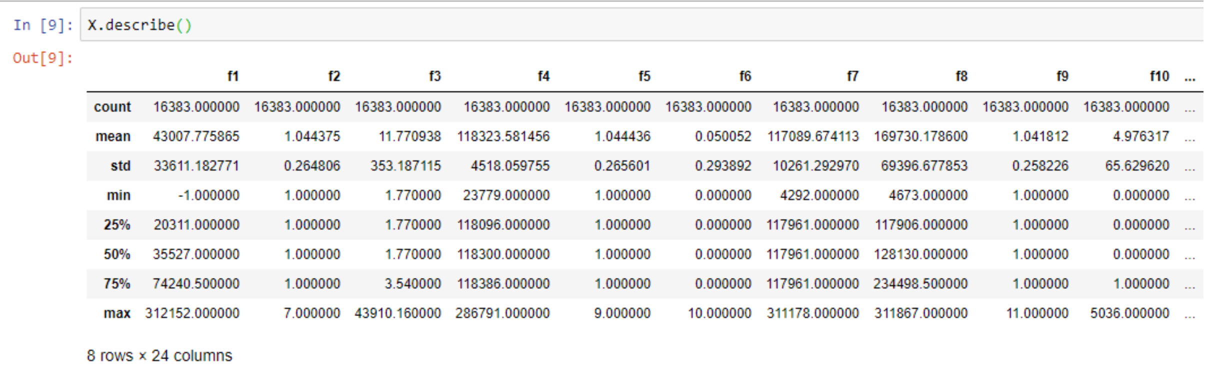


```
In [8]: test.head()
Out[8]:
```

	Id	f1	f2	f3	f4	f5	f6	f7	f8	f9	...	f15	f16	f17	f18	f19	f20	f21	f22	f23	f24
0	16384	37733	1	1.77	118603	1	0	118602	118097	1	...	13881	117941	117887	1	117885	1	1	1	1	1
1	16385	312129	1	3.54	118052	1	0	117961	290919	1	...	14638	118992	290919	1	118321	1	1	1	7	1
2	16386	24884	1	23.01	118300	1	0	117961	302830	1	...	770	119181	4673	1	128230	1	1	1	14	1
3	16387	4674	1	1.77	119091	1	0	119062	118036	1	...	16752	143531	290919	1	117905	1	1	1	81	1
4	16388	68725	1	3.54	118300	1	0	117961	171056	1	...	4945	118360	118638	1	118636	1	1	1	1	1

5 rows × 25 columns

Then I separated the X's (independent variables) and Y (dependent variable) in the train data and observed the descriptions of each column in the train data set.



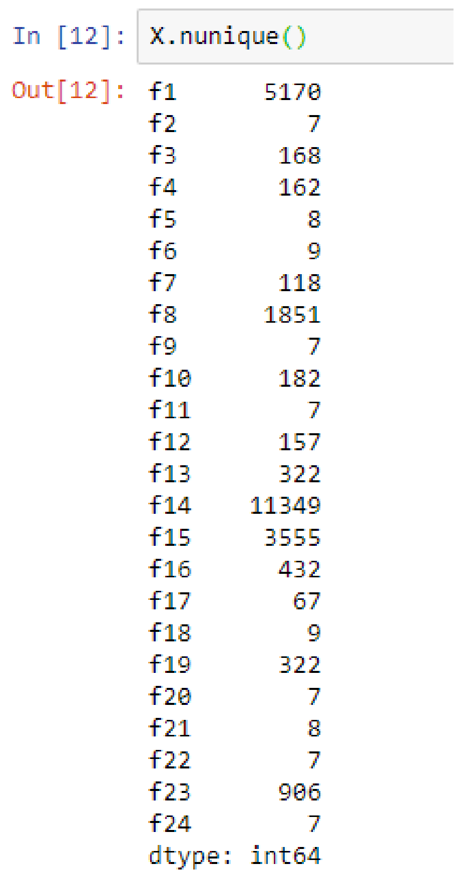
```
In [9]: X.describe()
Out[9]:
```

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	...
count	16383.000000	16383.000000	16383.000000	16383.000000	16383.000000	16383.000000	16383.000000	16383.000000	16383.000000	16383.000000	...
mean	43007.775865	1.044375	11.770938	118323.581456	1.044436	0.050052	117089.674113	169730.178600	1.041812	4.976317	...
std	33611.182771	0.264806	353.187115	4518.059755	0.265601	0.293892	10261.292970	69396.677853	0.258226	65.629620	...
min	-1.000000	1.000000	1.770000	23779.000000	1.000000	0.000000	4292.000000	4673.000000	1.000000	0.000000	...
25%	20311.000000	1.000000	1.770000	118096.000000	1.000000	0.000000	117961.000000	117906.000000	1.000000	0.000000	...
50%	35527.000000	1.000000	1.770000	118300.000000	1.000000	0.000000	117961.000000	128130.000000	1.000000	0.000000	...
75%	74240.500000	1.000000	3.540000	118386.000000	1.000000	0.000000	117961.000000	234498.500000	1.000000	1.000000	...
max	312152.000000	7.000000	43910.160000	286791.000000	9.000000	10.000000	311178.000000	311867.000000	11.000000	5036.000000	...

8 rows × 24 columns

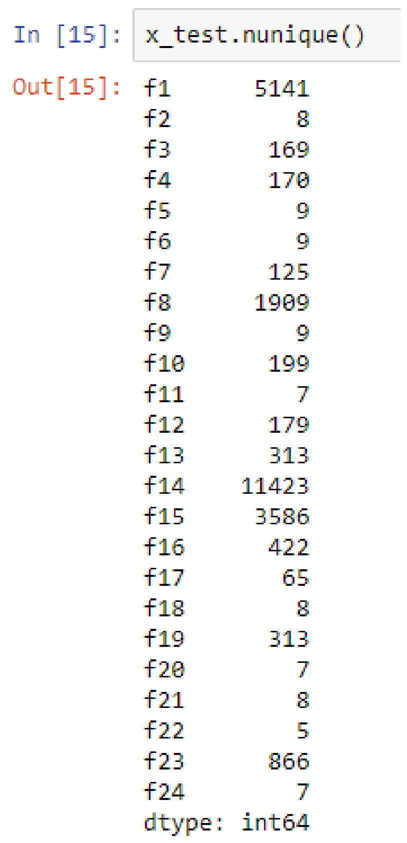
I removed the 'Id' variable from train as well as test data because an identification variable is different (increasing integer) for every observation and does not help with predictions.

Then, I went ahead and checked the number of unique values in all the columns of train and test data because most of the columns looked like categorical as they were integers between either 1 and 7, 1 and 8 or 1 and 9. If these columns were indeed categorical, I would have to get dummies for these columns in the train and test data.



```
In [12]: X.nunique()

Out[12]: f1       5170
         f2          7
         f3        168
         f4        162
         f5          8
         f6          9
         f7        118
         f8       1851
         f9          7
         f10       182
         f11         7
         f12       157
         f13       322
         f14     11349
         f15      3555
         f16       432
         f17        67
         f18         9
         f19       322
         f20         7
         f21         8
         f22         7
         f23       906
         f24         7
         dtype: int64
```

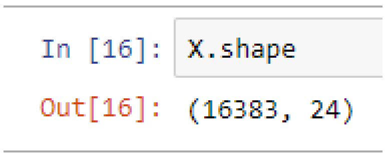
I checked for the same classes in test data.



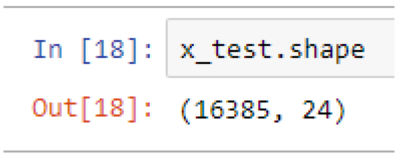
```
In [15]: x_test.nunique()

Out[15]: f1       5141
         f2          8
         f3        169
         f4        170
         f5          9
         f6          9
         f7        125
         f8       1909
         f9          9
         f10       199
         f11         7
         f12       179
         f13       313
         f14     11423
         f15      3586
         f16       422
         f17        65
         f18         8
         f19       313
         f20         7
         f21         8
         f22         5
         f23       866
         f24         7
         dtype: int64
```

To my surprise, the variables I considered categorical had different classes in train and test data sets which implied that the dummies I create for train data will not always work with test data. After realizing this, I decided to drop the idea of getting dummies and considered all variables as continuous.

I checked the shape of train and test data to check the difference in number of observations.

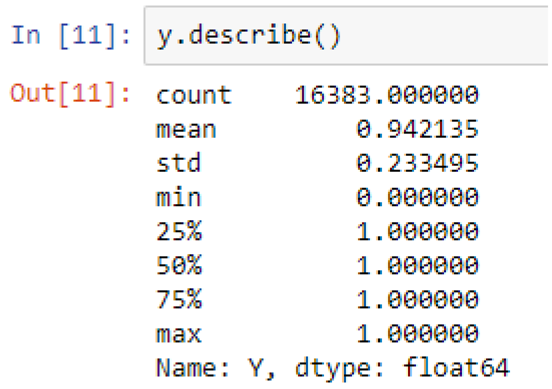


```
In [16]: X.shape
Out[16]: (16383, 24)
```



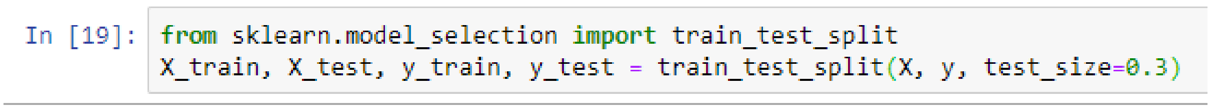
```
In [18]: x_test.shape
Out[18]: (16385, 24)
```

Since they were really similar, I made an assumption that the average of Y's from the train data would be a good estimator of average of Y's (predicted) in the test data.



```
In [11]: y.describe()
Out[11]: count    16383.000000
         mean         0.942135
         std          0.233495
         min          0.000000
         25%          1.000000
         50%          1.000000
         75%          1.000000
         max          1.000000
         Name: Y, dtype: float64
```

I went ahead to split train data further into train and test data to get a rough estimate for the score on my predictions.



```
In [19]: from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

The variable terminology after all this is as follows:

train: the train data from kaggle

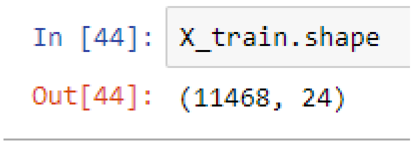
test: the test data from kaggle

X: independent variables in train data without 'Id' column

y: dependent variable in train data

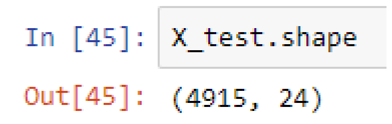
x_test: independent variables in test data without 'Id' column

X_train: a split of X into training data



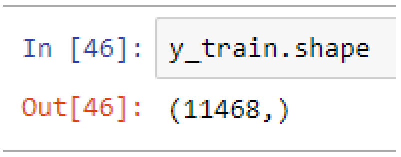
```
In [44]: X_train.shape
Out[44]: (11468, 24)
```

X_test: a split of X into test data



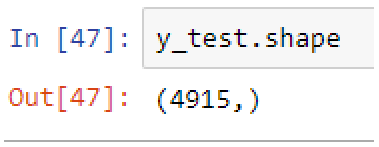
```
In [45]: X_test.shape
Out[45]: (4915, 24)
```

y_train: a split of y into training data



```
In [46]: y_train.shape
Out[46]: (11468,)
```

y_test: a split of y into test data



```
In [47]: y_test.shape
Out[47]: (4915,)
```

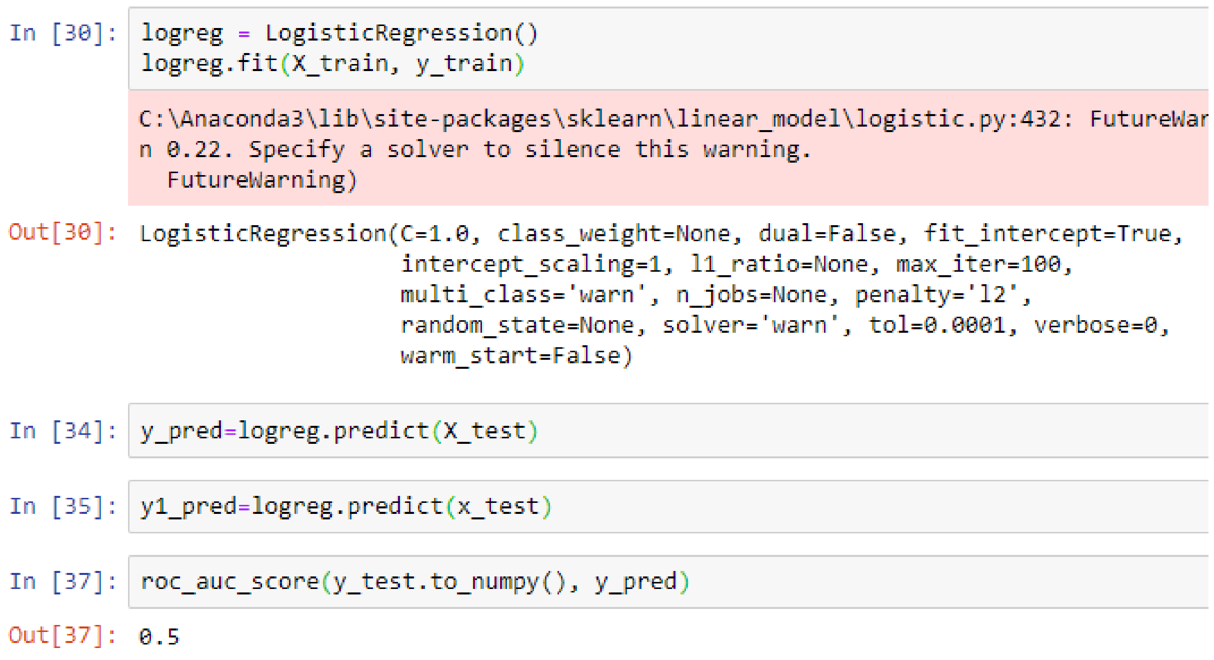
The variables in X were not correlated at all, hence I dropped the idea of Principal Component Analysis.

Initial Attempts

Initially, I was not entirely certain of the objective (to maximize area under the receiver operating characteristics curve). I saw this as a general classification problem. So, my first 7 or 8 attempts were made with hard labels (0's and 1's).

Logistic Regression

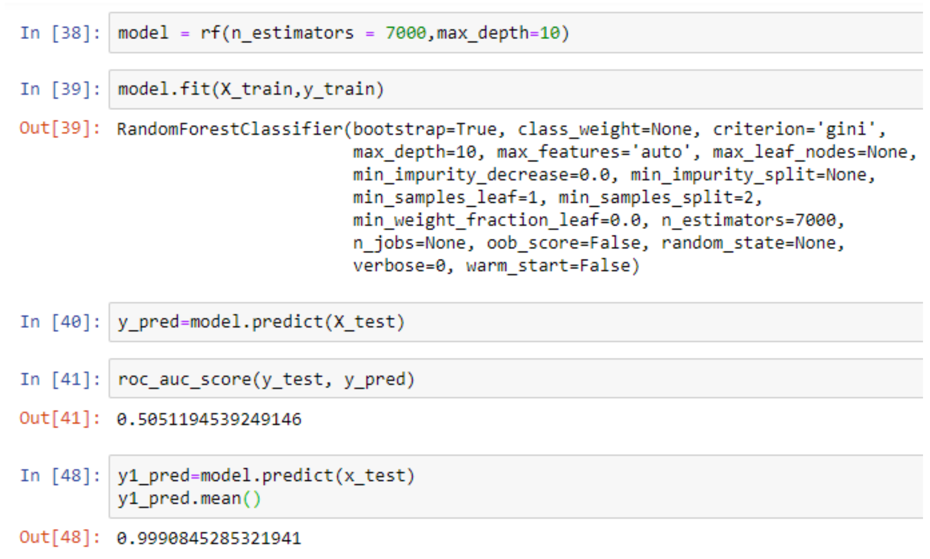
I checked the area under ROC curve using logistic regression on my test set and it was really low, so I decided to not waste any submissions on this.



```
In [30]: logreg = LogisticRegression()
         logreg.fit(X_train, y_train)

         C:\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWar
         n 0.22. Specify a solver to silence this warning.
           FutureWarning)

Out[30]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                            intercept_scaling=1, l1_ratio=None, max_iter=100,
                            multi_class='warn', n_jobs=None, penalty='l2',
                            random_state=None, solver='warn', tol=0.0001, verbose=0,
                            warm_start=False)

In [34]: y_pred=logreg.predict(X_test)

In [35]: y1_pred=logreg.predict(x_test)

In [37]: roc_auc_score(y_test.to_numpy(), y_pred)

Out[37]: 0.5
```

Random Forest

Random forest gave me better area under ROC score and my initial submissions were made using a Random Forest classifier with hard labels.

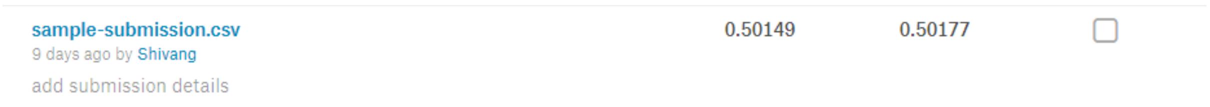


```
In [38]: model = rf(n_estimators = 7000,max_depth=10)

In [39]: model.fit(X_train,y_train)

Out[39]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                                max_depth=10, max_features='auto', 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, n_estimators=7000,
                                n_jobs=None, oob_score=False, random_state=None,
                                verbose=0, warm_start=False)

In [40]: y_pred=model.predict(X_test)

In [41]: roc_auc_score(y_test, y_pred)

Out[41]: 0.5051194539249146

In [48]: y1_pred=model.predict(x_test)
         y1_pred.mean()

Out[48]: 0.9990845285321941
```

The mean was still very high compared to what I was expecting, based on the mean of ‘y’.

After this, I ran the same model on the entire dataset (i.e, X and y) and predicted x_test to submit on kaggle.

This is how I scored on Kaggle using my first Random Forest:

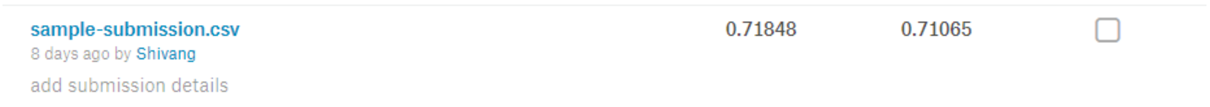


sample-submission.csv
9 days ago by Shivang
add submission details

0.50149 0.50177

I went on to further tune the random forest classifier as I realized that increasing the number of estimators (number of trees used for bagging) was giving me a better Area under ROC curve score.

The best score I could reach with 70,000 estimators and a maximum tree depth of 15 using a Random Forest Classifier with hard labels was:



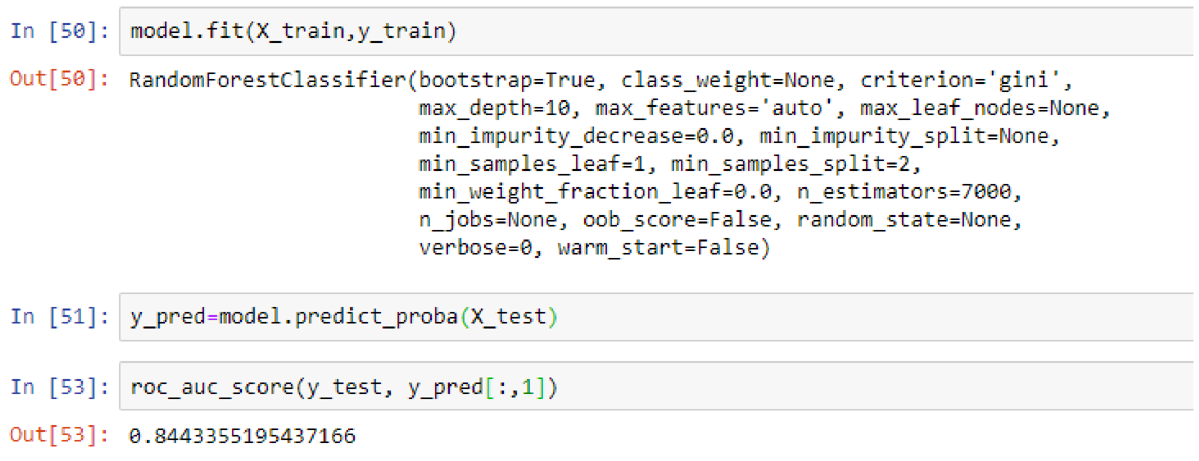
sample-submission.csv
8 days ago by Shivang
add submission details

0.71848 0.71065

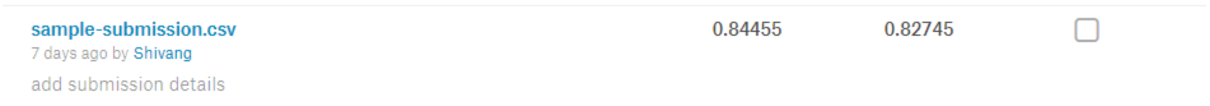
By this time I had realized that I was doing something fundamentally incorrect because I was observing scores of 0.88 and above on Kaggle so I decided to go back to the problem statement.

Under the Overview tab, I saw the Evaluation section understood my mistake, as the section clearly mentioned to use soft labels instead of rounding them to 0’s and 1’s. After figuring this out I went ahead and read more about the metric (Area under ROC curve) to understand it better.

I used Random Forest Classifier again with all the features and used predict_proba instead of predict and got the following results:



```
In [50]: model.fit(X_train,y_train)

Out[50]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=10, max_features='auto', 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, n_estimators=7000,
                     n_jobs=None, oob_score=False, random_state=None,
                     verbose=0, warm_start=False)

In [51]: y_pred=model.predict_proba(X_test)

In [53]: roc_auc_score(y_test, y_pred[:,1])

Out[53]: 0.8443355195437166
```



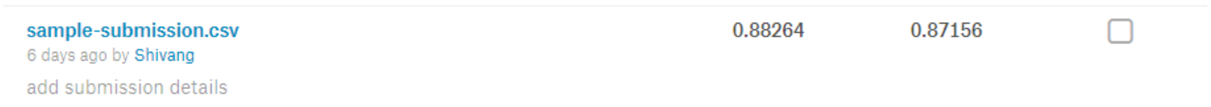
sample-submission.csv
7 days ago by Shivang
add submission details

0.84455 0.82745

Feature Selection and Cross Validation

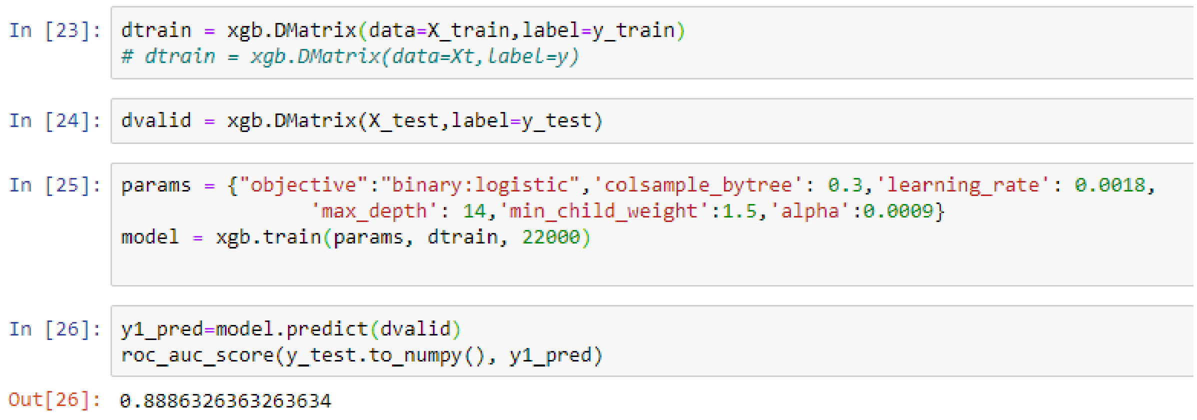
Gradient Boosting Decision Trees (XGBoost)

I tried the same parameters as Random Forest on XGBClassifier and got a better result, thereby concluding that XGBoost was a better way to tackle the problem.



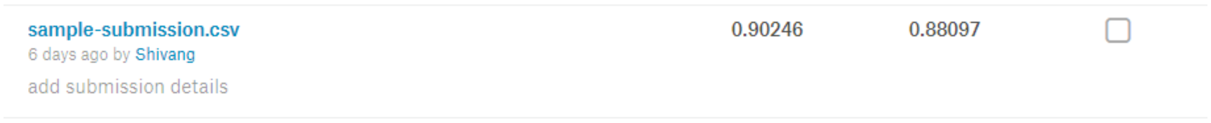
```
sample-submission.csv                                    0.88264        0.87156
6 days ago by Shivang
add submission details
```

While reading more about XGBoost, I came across a DataCamp article which stated “convert the dataset into an optimized data structure called Dmatrix that XGBoost supports and gives it acclaimed performance and efficiency gains”. So I tried it, tuned the parameters manually and the difference was evident:



```
In [23]: dtrain = xgb.DMatrix(data=X_train,label=y_train)
         # dtrain = xgb.DMatrix(data=Xt,label=y)

In [24]: dvalid = xgb.DMatrix(X_test,label=y_test)

In [25]: params = {"objective":"binary:logistic",'colsample_bytree': 0.3,'learning_rate': 0.0018,
                         'max_depth': 14,'min_child_weight':1.5,'alpha':0.0009}
         model = xgb.train(params, dtrain, 22000)

In [26]: y1_pred=model.predict(dvalid)
         roc_auc_score(y_test.to_numpy(), y1_pred)

Out[26]: 0.8886326363263634
```

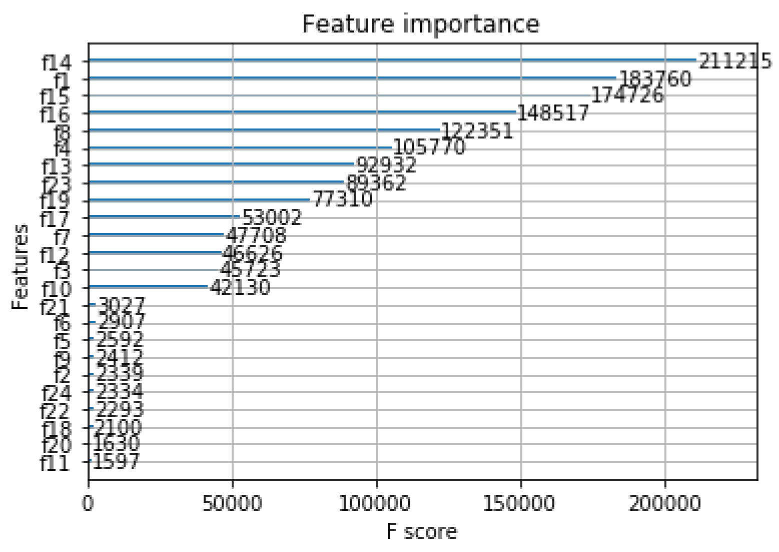
^This was not a classifier so I didn’t need to use predict_proba. Also, num_boost_round is set to 22,000 in order to get estimates from a large number of trees to and consider maximum variation in features.



```
sample-submission.csv                                    0.90246        0.88097
6 days ago by Shivang
add submission details
```

The next step according to me was to get the right features as I believed that not all features were contributing well to the prediction. For this, I used the plot_importance attribute of xgb.

```
In [28]: xgb.plot_importance(model)
         plt.rcParams['figure.figsize'] = [5, 5]
         plt.show()
```

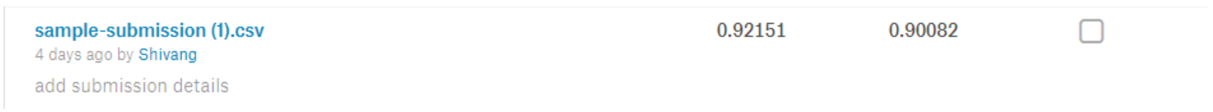


Based on these F scores, I decided to use the top six most important features and go on adding features till I reach a maximum value of Area under ROC curve keeping the parameters of my XGBoost model the same for every iteration. I realized that the top nine features were giving me the best results. Area under ROC score was lower for 8 or 10 features compared to 9.

Using these features along with tuned parameters, I got significantly better results.

```
In [65]: Xt_train=X_train.loc[:,('f14','f15','f1','f16','f8','f4','f13','f23','f19')]
         Xt_test=X_test.loc[:,('f14','f15','f1','f16','f8','f4','f13','f23','f19')]
         xt_test=x_test.loc[:,('f14','f15','f1','f16','f8','f4','f13','f23','f19')]

In [66]: Xt=X.loc[:,('f14','f15','f1','f16','f8','f4','f13','f23','f19')]

In [68]: dtrain = xgb.DMatrix(data=Xt_train,label=y_train)
         # dtrain = xgb.DMatrix(data=Xt,label=y)

In [69]: dvalid = xgb.DMatrix(Xt_test,label=y_test)

In [91]: params = {"objective":"binary:logistic",'colsample_bytree': 0.3,'learning_rate': 0.085,
                 'max_depth': 16,'min_child_weight':1.5,'alpha':0.0009}
         model = xgb.train(params, dtrain, 222)

In [92]: y1_pred=model.predict(dvalid)
         roc_auc_score(y_test.to_numpy(), y1_pred)

Out[92]: 0.9001791402743667
```

After running this model on the entire train dataset, I got the following score on Kaggle:



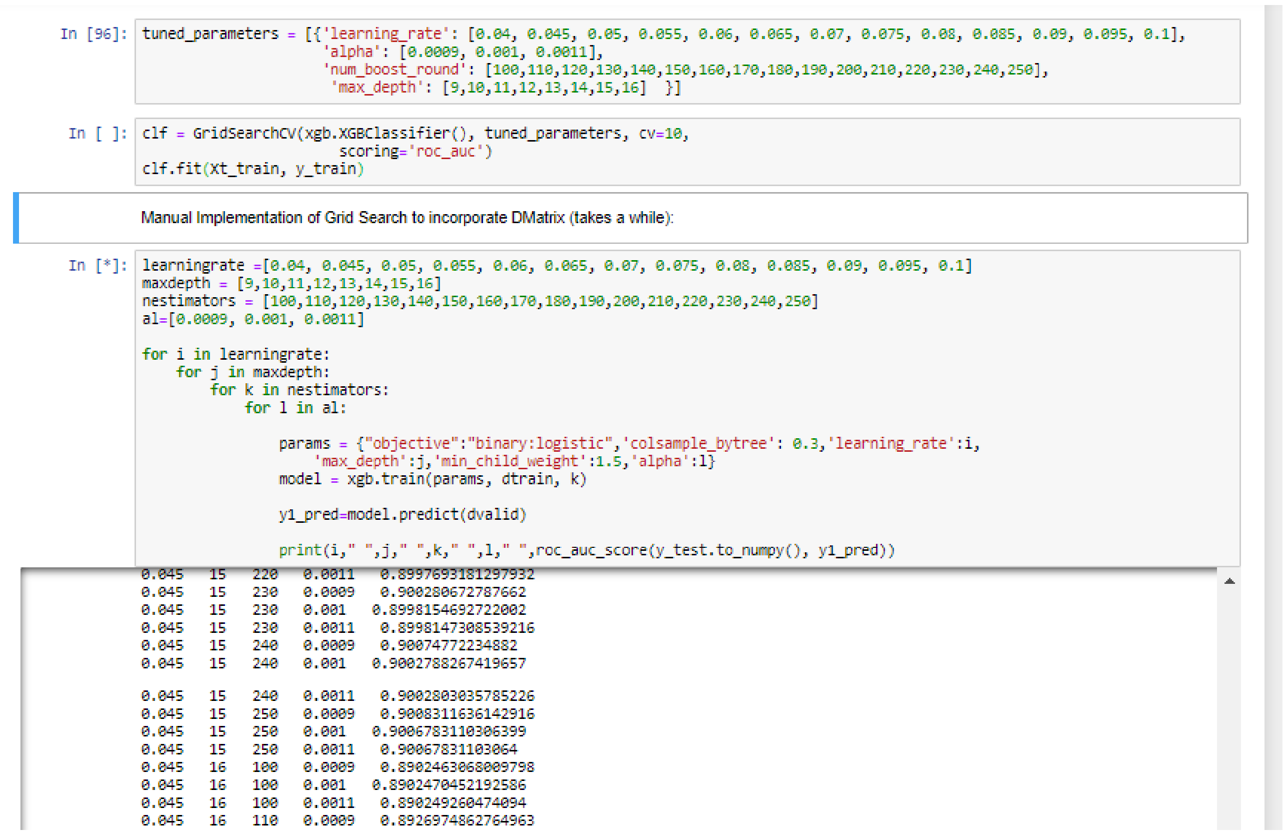
sample-submission (1).csv 0.92151 0.90082 ☐
4 days ago by Shivang
add submission details

Grid Search CV

While manually tuning the parameters, I discovered a relationship between n_estimators (or num_boost_round) and learning rate. As the learning rate decreased, the n_estimators had to be increased in order to achieve the best score on area under ROC curve.

After a lot of manual parameter estimation for XGBoost, I had finally narrowed down the list of parameters enough for running GridSearchCV to the following:

1. learning_rate: between 0.04 and 0.1 with an increment of 0.005 (i.e. 0.04, 0.045, 0.05, 0.055, 0.06, 0.065, 0.07, 0.075, 0.08, 0.085, 0.09, 0.095, 0.1)
2. n_estimators: between 100 and 250 with an increment of 10 (i.e. 100, 110, 120, 130, 140, 150, 160…..)
3. max_depth: between 9 and 16 (i.e. 9, 10, 11, 12, 13, 14, 15, 16)
4. alpha: 0.0009, 0.001, 0.0011



```
In [96]: tuned_parameters = [{'learning_rate': [0.04, 0.045, 0.05, 0.055, 0.06, 0.065, 0.07, 0.075, 0.08, 0.085, 0.09, 0.095, 0.1],
                             'alpha': [0.0009, 0.001, 0.0011],
                             'num_boost_round': [100,110,120,130,140,150,160,170,180,190,200,210,220,230,240,250],
                              'max_depth': [9,10,11,12,13,14,15,16]  }]
```

```
In [ ]: clf = GridSearchCV(xgb.XGBClassifier(), tuned_parameters, cv=10,
                           scoring='roc_auc')
        clf.fit(Xt_train, y_train)
```

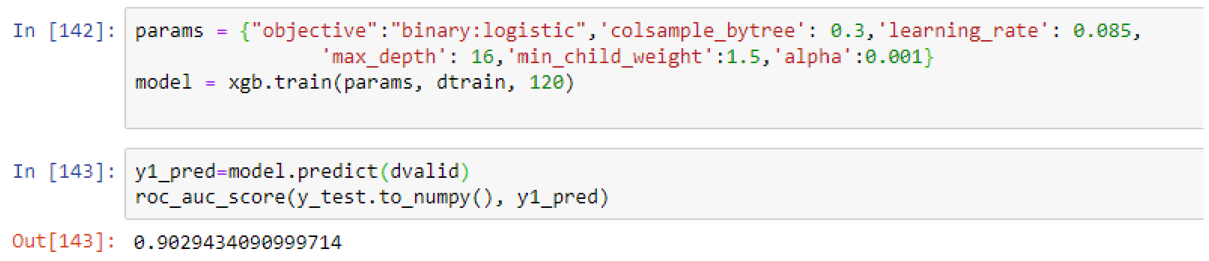
Manual Implementation of Grid Search to incorporate DMatrix (takes a while):

```
In [*]: learningrate =[0.04, 0.045, 0.05, 0.055, 0.06, 0.065, 0.07, 0.075, 0.08, 0.085, 0.09, 0.095, 0.1]
        maxdepth = [9,10,11,12,13,14,15,16]
        nestimators = [100,110,120,130,140,150,160,170,180,190,200,210,220,230,240,250]
        al=[0.0009, 0.001, 0.0011]

        for i in learningrate:
            for j in maxdepth:
                for k in nestimators:
                    for l in al:

                        params = {"objective":"binary:logistic",'colsample_bytree': 0.3,'learning_rate':i,
                            'max_depth':j,'min_child_weight':1.5,'alpha':l}
                        model = xgb.train(params, dtrain, k)

                        y1_pred=model.predict(dvalid)

                        print(i," ",j," ",k," ",l," ",roc_auc_score(y_test.to_numpy(), y1_pred))
```

```
0.045   15   220   0.0011    0.8997693181297932
0.045   15   230   0.0009    0.900280672787662
0.045   15   230   0.001   0.8998154692722002
0.045   15   230   0.0011    0.8998147308539216
0.045   15   240   0.0009    0.90074772234882
0.045   15   240   0.001   0.9002788267419657

0.045   15   240   0.0011    0.9002803035785226
0.045   15   250   0.0009    0.9008311636142916
0.045   15   250   0.001   0.9006783110306399
0.045   15   250   0.0011    0.90067831103064
0.045   16   100   0.0009    0.8902463068009798
0.045   16   100   0.001   0.8902470452192586
0.045   16   100   0.0011    0.890249260474094
0.045   16   110   0.0009    0.8926974862764963
```

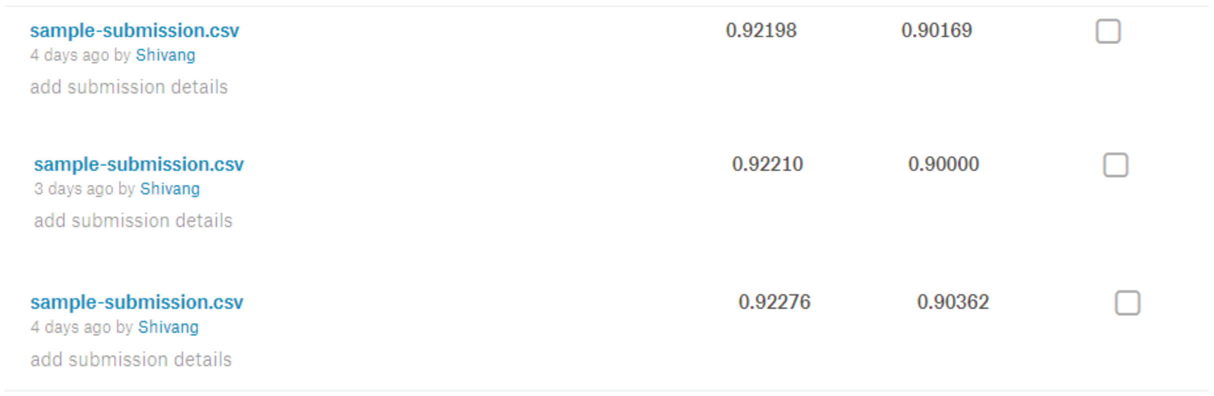
I got the following three sets of parameters that gave the best score:

learning_rate	n_estimators	max_depth	alpha
0.085	120	16	0.001
0.075	222	15	0.0011
0.080	200	16	0.0011



```
In [142]: params = {"objective":"binary:logistic",'colsample_bytree': 0.3,'learning_rate': 0.085,
                          'max_depth': 16,'min_child_weight':1.5,'alpha':0.001}
          model = xgb.train(params, dtrain, 120)

In [143]: y1_pred=model.predict(dvalid)
          roc_auc_score(y_test.to_numpy(), y1_pred)

Out[143]: 0.9029434090999714
```

I made a submission for a model with each of the above sets of parameters trained on the entire training data and got the following results on kaggle:



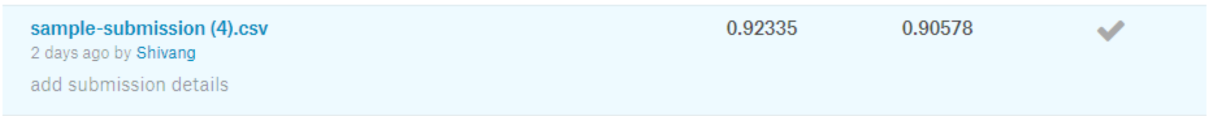
```
sample-submission.csv                      0.92198      0.90169
4 days ago by Shivang
add submission details

sample-submission.csv                      0.92210      0.90000
3 days ago by Shivang
add submission details

sample-submission.csv                      0.92276      0.90362
4 days ago by Shivang
add submission details
```

Ensemble

After everything seen above, I figured that XGBoost was at its limit as nothing I tried further improved the area under ROC curve. So I had to take a new approach to improve my models.

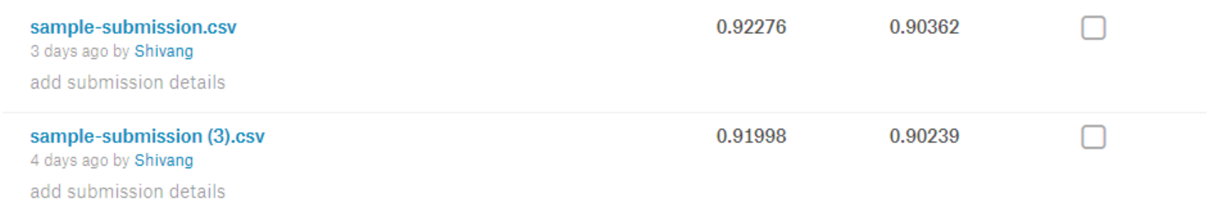
I was already at the last day of submission when I decided to go for an ensemble of my own best models as I came across an article on analyticsvidhya.com where I saw the author take a mean of three different model predictions.

I attempted the same but instead of taking the time to write code for it, I used Excel and averaged my two best submissions and got a significant improvement in my previous best score:



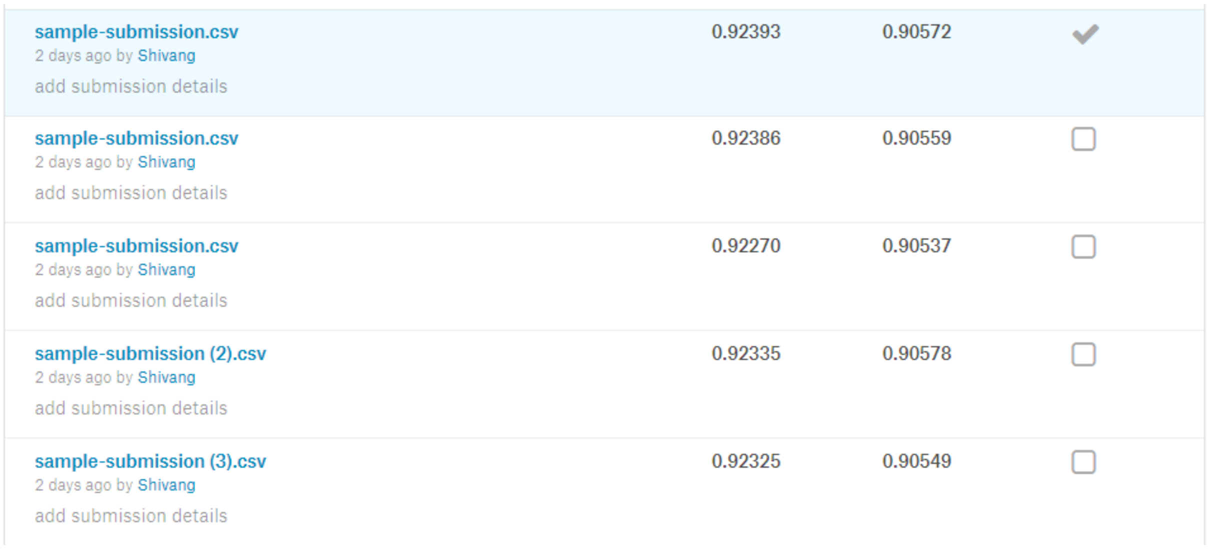
sample-submission (4).csv 2 days ago by Shivang add submission details	0.92335	0.90578	✓

^The predictions in this submission were an average of these two submissions:



sample-submission.csv 3 days ago by Shivang add submission details	0.92276	0.90362	☐
sample-submission (3).csv 4 days ago by Shivang add submission details	0.91998	0.90239	☐

So I decided to spend all my remaining submissions on the last day on ensemble models of my previous submissions.



sample-submission.csv 2 days ago by Shivang add submission details	0.92393	0.90572	✓
sample-submission.csv 2 days ago by Shivang add submission details	0.92386	0.90559	☐
sample-submission.csv 2 days ago by Shivang add submission details	0.92270	0.90537	☐
sample-submission (2).csv 2 days ago by Shivang add submission details	0.92335	0.90578	☐
sample-submission (3).csv 2 days ago by Shivang add submission details	0.92325	0.90549	☐

My final submission was an ensemble of all my submission that got area under ROC score greater than or equal to 0.90, so a total of five to six models. I believe that this is the reason I managed to get a better score on the private leaderboard as averaging more number of models reduced overfitting to the test data and generalized my predictions better.

Summary

What I tried, why I tried it and did it work?

As hinted by the professor in the class, my first two attempts were Logistic Regression and Random Forest.

1. **Logistic Regression:** I tried it because it was one of the first classification techniques I learnt and wanted to see what kind of results I could achieve using it. This approach did not work well. I suspect this occurred because the distribution of classes was not even. There were way too many 1's than 0's. Logistic regression was not able to fir the 'S' curve correctly and predicted all values as 1 or very close to 1.

2. **Random Forest Classifier:** A really good technique to use with unsupervised classification. Bagging of features and criterion of selecting the best feature to split makes it really convenient to use when you are not familiar with the dataset. This worked much better than Logistic Regression as it was able to identify the best criteria to split on.

3. **Decision Tree Classifier:** Easy to visualize, works well with categorical data but not the best dataset to use it. Since the data was continuous, decision trees did not work well.

4. **XGBoost:** Gradient boosting in decision trees to reduce bias and variance. Seemed like the logical next step after Random Forest and Decision Tree Classifiers. This was the single most successful algorithm with the data given. The concept of boosting to find optimum betas applied really well to this data.

5. **Ensemble:** To average out my best models and reduce overfitting on 30% of the test data used to evaluate the public leaderboard. Worked unexpectedly well. I was not expecting so much of a difference on the area under ROC score in case of public and private leaderboard.