EE 460J - Kaggle Report

Angshuman Chakraborty (ac72765)

MS Operations Research and Industrial Engineering, UT Austin

Dated: 29th October 2021

Private Leaderboard: - 15th Position

13	▲ 4	Sriraman Iyengar	0.92439	20	4d
14	▼ 2	NickRiveira	0.92343	31	4d
15	—	Angshuman Chakraborty	0.92229	33	4d
16	▼ 2	Jeffrey Liu	0.92097	14	4d

Public Leaderboard: - 15th Position

14	Jeffrey Liu	0.91209	14	4d
15	Angshuman Chakraborty	0.91188	33	4d

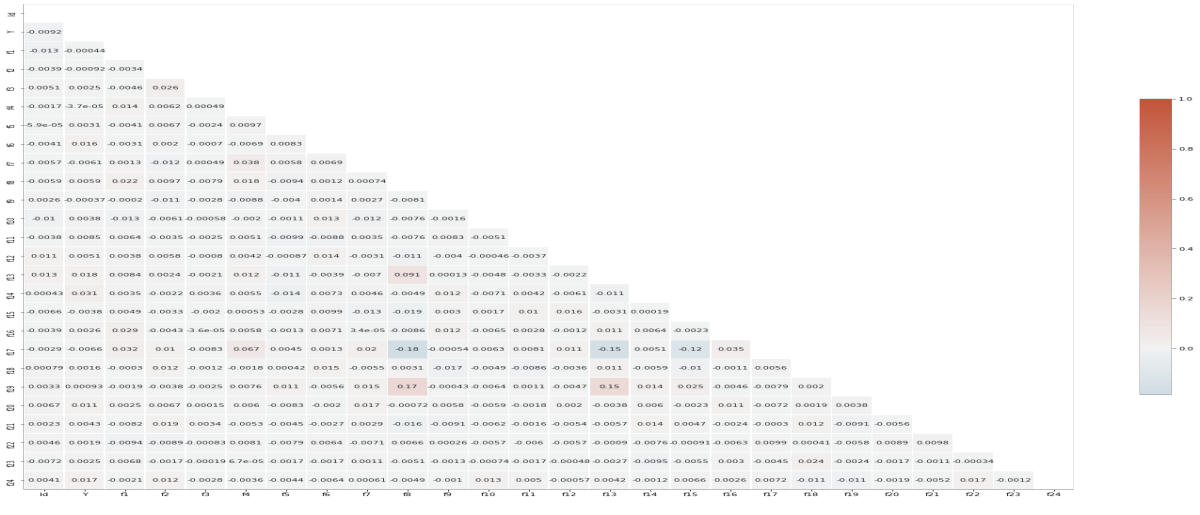
Introduction

The Kaggle project was of binary classification type and the test accuracy metric for evaluation used was the ROC – AUC score (also called as Area Under the Curve). The train and test dataset were almost equally split with ~ 16000 rows and 24 features. Various ML models such as Random Forests, Decision Tress, Gradient Boosting (including XG Boost) were primarily deployed for training on the train dataset.

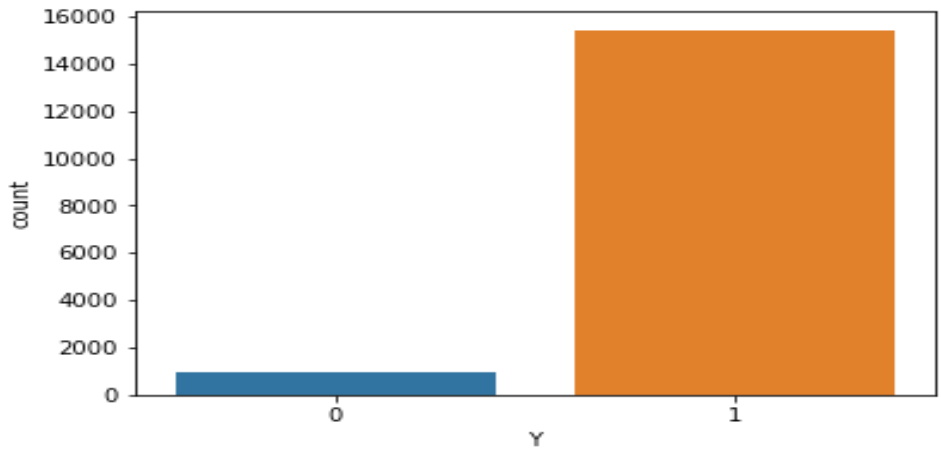
The report begins with a description of the steps undertaken for data preprocessing and feature engineering and presents my analysis, findings, modeling approach, and hyper-parameter tuning aspects for the Kaggle competition. It touches upon various techniques and tricks such as blending, oversampling using SMOTE, Knowledge Distillation and Pseudo-labelling that I tried to get a private leaderboard score of 0.92229.

Data Exploration and Feature Engineering

- Check for missing data: There were no missing values both in train and test datasets.
- Looked at the unique values for each feature to identify which of the features could be potential categorical features and then one hot encoded those features
- One hot encoded the data for those features which had less than 12 unique values on the entire dataset (train and test dataset combined)
- Created heat map to visualize correlation between 24 features and Y-labels.
- **Inference**: Features are not strongly correlated with Y label and among each other



- As seen from the count plot of Y-labels the data is unbalanced. 94.2% of the data was 1 and only 5.8% of the data was labeled as 0.



MACHINE LEARNING MODELS

Logistic Regression:

I decided to start with a logistic regression model and used LogisticRegressionCV with solvers 'lbfgs' and 'saga' and L1 and L2 penalties. As expected, the model had a pretty low roc-auc score.

Random Forests:

- I then moved onto Random Forest and ran a Random Forest Classifier on the one hot encoded data with 100 features.
- Employed Randomized Search CV on a grid of parameters to train the Random Forest model with cross validation split
- Performed a Grid Search on a smaller interval of the parameters to further tune the hyperparameters
- Got a Kaggle score of 0.8758 on this submission

```
#Grid of parameters to train the Random Forest over

param_grid = {
    'rfc__n_estimators': [10,50,100,200,300,500,700,1000],
    'rfc__max_depth': [10,20,30,40,50,60,80,100,None],
    'rfc__min_samples_split': [2, 5, 10,20,25,50],
    'rfc__min_samples_leaf': [1, 2, 4,6,8]
}
pipe = Pipeline([('rfc', RandomForestClassifier())])

rforest = RandomizedSearchCV(estimator=pipe,param_distributions= param_grid, cv=5, n_iter=100,
                             verbose=2, random_state=42,n_jobs=-1,scoring = 'roc_auc')
```

```
print(rforest.best_params_)
```

```
{'rfc__n_estimators': 1000, 'rfc__min_samples_split': 2, 'rfc__min_samples_leaf': 1, 'rfc__max_depth': 40}
```

XGBOOST:

- As suggested during the lectures, I employed a XG Boost Classifier on the one hot encoded data
- Used Randomized Search CV and further Grid Search CV with 3-fold cv on a set of parameters to tune the hyperparameters and train the model
- Compared to Random Forest, the cross-validation scores seemed to increase substantially using XG Boost
- Got a Kaggle score of 0.89599 and jumped to #2 position on the public leaderboard

```
param_grid = {
    'xgb__n_estimators': np.arange(10,400,30),
    'xgb__learning_rate': [0.1,0.2,0.3, 0.4, 0.5],
    'xgb__max_depth' : [2, 4,6, 8, 10, 12],
    'xgb__colsample_bytree': [0.2,0.3,0.5],
}
pipe = Pipeline([('xgb', xgb.XGBClassifier())])
xgb_model = RandomizedSearchCV(pipe, param_grid, cv=3, n_iter=100, verbose=2, n_jobs=-1,scoring = 'roc_auc')
```

```
print(xgb_model.best_params_)
```

```
{'xgb__n_estimators': 340, 'xgb__max_depth': 12, 'xgb__learning_rate': 0.2, 'xgb__colsample_bytree': 0.2}
```

```
from sklearn.model_selection import cross_val_score
# score the model using cross_val_score

xgb_cv = xgb.XGBClassifier(n_estimators= 340, max_depth= 12,
                           learning_rate = 0.2,colsample_bytree = 0.2, random_state = 42)

scores = cross_val_score(xgb_cv, X_train1, y, cv=10, scoring='roc_auc')
```

Running XG Boost and Random Forest Models without Dummy variables

- Since I had a couple of submissions left for the day, I ran Random Forest and XG Boost on the train dataset without dummies (24 features) with the same set of hyperparameters grid.
- I got a slightly better public leaderboard score of 0.88002 on the Random Forest as compared to my previous submission (of my random forest model), but my score on XG Boost went down to 0.88829.
- Inference: Using one hot encoded train dataset resulted in a better leaderboard score

Scaling and standardization of features using Robust Scaler and Power Transformer

- Used Scikit-Learn's Robust Scaler function to scale and standardize my train and test data and ran a XG Boost model; employed Randomized Search CV to tune the hyperparameters as usual.
- The score using XG Boost model went down to 0.7835 after scaling the dataset.
- I also tried applying Scikit Learn's Power Transformer function feature-wise on the numerical features (four features: columns f1, f8, f14 and f15) to make the data look more Gaussian-like

```
scaler = RobustScaler(copy = True)
train1_sc = scaler.fit_transform(train1)
train1_sc = pd.DataFrame(train1_sc,columns = train1.columns)

test1_sc = scaler.fit_transform(test1)
test1_sc = pd.DataFrame(test1_sc,columns = test1.columns)
```

```
train1_copy = train1.copy()

test1_copy = test1.copy()
train1_copy[['f1','f14','f15']] = pt.fit_transform(train1_copy[['f1','f14','f15']])
test1_copy[['f1','f14','f15']] = pt.fit_transform(test1_copy[['f1','f14','f15']])
```

- Ran a XG Boost Classifier model after feature scaling and transformation and one hot encoding the dataset. Although the cross-validation score was decent, my public leaderboard score went down to 0.80659 for this submission

Stratified K-Fold and Blending models

- As the train dataset was imbalanced and had lesser zeroes, I decided to use Stratified K-fold for cross validation split (n-splits = 5) and run my XG Boost and Random Forest models over the parameter grid again
- The score increased for both tuned XG Boost and Random Forest models. Public Leaderboard score increased to 0.90051 and then to 0.90299 by further tuning the XG Boost model with stratified K-fold.

```
param_grid = {
    'xgb__n_estimators': np.arange(10,400,30),
    'xgb__learning_rate': [0.1,0.2,0.3, 0.4, 0.5],
    'xgb__max_depth' : [2, 4,6, 8, 10, 12],
    'xgb__colsample_bytree': [0.2,0.3,0.5],
}
pipe = Pipeline([('xgb', xgb.XGBClassifier())])
xgb_model = RandomizedSearchCV(pipe, param_grid, cv=StratifiedKFold(n_splits=5), n_iter=200,
                               verbose=2, n_jobs=-1,scoring = 'roc_auc')
```

```
xgb_model.fit(X_train1, y)
```

```
Fitting 5 folds for each of 200 candidates, totalling 1000 fits
```

```
print(xgb_model.best_params_)
```

```
{'xgb__n_estimators': 190, 'xgb__max_depth': 12, 'xgb__learning_rate': 0.1, 'xgb__colsample_bytree': 0.2}
```

```
print(xgb_model.cv_results_['mean_test_score'].max())
```

```
0.8973912772068025
```

```
xgb_cv = xgb.XGBClassifier(n_estimators= 190, max_depth= 12,
                           learning_rate = 0.1,colsample_bytree = 0.2, random_state = 42)

scores = cross_val_score(xgb_cv, X_train1, y, cv=10, scoring='roc_auc')

print(scores)
print(scores.mean())
```

```
print(rforest.best_params_)
```

```
{'rfc__n_estimators': 1000, 'rfc__min_samples_split': 3, 'rfc__min_samples_leaf': 1, 'rfc__max_depth': 50}
```

```
print(rforest.cv_results_['mean_test_score'].max())
```

```
0.8669393826141137
```

```
# score the model using cross_val_score

rfc_cv = RandomForestClassifier(n_estimators= 1000,
                                    min_samples_split= 3, min_samples_leaf= 1, max_depth= 50,random_state = 42)

scores_rfc = cross_val_score(rfc_cv, X_train1, y, cv=10, scoring='roc_auc')


print(scores_rfc)
print(scores_rfc.mean())
```

```
[0.87197641 0.82659531 0.85500409 0.87638835 0.87907549 0.87977965
 0.84988232 0.86720674 0.88241293 0.91670362]
0.870502491662355
```

```
rfc_cv.fit(X_train1, y)
```

```
RandomForestClassifier(max_depth=50, min_samples_split=3, n_estimators=1000,
                       random_state=42)
```

```
test_preds = rfc_cv.predict_proba(X_test1)
rfc_preds = test_preds[:,1]
rfc_preds
```

```
array([0.8793    , 0.72200714, 0.98503333, ..., 0.99125   , 0.9629    ,
       0.95491667])
```

- Tried blending XG boost and Random Forest Classifier models. Using the train_test_split function, I split my train data further into training and validation set and trained my XG boost and Random Forest models on the training data and made predictions on the validation data.
- Checked my ROC-AUC score over a set of weights for blending my tuned XG Boost and Random Forest models by comparing my predictions against the y-labels of the validation set. Score increased to 0.90656 by blending in the ratio 0.9:0.1 for XG Boost and Random Forest respectively

```
X_tr,X_tst,Y_tr,Y_tst = train_test_split(X_train1,y,test_size = 0.3,random_state=42)
```

```
param_grid = {
    'xgb__n_estimators': np.arange(10,310,30),
    'xgb__learning_rate': [0.1,0.15,0.2,0.3, 0.4],
    'xgb__max_depth' : [2, 4, 8, 10, 12],
    'xgb__colsample_bytree': [0.2,0.3,0.4],
}
pipe = Pipeline([('xgb', xgb.XGBClassifier())])
blxgb_model = RandomizedSearchCV(pipe, param_grid, cv=StratifiedKFold(n_splits=3), n_iter=200,
                                 verbose=2, n_jobs=-1,scoring = 'roc_auc')
```

```
blxgb_model.fit(X_tr, Y_tr)
```

```
Fitting 3 folds for each of 200 candidates, totalling 600 fits
```

```
print(blxgb_model.best_params_)
print(blxgb_model.cv_results_['mean_test_score'].max())
```

```
{'xgb__n_estimators': 250, 'xgb__max_depth': 12, 'xgb__learning_rate': 0.1, 'xgb__colsample_bytree': 0.2}
0.8692771054657188
```

```
xgb_bl = xgb.XGBClassifier(n_estimators= 250, max_depth= 12,
                           learning_rate = 0.1,colsample_bytree = 0.2, random_state = 42)

scores = cross_val_score(xgb_bl, X_tr, Y_tr, cv=StratifiedKFold(n_splits=3), scoring='roc_auc')

print(scores)
print(scores.mean())
```

```
xgb_bl.fit(X_tr, Y_tr)
testbl_preds = xgb_bl.predict_proba(X_tst)
blxgb_preds = testbl_preds[:,1]
blxgb_preds
```

```
#Grid of parameters to train the Random Forest over

param_grid = {
    'rfc__n_estimators': [100,300,500,600,700,800,900,1000],
    'rfc__max_depth': [10,20,30,40,45,50,60,80,100],
    'rfc__min_samples_split': [2,3,4,6,8,10,20],
    'rfc__min_samples_leaf': [1, 2, 3,5,8]
}
pipe = Pipeline([('rfc', RandomForestClassifier())])

rforest_bl = RandomizedSearchCV(estimator=pipe,param_distributions= param_grid, cv=StratifiedKFold(n_splits=3),
                                n_iter=200, verbose=2, random_state=42,n_jobs=-1,scoring = 'roc_auc')
```

```
rforest_bl.fit(X_tr, Y_tr)
```

```
print(rforest_bl.best_params_)
print(rforest_bl.cv_results_['mean_test_score'].max())
```

```
{'rfc__n_estimators': 800, 'rfc__min_samples_split': 2, 'rfc__min_samples_leaf': 2, 'rfc__max_depth': 80}
0.8364248563053392
```

```
rfc_bl = RandomForestClassifier(n_estimators= 800,
                                min_samples_split= 2, min_samples_leaf= 2, max_depth= 80,random_state = 42)

scores_rfc = cross_val_score(rfc_bl, X_tr, Y_tr, cv=StratifiedKFold(n_splits=3), scoring='roc_auc')


print(scores_rfc)
print(scores_rfc.mean())
```

```
[0.83932038 0.83639882 0.82130834]
0.8323425107960721
```

```
rfc_bl.fit(X_tr, Y_tr)
testrfcbl_preds = rfc_bl.predict_proba(X_tst)
blrfc_preds = testrfcbl_preds[:,1]
blrfc_preds
```

```
array([0.83549538, 0.9008488 , 0.94119658, ..., 0.92817877, 0.93189322,
       0.96454858])
```

```
for i in np.linspace(0,1,21):
    bld_preds = (1-i)*blxgb_preds + i*blrfc_preds
    print(roc_auc_score(Y_tst,bld_preds))
```

```
0.8745576685404457
0.8744799266435435
0.8736100147510266
0.8728581110712436
0.8720671371048119
0.8713152334250289
0.8705641270980345
```

- However, my attempt at randomly blending two XG Boost models with and without Stratified K-fold got me a low score of 0.80659

Pseudo-Labelling with XG Boost

- I decided to try Pseudo-labelling of test data, which was another approach discussed in class, to increase the amount of data the model could use to train over
- Used XG Boost model's best parameters to initially predict my y-labels for test data. Then, I used the combination of my original train and test dataset along with the original and predicted Y -labels (using my best XGB model) to train my second XG Boost model

- Unfortunately, this approach did not work, and I got a score of 0.89989 on the public leaderboard

Knowledge Distillation with Blended XG Boost

- Since my score was not improving, I then decided to try Knowledge Distillation (label softening of train set) using my XG Boost model
- After fitting my XG Boost model on my train dataset using the best parameters, I predicted a new set of y-labels again on the train data and used the new y-labels to again train a new XG Boost model
- Used Random Search CV for a wide range of parameter values to narrow down on to my optimum hyperparameters
- Scored increased to 0.90604 on the public leaderboard
- I then tried running Grid Search CV on a narrow range of parameters and then blending with my Random Forest model predictions in the optimal ratio (0.9:0.1) obtained previously
- My score increased to 0.90844 on the public leaderboard
- Since I had two submissions left, I randomly tried blending in the ratio 0.95: 0.05 and 0.93:0.07 for XG Boost: Random Forest models and surprisingly, my score increased marginally to 0.90960 for 0.93:0.07 ratio

```
xgb_cv = xgb.XGBClassifier(n_estimators= 190, max_depth= 12,
                           learning_rate = 0.1,colsample_bytree = 0.2, random_state = 42)


xgb_cv.fit(X_train1, y)

train_preds = xgb_cv.predict_proba(X_train1)
#test_preds = xgb_cv.predict_proba(X_test1)

y_train = train_preds[:,1]

```

```
param_grid = {
    'xgb__n_estimators': np.arange(10,310,30),
    'xgb__learning_rate': [0.1,0.2,0.3, 0.4, 0.5],
    'xgb__max_depth' : [2,5, 8, 10, 12],
    'xgb__colsample_bytree': [0.2,0.3,0.5],
}
pipe = Pipeline([('xgb', xgb.XGBClassifier())])
xgb_model = RandomizedSearchCV(pipe, param_grid, cv=StratifiedKFold(n_splits=3), n_iter=200,
                               verbose=2, n_jobs=-1,scoring = 'roc_auc')
xgb_model.fit(X_train1, y_train)
```

```
Fitting 3 folds for each of 200 candidates, totalling 600 fits
```

```
param_grid = {
    'xgb__n_estimators': np.arange(220,260,10),
    'xgb__learning_rate': [0.08,0.1],
    'xgb__max_depth' : [14,16,18,20],
    'colsample_bylevel': np.arange(0.1,0.5,0.2),
    'gamma': np.arange(0.1,0.55,0.1)
}
#pipe = Pipeline([('xgb', xgb.XGBClassifier())])
xgb_model = GridSearchCV(xgb.XGBClassifier(), param_grid, cv=StratifiedKFold(n_splits=3),
                               verbose=2, n_jobs=-1,scoring = 'roc_auc')
xgb_model.fit(X_train1, y_train)
```

```
Fitting 3 folds for each of 320 candidates, totalling 960 fits
```

```
print(xgb_model.best_params_)
print(xgb_model.cv_results_['mean_test_score'].max())
```

```
{'colsample_bylevel': 0.30000000000000004, 'gamma': 0.1, 'xgb__learning_rate': 0.08, 'xgb__max_depth': 14, 'xgb__n_es
timators': 220}
0.884701541606442
```

```
xgb_cv.fit(X_train1, y_train)

test_preds = xgb_cv.predict_proba(X_test1)

xgb_predictions = test_preds[:,1]
xgb_predictions
```

```
blendxgbrfc6 = 0.93*xgb_predictions + 0.07*rfc_preds
output = pd.DataFrame({'Id': (test['Id'].values),
                       'Y': blendxgbrfc6})
output.to_csv('SDistGridSblendxgbrfc11.csv', index=False)
```

Oversampling with SMOTE

- Since the train dataset was unbalanced, I tried oversampling with SMOTE (from imblearn.over_sampling) to create synthetic samples and thereby get more 0's in my Y-label
- Initially, I tried with sampling strategy = 1 so that there are equal number of 0's and 1's in my train dataset and trained my XG B model over the set of parameters and then blended the predictions with random forest
- This strategy did not yield good results and my leaderboard score dropped to 0.89039
- I realized that there's a possibility that I might be creating too many synthetic samples just to balance my dataset, so I lowered my sampling strategy to 0.25
- I used this sampling strategy and trained my XG B model with and without softening my y-labels of the train data (along with blending with random forest predictions in both cases)
- Neither of these two approaches seemed to work as my highest score using SMOTE was 0.90011 which was lower than my best score

```
import imblearn
import matplotlib.pyplot
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline

```

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(sampling_strategy=0.25)
X_sm, y_sm = smote.fit_resample(X_train1, y)

```

```
param_grid = {
    'xgb__n_estimators': np.arange(10,400,30),
    'xgb__learning_rate': [0.1,0.2,0.3, 0.4, 0.5],
    'xgb__max_depth' : [4, 8, 10, 12,14,16,18,20],
    'xgb__colsample_bytree': [0.2,0.3,0.4,0.5],
}
pipe = Pipeline([('xgb', xgb.XGBClassifier())])
smxgb_model = RandomizedSearchCV(pipe, param_grid, cv=StratifiedKFold(n_splits=3), n_iter=200,
                                 verbose=2, n_jobs=-1,scoring = 'roc_auc')
smxgb_model.fit(X_sm, y_sm)
```

```
Fitting 3 folds for each of 200 candidates, totalling 600 fits
```

Gradient Boosting Classifier and Decision Tree Classifier

- As my score stagnated, I hopelessly tried out other Gradient Boosting classifier and Decision tree classifier. The idea was to blend these models with my tuned XG Boost and Random Forest predictions and possibly get a better score
- I tried blending these models with XG B and Random Forest in different proportions and my score crossed the 0.91's and marginally increased to 0.91048 with 93% weight to XG Boost model and 2 % to Random Forest and 5 % to Decision Tree classifier

```
param_grid = {
    'min_samples_split': [2,4,6,8],
    'max_depth' : [2, 4,8, 10, 12,16,18,20,None],
                }
#pipe = Pipeline([('dtc', DecisionTreeClassifier())])
dtc_model = RandomizedSearchCV(DecisionTreeClassifier(), param_grid, cv=StratifiedKFold(n_splits=3), n_iter=200,
                               verbose=2, n_jobs=-1,scoring = 'roc_auc')
dtc_model.fit(X_train1, y)
```

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits
```

```
print(dtc_model.best_params_)
print(dtc_model.cv_results_['mean_test_score'].max())
```

```
{'min_samples_split': 4, 'max_depth': 4}
0.7768339463573254
```

```
dtc_cv = DecisionTreeClassifier(criterion='gini',min_samples_split = 4, max_depth= 4, random_state = 42)

scores = cross_val_score(dtc_cv, X_train1, y, cv=10, scoring='roc_auc')

print(scores)
print(scores.mean())
```

```
[0.77291042 0.74733092 0.76500205 0.76234015 0.77740533 0.77029369
 0.76519426 0.79955998 0.78089845 0.79871406]
0.7739649302990792
```

```
dtc_cv.fit(X_train1, y)

test_preds = dtc_cv.predict_proba(X_test1)

dtc_predictions = test_preds[:,1]
```

```
param_grid = {
    'n_estimators': np.arange(10,400,30),
    'learning_rate': [0.1,0.2,0.3, 0.4],
    'max_depth' : [2, 4,8, 10, 12,16,18,20],
    }

gbc_model = RandomizedSearchCV(GradientBoostingClassifier(), param_grid, cv=StratifiedKFold(n_splits=3),
                               n_iter=200, verbose=2, n_jobs=-1,scoring = 'roc_auc')
gbc_model.fit(X_train1, y)
```

```
Fitting 3 folds for each of 200 candidates, totalling 600 fits
```

```
print(gbc_model.best_params_)
print(gbc_model.cv_results_['mean_test_score'].max())
```

```
{'n_estimators': 370, 'max_depth': 4, 'learning_rate': 0.1}
0.8638719743473038
```

```
gbc_cv = GradientBoostingClassifier(n_estimators = 370, max_depth= 4, learning_rate = 0.1, random_state = 42)

scores = cross_val_score(gbc_cv, X_train1, y, cv=10, scoring='roc_auc')

print(scores)
print(scores.mean())
```

```
[0.89121898 0.82688165 0.83443551 0.87934076 0.88776045 0.89017294
 0.8347921  0.86086571 0.88792168 0.86059283]
0.8653982595139487
```

I also tried soft labeling the y-labels as above on the original train set (without one hot encoding the data) using the 24 features and then blending several tuned models (XG B, Random Forests etc.) but that didn't increase my score either

Checking for synthetic samples on the train and test data

- While going through the Santander Customer Transaction Prediction, I found a notebook about removing fake/synthetic samples from the train and test data. https://www.kaggle.com/yag320/list-of-fake-samples-and-public-private-lb-split
- I thought of giving this approach a try as well. The rationale behind this approach was to remove synthetic samples as there was a possibility that these samples could be reducing my score (as observed when I tried creating synthetic samples and oversampling using SMOTE)

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from tqdm import tqdm_notebook as tqdm
```

```
train_path = '/Users/angshumanchakraborty/Desktop/dslab-kaggle-competition-fall-21/Raw Data/train_final.csv'
```

```
df_train = pd.read_csv(train_path)
```

```
df_train.drop(['Id'], axis=1, inplace=True)
df_train = df_train.values

unique_samples = []
unique_count = np.zeros_like(df_train)
for feature in tqdm(range(df_train.shape[1])):
    _, index_, count_ = np.unique(df_train[:, feature], return_counts=True, return_index=True)
    unique_count[index_[count_ == 1], feature] += 1
```

```
# Samples which have unique values are real the others are fake
real_samples_indexes = np.argwhere(np.sum(unique_count, axis=1) > 0)[:, 0]
synthetic_samples_indexes = np.argwhere(np.sum(unique_count, axis=1) == 0)[:, 0]

print(len(real_samples_indexes))
print(len(synthetic_samples_indexes))
```

```
12876
3507
```

```
df_train_syn = df_train[synthetic_samples_indexes].copy()
df_train_syn = pd.DataFrame(df_train_syn,columns = trainz.columns)
```

```
train2 = train[columns]
y = train['Y']
```

```
df_train_real = df_train[real_samples_indexes].copy()
df_train_real[:,0]
```

```
array([1., 1., 1., ..., 1., 1., 1.])
```

```
df_train_real = df_train[real_samples_indexes].copy()
df_train_real = pd.DataFrame(df_train_real,columns = train2.columns)
```

```
train2 = df_train_real[cols]
y = df_train_real['Y']
```

```
test2 = test[cols]
```

```
for col in cols:
    train2[col] = train2[col].astype(int)
```

```
train_objs_num = len(train2)
train_objs_num
```

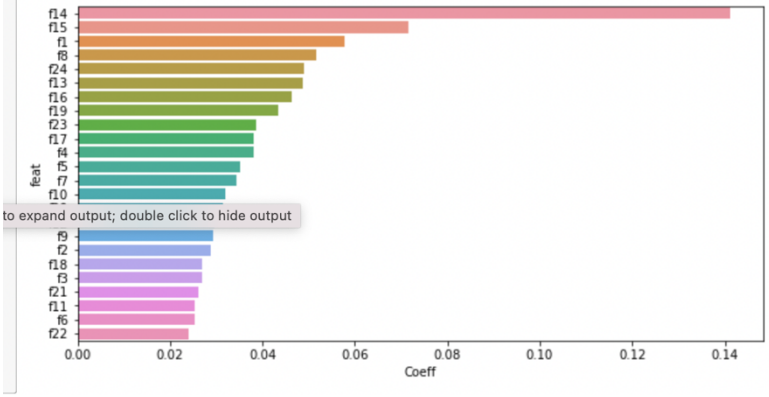
```
12876
```

- I dropped those specific samples which were identified as synthetic and trained my models on the new train data along with hyperparameter tuning.
- However, training the models after soft labelling them and then blending them in the same proportion as previously done reduced my score to 0.905

At this point, I wanted to be as exhaustive as possible, so I decided to one hot encode any feature which had less than 200 unique values in the combined dataset and had a higher feature importance for XG Boost since it was my primary model. As a result, I trained my models on ~ 469 columns after one hot encoding them and repeated the process of soft labelling and then blending various models as done above. There was no substantial improvement to my score using this approach and it marginally increased to 0.91066.

```
plt.figure(figsize=(10,5))
xgb_eval = pd.DataFrame()
xgb_eval['Coeff'] = list(xgb_cv.feature_importances_)
xgb_eval['feat'] = list(train1.columns)
xgb_eval.sort_values(['Coeff'],ascending=False, inplace=True)
sns.barplot(y = xgb_eval['feat'], x = xgb_eval['Coeff'])
```

```
<AxesSubplot:xlabel='Coeff', ylabel='feat'>
```



CatBoost

- I briefly tried out CatBoost since I wanted to try out any remaining alternative to improve my score and it is generally ideal from smaller datasets.
- For hyperparameter tuning, I did a Randomized Search CV on four parameters (learning rate, depth, iterations)
- As I had just two submissions left, I used two random weights for blending my CatBoost Model predictions with my prediction for the best score so far
- Blended CatBoost with my previous best model predictions in the ratio 20:80 (i.e. Final proportions are in the order of 20 % CatBoost, 74% XG Boost, 2 % Random Forest and 4 % Decision Tree classifier
- I got my best score in my last submission and my public leaderboard score increased to 0.91188

```
CatBoostClassifier?
```

```
from catboost import CatBoostClassifier
param_grid = {'learning_rate': [0.03, 0.05,0.08,0.1],
        'depth': [4, 6, 8,10,12],
        'l2_leaf_reg': [1, 3, 5, 7, 9],
        'iterations': np.arange(10,400,30)}

#pipe = Pipeline([('cgbm', CatBoostClassifier())])
cgbm_model = RandomizedSearchCV(CatBoostClassifier(), param_grid, cv=StratifiedKFold(n_splits=3), n_iter=200,
                                verbose=2, n_jobs=-1,scoring = 'roc_auc')
cgbm_model.fit(X_train2, y)
```

```
Fitting 3 folds for each of 200 candidates, totalling 600 fits
```

```
print(cgbm_model.best_params_)
print(cgbm_model.cv_results_['mean_test_score'].max())
```

```
{'learning_rate': 0.1, 'l2_leaf_reg': 9, 'iterations': 370, 'depth': 10}
0.8605846075621327
```

```
cgbm_cv = CatBoostClassifier(iterations= 370, depth= 10,
                             learning_rate = 0.1,l2_leaf_reg = 9, random_state = 42)

```

```
cgbm_cv.fit(X_train2, y)

test_preds = cgbm_cv.predict_proba(X_test2)

cgbm_predictions = test_preds[:,1]
cgbm_predictions
```

```
Custom logger is already specified. Specify more than one logger at same time is not thread safe.

0:      learn: 0.5704142        total: 23.2ms   remaining: 8.55s
1:      learn: 0.4854414        total: 43.6ms   remaining: 8.03s
2:      learn: 0.4217674        total: 61.4ms   remaining: 7.51s
3:      learn: 0.3734755        total: 79.5ms   remaining: 7.28s
4:      learn: 0.3365210        total: 102ms    remaining: 7.48s
```

```
blendxgbrfc12 = 0.8*blend + 0.2*cgbm_predictions
output = pd.DataFrame({'Id': (test['Id'].values),
                       'Y': blendxgbrfc12})
output.to_csv('DummySDistGridSblendxgbrfcdtc3.5.csv', index=False)
```

What Could Have Gone Better: Key Takeaways

I believe I should have spent a little bit more time on EDA and feature engineering in the first couple of days itself. From the class discussions, it was apparent that people who had better scores did a little bit of more feature engineering (splitting a feature, removing noise from features, dropping features etc.). I believe that I spent a bit more time than required on model tuning and by investing some time on feature engineering, I could have pushed my score beyond 0.92 on the public leaderboard.