Amazon Employee Access Challenge (Part-2)

In [ ]:

```
#Importing libraries
import numpy as np
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings("ignore")

from scipy import sparse
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import sklearn.metrics as metrics
```

In [ ]:

```
!pip install category_encoders
```

```
Requirement already satisfied: category_encoders in /usr/local/lib/python3.
6/dist-packages (2.2.2)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.6/dist
-packages (from category_encoders) (0.5.1)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.
6/dist-packages (from category_encoders) (0.10.2)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.6/dis
t-packages (from category_encoders) (1.18.5)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.6/dist
-packages (from category_encoders) (1.4.1)
Requirement already satisfied: pandas>=0.21.1 in /usr/local/lib/python3.6/di
st-packages (from category_encoders) (1.0.5)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python
3.6/dist-packages (from category_encoders) (0.22.2.post1)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages
(from patsy>=0.5.1->category_encoders) (1.12.0)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist
-packages (from pandas>=0.21.1->category_encoders) (2018.9)
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/local/lib/pyth
on3.6/dist-packages (from pandas>=0.21.1->category_encoders) (2.8.1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist
-packages (from scikit-learn>=0.20.0->category_encoders) (0.15.1)
```

In [ ]:

```
#Reading data
data = pd.read_csv('train.csv')
data_test = pd.read_csv('test.csv')
```

#Feature Engineering

In [ ]:

```
Y = data['ACTION']
X = data.drop('ACTION', axis = 1)

#Dropping ROLE_CODE feature.
X = X.drop('ROLE_CODE', axis = 1)

X_test = data_test.drop('ROLE_CODE', axis = 1)
X_test = X_test.drop('id', axis = 1)
```

In [ ]:

```
X_test.head()
```

Out[33]:

	RESOURCE	MGR_ID	ROLE_ROLLUP_1	ROLE_ROLLUP_2	ROLE_DEPTNAME	ROLE_TITLE
0	78766	72734	118079	118080	117878	117879
1	40644	4378	117961	118327	118507	118863
2	75443	2395	117961	118300	119488	118172
3	43219	19986	117961	118225	118403	120773
4	42093	50015	117961	118343	119598	118422

In [ ]:

```
X_test.columns
```

Out[34]:

```
Index(['RESOURCE', 'MGR_ID', 'ROLE_ROLLUP_1', 'ROLE_ROLLUP_2', 'ROLE_DEPTNAM
E',
       'ROLE_TITLE', 'ROLE_FAMILY_DESC', 'ROLE_FAMILY'],
      dtype='object')
```

##Hybrid Features

Since these features are already encoded randomly and they are nominal features. So, adding, subtracting the feature values won't help much. However we can try concatenating them with each other.

In [ ]:

```
n = len(X.columns)
print(f"We can {n} no. of features.")
```

```
We can 8 no. of features.
```

In [ ]:

```
from tqdm import tqdm
from itertools import combinations

def concat_features_duplet(df_train, cols):
    dup_features = []
    for indicies in combinations(range(len(cols)), 2):
        dup_features.append([hash(tuple(v)) for v in df_train[:,list(indicies)]])
    return np.array(dup_features).T
```

In [ ]:

```
def concat_features_triplet(df_train, cols):
    tri_features = []
    for indicies in combinations(range(len(cols)), 3):
        tri_features.append([hash(tuple(v)) for v in df_train[:,list(indicies)]])
    return np.array(tri_features).T
```

Let's add feature category frequencies as well.

In [ ]:

```
from collections import Counter

def category_freq(X):
    X_new = X.copy()
    for f in X_new.columns:
        col_count = dict(Counter(X_new[f].values))

        for r in X_new.itertuples():
            X_new.at[r[0], f'{f}_freq'] = col_count[X_new.loc[r[0], f]]
    return X_new
```

In [ ]:

```
X.nunique()
```

Out[39]:

```
RESOURCE           7518
MGR_ID             4243
ROLE_ROLLUP_1       128
ROLE_ROLLUP_2       177
ROLE_DEPTNAME       449
ROLE_TITLE          343
ROLE_FAMILY_DESC   2358
ROLE_FAMILY          67
dtype: int64
```

In [ ]:

```
X_dup_train = concat_features_duplet(np.array(X), ['RESOURCE', 'MGR_ID', 'ROLE_ROLLUP_1', '
       'ROLE_TITLE', 'ROLE_FAMILY_DESC', 'ROLE_FAMILY'])

X_tri_train = concat_features_triplet(np.array(X), ['RESOURCE', 'MGR_ID', 'ROLE_ROLLUP_1',
       'ROLE_TITLE', 'ROLE_FAMILY_DESC', 'ROLE_FAMILY'])
```

In [ ]:

```
X_dup_test = concat_features_duplet(np.array(X_test), ['RESOURCE', 'MGR_ID', 'ROLE_ROLLUP_1
       'ROLE_TITLE', 'ROLE_FAMILY_DESC', 'ROLE_FAMILY'])

X_tri_test = concat_features_triplet(np.array(X_test), ['RESOURCE', 'MGR_ID', 'ROLE_ROLLUP_
       'ROLE_TITLE', 'ROLE_FAMILY_DESC', 'ROLE_FAMILY'])
```

In [ ]:

```
from category_encoders import OrdinalEncoder

X_dup_all = np.vstack((X_dup_train, X_dup_test))
X_tri_all = np.vstack((X_tri_train, X_tri_test))

enc = OrdinalEncoder().fit(X_dup_all)
X_dup_train = enc.transform(X_dup_train)
X_dup_test = enc.transform(X_dup_test)

enc1 = OrdinalEncoder().fit(X_tri_all)
X_tri_train = enc1.transform(X_tri_train)
X_tri_test = enc1.transform(X_tri_test)

import pickle
#Saving pickle file for one-hot encoding
with open('lab_dup.pickle', 'wb') as f:
    pickle.dump(enc, f)

with open('lab_tri.pickle', 'wb') as g:
    pickle.dump(enc1, g)
```

In [ ]:

```
X_dup_train.shape
```

Out[50]:

```
(32769, 28)
```

In [ ]:

```
X_freq_train = np.array(category_freq(X).iloc[:,8:])
X_freq_test = np.array(category_freq(X_test).iloc[:,8:])
```

In [ ]:

```
X_freq_train.shape
```

Out[52]:

```
(32769, 8)
```

Combining all the original features + hybrid categorical features

In [ ]:

```
X_train_all_categorical = np.hstack((X, X_dup_train, X_tri_train))
X_test_all_categorical = np.hstack((X_test, X_dup_test, X_tri_test))
```

In [ ]:

```
X_train_all_categorical.shape
```

Out[54]:

```
(32769, 92)
```

###One-Hot Encoding all categorical variables

We are using One-Hot Encoding only here since it worked best with our base model.

In [ ]:

```
from sklearn.preprocessing import OneHotEncoder

one_hot = OneHotEncoder()
one_hot.fit(np.vstack((X_train_all_categorical, X_test_all_categorical)))
X_train_cat_one_enc = one_hot.transform(X_train_all_categorical)
X_test_cat_one_enc = one_hot.transform(X_test_all_categorical)
```

In [ ]:

```
X_train_cat_one_enc
```

Out[56]:

```
<32769x1595082 sparse matrix of type '<class 'numpy.float64'>'
        with 3014748 stored elements in Compressed Sparse Row format>
```

###Standard Scaler for Numerical Features

In [ ]:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(np.vstack((X_freq_train, X_freq_test)))

X_train_freq = scaler.transform(X_freq_train)
X_test_freq = scaler.transform(X_freq_test)
```

In [ ]:

```
X_train_freq.shape
```

Out[58]:

```
(32769, 8)
```

In [ ]:

```
#Saving pickle file for Standard Scaler encoding
with open('scaler.pickle', 'wb') as f:
    pickle.dump(scaler, f)
```

###Thats all. We will have a lots of options to try with all these different encodings and hybrid features. Let's jump into the interesting part i.e. Modelling!

#Modelling

#Base Model

We had a base model AUC: 0.8497 with one-hot encoding using only original features.

#Advanced Models

##1. Logistic Regression

Let's try Logistic Regression on hybrid features this time.

We will be relying on our 10-fold Cross Validation technique to compare AUC Score.

In [ ]:

```
def cv_loop(X, Y, model):

  mean_auc = []
  seed = 42

  for i in range(10):

      X_train, X_cv, y_train, y_cv = train_test_split(X, Y, test_size = .20, random_state =

      # train model and make predictions
      model.fit(X_train, y_train)
      preds = model.predict_proba(X_cv)[:, 1]

      # compute AUC metric for this CV fold
      fpr, tpr, thresholds = metrics.roc_curve(y_cv, preds)
      roc_auc = metrics.auc(fpr, tpr)
      mean_auc.append(roc_auc)

  return np.mean(mean_auc)
```

In [ ]:

```
X_train_freq.shape
```

Out[61]:

```
(32769, 8)
```

In [ ]:

```
#Combining all the one-hot encoded features and numerical features together.
X_train_all = sparse.hstack((X_train_freq, X_train_cat_one_enc))

for c in [ 0.01, 0.1, 1, 10]:
    auc = cv_loop(X_train_all, Y, LogisticRegression(C = c, class_weight = 'balanced'))
    print(f"For C: {c}, AUC: {auc}")
```

```
For C: 0.01, AUC: 0.8767942059329366
For C: 0.1, AUC: 0.8825108905814363
For C: 1, AUC: 0.8822030700864196
For C: 10, AUC: 0.8803338425218719
```

Thats a Big Improvement from our Base Model.

Best AUC: 0.8825

###Forward Feature Selection

In [ ]:

```
#One-Hot Encoding all the features seperately

X_train_one_enc_featurewise = []
X_test_one_enc_featurewise = []

for k in range(X_train_all_categorical.shape[1]):
    one_hot = OneHotEncoder()
    one_hot.fit(np.vstack((X_train_all_categorical[:, [k]], X_test_all_categorical[:, [k]])))
    X_train_one_enc_featurewise.append(one_hot.transform(X_train_all_categorical[:, [k]]))
    X_test_one_enc_featurewise.append(one_hot.transform(X_test_all_categorical[:, [k]]))

for m in range(X_train_freq.shape[1]):
    X_train_one_enc_featurewise.append(sparse.csr_matrix(X_train_freq[:, m].reshape(-1, 1)))
    X_test_one_enc_featurewise.append(sparse.csr_matrix(X_test_freq[:, m].reshape(-1, 1)))
```

In [ ]:

```
len(X_train_one_enc_featurewise)
```

Out[63]:

```
100
```

In [ ]:

```
#Forward Feature Selection

selected_features = []
best_auc = [0, 0.5]

while best_auc[-1] > best_auc[-2]:
    auc_scores = []

    for idx, f in enumerate(X_train_one_enc_featurewise):

        if idx not in selected_features:
            feats = selected_features + [idx]
            ft = sparse.hstack([X_train_one_enc_featurewise[j] for j in feats]).tocsr()
            auc = cv_loop(ft, Y, LogisticRegression(C = 0.1, class_weight = 'balanced'))
            auc_scores.append((idx, auc))

    best_auc.append(max(auc_scores,key=lambda x:x[1])[1])
    selected_features.append(max(auc_scores,key=lambda x:x[1])[0])

    print(f"Feature Set: {selected_features}, AUC: {best_auc[-1]}")
```

```
Feature Set: [64], AUC: 0.8378085735955476
Feature Set: [64, 42], AUC: 0.866011900056616
Feature Set: [64, 42, 69], AUC: 0.8700898379498575
Feature Set: [64, 42, 69, 11], AUC: 0.8756824372683282
Feature Set: [64, 42, 69, 11, 85], AUC: 0.8789372629869783
Feature Set: [64, 42, 69, 11, 85, 0], AUC: 0.8827222928439428
Feature Set: [64, 42, 69, 11, 85, 0, 65], AUC: 0.8856761988185402
Feature Set: [64, 42, 69, 11, 85, 0, 65, 93], AUC: 0.8865594522128337
Feature Set: [64, 42, 69, 11, 85, 0, 65, 93, 67], AUC: 0.8873691901765779
Feature Set: [64, 42, 69, 11, 85, 0, 65, 93, 67, 29], AUC: 0.888485894053176
4
Feature Set: [64, 42, 69, 11, 85, 0, 65, 93, 67, 29, 9], AUC: 0.889430762821
403
Feature Set: [64, 42, 69, 11, 85, 0, 65, 93, 67, 29, 9, 66], AUC: 0.89006887
93223163
Feature Set: [64, 42, 69, 11, 85, 0, 65, 93, 67, 29, 9, 66, 47], AUC: 0.8904
549795434397
Feature Set: [64, 42, 69, 11, 85, 0, 65, 93, 67, 29, 9, 66, 47, 60], AUC: 0.
8908003589245084
Feature Set: [64, 42, 69, 11, 85, 0, 65, 93, 67, 29, 9, 66, 47, 60, 10], AU
C: 0.8911855618343605
Feature Set: [64, 42, 69, 11, 85, 0, 65, 93, 67, 29, 9, 66, 47, 60, 10, 12],
AUC: 0.8914723596081741
Feature Set: [64, 42, 69, 11, 85, 0, 65, 93, 67, 29, 9, 66, 47, 60, 10, 12,
97], AUC: 0.8917010030658135
Feature Set: [64, 42, 69, 11, 85, 0, 65, 93, 67, 29, 9, 66, 47, 60, 10, 12,
97, 71], AUC: 0.8919246044886927
Feature Set: [64, 42, 69, 11, 85, 0, 65, 93, 67, 29, 9, 66, 47, 60, 10, 12,
97, 71, 8], AUC: 0.8921800818262421
Feature Set: [64, 42, 69, 11, 85, 0, 65, 93, 67, 29, 9, 66, 47, 60, 10, 12,
97, 71, 8, 53], AUC: 0.8923069872774082
Feature Set: [64, 42, 69, 11, 85, 0, 65, 93, 67, 29, 9, 66, 47, 60, 10, 12,
97, 71, 8, 53, 79], AUC: 0.8924385074722527
Feature Set: [64, 42, 69, 11, 85, 0, 65, 93, 67, 29, 9, 66, 47, 60, 10, 12,
97, 71, 8, 53, 79, 19], AUC: 0.8925806672150234
Feature Set: [64, 42, 69, 11, 85, 0, 65, 93, 67, 29, 9, 66, 47, 60, 10, 12,
97, 71, 8, 53, 79, 19, 36], AUC: 0.8927166739662228
Feature Set: [64, 42, 69, 11, 85, 0, 65, 93, 67, 29, 9, 66, 47, 60, 10, 12,
```

```
97, 71, 8, 53, 79, 19, 36, 63], AUC: 0.8927800839626974
Feature Set: [64, 42, 69, 11, 85, 0, 65, 93, 67, 29, 9, 66, 47, 60, 10, 12,
97, 71, 8, 53, 79, 19, 36, 63, 37], AUC: 0.892871225150353
```

Best Set of Features: [64, 42, 69, 11, 85, 0, 65, 93, 67, 29, 9, 66, 47, 60, 10, 12, 97, 71, 8, 53, 79, 19, 36, 63, 37]

AUC: 0.8928. Much improvement again.

Let's try to improve it further with hyperparameter tuning.

In [ ]:

```
#Saving one-hot encoder for pipeline
X_test_all_categorical_selected= X_test_all_categorical[:, [64, 42, 69, 11, 85, 0, 65, 67,
X_train_all_categorical_selected= X_train_all_categorical[:, [64, 42, 69, 11, 85, 0, 65, 67

one_hot = OneHotEncoder()
one_hot.fit(np.vstack((X_train_all_categorical_selected, X_test_all_categorical_selected)))

with open('one_hot.pickle', 'wb') as f:
    pickle.dump(one_hot, f)
```

###Hyper Parameter Tuning

In [ ]:

```
feats = [64, 42, 69, 11, 85, 0, 65, 93, 67, 29, 9, 66, 47, 60, 10, 12, 97, 71, 8, 53, 79, 1
fts = sparse.hstack([X_train_one_enc_featurewise[j] for j in feats]).tocsr()
```

In [ ]:

```
import warnings
warnings.filterwarnings("ignore")

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import sklearn.metrics as metrics

seed = 42

for n in [0.01, 0.1, 0.5, 1, 10]:

    mean_auc = []

    for i in range(10):

        logreg = LogisticRegression(C = n, class_weight = 'balanced')
        X_train, X_cv, y_train, y_cv = train_test_split(fts, Y, test_size = .20, random_sta

        # train model and make predictions
        logreg.fit(X_train, y_train)
        preds = logreg.predict_proba(X_cv)[:, 1]

        # compute AUC metric for this CV fold
        fpr, tpr, thresholds = metrics.roc_curve(y_cv, preds)
        roc_auc = metrics.auc(fpr, tpr)
        mean_auc.append(roc_auc)

    print(f"For C: {n}, AUC = {np.mean(mean_auc)}")
```

```
For C: 0.01, AUC = 0.8752494418510249
For C: 0.1, AUC = 0.8929433946139961
For C: 0.5, AUC = 0.8943984062042667
For C: 1, AUC = 0.894223473235555
For C: 10, AUC = 0.8925096941664086
```

###For C = 0.5 we get best AUC Score of 0.8943.

Let's make a submission to see our standing on the leaderboard.

In [ ]:

```
fts_test = sparse.hstack([X_test_one_enc_featurewise[j] for j in feats]).tocsr()
```

In [ ]:

```
fts.shape
```

Out[68]:

```
(32769, 583240)
```

In [ ]:

```
fts_test.shape
```

Out[69]:

```
(58921, 583240)
```

In [ ]:

```
model = LogisticRegression(C = 0.5)

# train model and make predictions
model.fit(fts, Y)
preds = model.predict_proba(fts_test)[:, 1]
```

In [ ]:

```
submission = pd.DataFrame(preds, columns = ['Action'])

submission['Id'] = range(1, len(preds)+1)
```

In [ ]:

```
submission.to_csv('logreg_with_FFSelection_updated.csv', index = False)
```

In [ ]:

```
#Saving model weights
filename = 'logreg1.sav'
pickle.dump(model, open(filename, 'wb'))
```

###AUC on Public LeaderBoard: 0.90520 and on Private LeaderBoard: 0.89930 Which is Top 15%.

##2. Random Forest

In [ ]:

```
#Our selected Train and Test Features
print(fts.shape)
print(fts_test.shape)
print(Y.shape)
```

```
(32769, 583240)
(58921, 583240)
(32769,)
```

In [ ]:

```
#Random Forest works better with lesser number of features and numerical features.
#Here we are using Stratified Cross Validation as our Cross Validation Technique
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold

kf = StratifiedKFold(n_splits=5,shuffle=True,random_state=42)

mean_auc = []

for train_index,test_index in kf.split(X_freq_train, Y):

    X_train = X_freq_train[train_index, :]
    y_train = Y.iloc[train_index]
    X_cv = X_freq_train[test_index, :]
    y_cv = Y.iloc[test_index]

    rf = RandomForestClassifier(n_jobs = -1)
    rf.fit(X_train, y_train)
    preds = rf.predict_proba(X_cv)[:, 1]

    # compute AUC metric for this CV fold
    fpr, tpr, thresholds = metrics.roc_curve(y_cv, preds)
    roc_auc = metrics.auc(fpr, tpr)
    mean_auc.append(roc_auc)

print(f"AUC Score from Random Forest: {np.mean(mean_auc)}")
```

```
AUC Score from Random Forest: 0.848450938715216
```

###Hyper Parameter Tuning

In [ ]:

```
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
```

In [ ]:

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 20 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter =
# Fit the random search model
rf_random.fit(X_freq_train, Y)
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:  9.8min
[Parallel(n_jobs=-1)]: Done  60 out of  60 | elapsed: 18.1min finished
```

Out[78]:

```
RandomizedSearchCV(cv=3, error_score=nan,
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    max_samples=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=100,
                                                    n_jobs...
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [10, 20, 30, 40, 50, 6
0,
                                                      70, 80, 90, 100, 110,
                                                      None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [200, 400, 600, 800,
                                                         1000, 1200, 1400, 1
600,
                                                         1800, 2000]},
                   pre_dispatch='2*n_jobs', random_state=42, refit=True,
                   return_train_score=False, scoring='roc_auc', verbose=2)
```

In [ ]:

```
rf_random.best_score_
```

Out[79]:

```
0.8575184873311453
```

In [ ]:

```
rf_random.best_params_
```

Out[80]:

```
{'bootstrap': False,
 'max_depth': 20,
 'max_features': 'auto',
 'min_samples_leaf': 2,
 'min_samples_split': 10,
 'n_estimators': 400}
```

##3. CatBoost Classifier

In [ ]:

```
!pip install catboost
```

```
Collecting catboost
  Downloading https://files.pythonhosted.org/packages/b2/aa/e61819d04ef2bbee
778bf4b3a748db1f3ad23512377e43ecfdc3211437a0/catboost-0.23.2-cp36-none-manyl
inux1_x86_64.whl (https://files.pythonhosted.org/packages/b2/aa/e61819d04ef2
bbee778bf4b3a748db1f3ad23512377e43ecfdc3211437a0/catboost-0.23.2-cp36-none-m
anylinux1_x86_64.whl) (64.8MB)
     |████████████████████████████████| 64.8MB 76kB/s
Requirement already satisfied: plotly in /usr/local/lib/python3.6/dist-packa
ges (from catboost) (4.4.1)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.6/dis
t-packages (from catboost) (1.18.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-p
ackages (from catboost) (3.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packag
es (from catboost) (1.4.1)
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.6/di
st-packages (from catboost) (1.0.5)
Requirement already satisfied: graphviz in /usr/local/lib/python3.6/dist-pac
kages (from catboost) (0.10.1)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages
(from catboost) (1.12.0)
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.6/d
ist-packages (from plotly->catboost) (1.3.3)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python
3.6/dist-packages (from matplotlib->catboost) (2.8.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.
6/dist-packages (from matplotlib->catboost) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist
-packages (from matplotlib->catboost) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /
usr/local/lib/python3.6/dist-packages (from matplotlib->catboost) (2.4.7)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist
-packages (from pandas>=0.24.0->catboost) (2018.9)
Installing collected packages: catboost
Successfully installed catboost-0.23.2
```

In [ ]:

```
from catboost import CatBoostClassifier

X_train_all = np.hstack((X_train_all_categorical, X_train_freq))
X_train_all = X_train_all.astype('int32')
X_train_all = pd.DataFrame(data=X_train_all, columns=list(range(100)))
X_train_all = X_train_all[feats]

X_test_all = np.hstack((X_test_all_categorical, X_test_freq))
X_test_all = X_test_all.astype('int32')
X_test_all = pd.DataFrame(data=X_test_all, columns=list(range(100)))
X_test_all = X_test_all[feats]

X_train, X_cv, Y_train, Y_cv = train_test_split(X_train_all, Y, test_size = 0.2, random_sta

cat_features = list(range(25))

model = CatBoostClassifier(custom_metric=['AUC'], early_stopping_rounds=100, eval_metric='A

model.fit(X_train, Y_train, cat_features=cat_features, plot=True, verbose=False, use_best_m
```

```
MetricVisualizer(layout=Layout(align_self='stretch', height='500px'))
```

Out[52]:

```
<catboost.core.CatBoostClassifier at 0x7f9d4a28ca20>
```

In [ ]:

```
preds = model.predict_proba(X_cv)[:, 1]

# compute AUC metric for this CV fold
fpr, tpr, thresholds = metrics.roc_curve(Y_cv, preds)
roc_auc = metrics.auc(fpr, tpr)
```

In [ ]:

```
roc_auc
```

Out[54]:

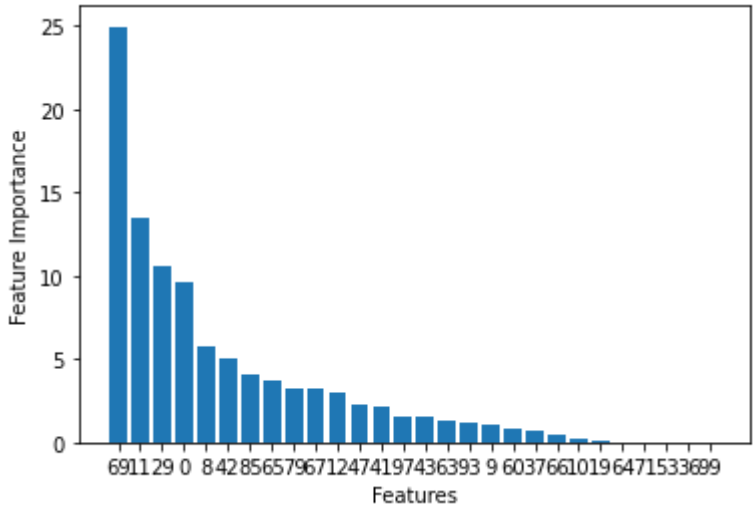
```
0.8961595077606743
```

In [ ]:

```
feat_imp=model.get_feature_importance(prettified=True)
plt.bar(feat_imp['Feature Id'], feat_imp['Importances'])
plt.xlabel('Features')
plt.ylabel('Feature Importance')
```

Out[55]:

```
Text(0, 0.5, 'Feature Importance')
```



Surprising Results! CatBoost Classifier really performs very well with categorical dataset.

##4. ExtraTrees Classifier

In [ ]:

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import StratifiedKFold

kf = StratifiedKFold(n_splits=5,shuffle=True,random_state=seed)

for train_index,test_index in kf.split(X_train_all_categorical, Y):

    X_train = X_train_all_categorical[train_index,:]
    y_train = Y.iloc[train_index]
    X_cv = X_train_all_categorical[test_index,:]
    y_cv = Y.iloc[test_index]

    rf = ExtraTreesClassifier(n_jobs = -1)
    rf.fit(X_train, y_train)
    preds = rf.predict_proba(X_cv)[:, 1]

    # compute AUC metric for this CV fold
    fpr, tpr, thresholds = metrics.roc_curve(y_cv, preds)
    roc_auc = metrics.auc(fpr, tpr)

    print(f"AUC: {roc_auc}")
```

```
AUC: 0.8539147340647132
AUC: 0.8192659139222116
AUC: 0.8560659727550168
AUC: 0.847359470103831
AUC: 0.8577332553828166
```

ExtraTrees Classifier gives AUC of around 0.84.

##5. KNN

Trying Stratified K-Fold CV

In [ ]:

```
from sklearn.model_selection import StratifiedKFold

kf = StratifiedKFold(n_splits=5,shuffle=True,random_state=seed)

for k in [5, 8, 11, 22, 55, 111, 1111]:
    mean_auc = []

    knn = KNeighborsClassifier(n_neighbors=k)

    for train_index,test_index in kf.split(fts, Y):

        X_train = fts[train_index, :]
        y_train = Y.iloc[train_index]
        X_cv = fts[test_index, :]
        y_cv = Y.iloc[test_index]

        knn.fit(X_train, y_train)
        preds = knn.predict_proba(X_cv)[:, 1]

        # compute AUC metric for this CV fold
        fpr, tpr, thresholds = metrics.roc_curve(y_cv, preds)
        roc_auc = metrics.auc(fpr, tpr)

        mean_auc.append(roc_auc)
    print(f"AUC for n= {k}: {np.mean(mean_auc)}")
```

```
AUC for n= 5: 0.8254961915917566
AUC for n= 8: 0.8351992720922716
AUC for n= 11: 0.8398592082911522
AUC for n= 22: 0.8284904574876674
AUC for n= 55: 0.7953198296478907
AUC for n= 111: 0.7673762208710232
AUC for n= 1111: 0.6600229383559151
```

For KNN: n_neighbours = 11 is giving the best results.

##6. XGBoost

In [ ]:

```
## Hyper Parameter Optimization

params={
 "learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
 "max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15],
 "min_child_weight" : [ 1, 3, 5, 7 ],
 "gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
 "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ]

}
```

In [ ]:

```
## Hyperparameter optimization using RandomizedSearchCV
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
import xgboost

classifier=xgboost.XGBClassifier()

random_search=RandomizedSearchCV(classifier, param_distributions=params, n_iter=100, scorir

random_search.fit(X_freq_train,Y)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed:   45.2s
[Parallel(n_jobs=-1)]: Done 124 tasks      | elapsed:  2.7min
[Parallel(n_jobs=-1)]: Done 284 tasks      | elapsed:  6.2min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  6.7min finished
```

Out[45]:

```
RandomizedSearchCV(cv=3, error_score=nan,
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step
=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_alpha=0,
                                           reg_lambda=1, sc...
                                           verbosity=1),
                   iid='deprecated', n_iter=100, n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.3, 0.4, 0.5,
                                                             0.7],
                                        'gamma': [0.0, 0.1, 0.2, 0.3, 0.4],
                                        'learning_rate': [0.05, 0.1, 0.15,
0.2,
                                                          0.25, 0.3],
                                        'max_depth': [3, 4, 5, 6, 8, 10, 12,
                                                      15],
                                        'min_child_weight': [1, 3, 5, 7]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring='roc_auc', verbose=3)
```

In [ ]:

```
random_search.best_score_
```

Out[47]:

```
0.8568873697010403
```

In [ ]:

```
random_search.best_estimator_
```

Out[48]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.4, gamma=0.4,
              learning_rate=0.05, max_delta_step=0, max_depth=15,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

Best XGBoost Estimator is giving AUC: 0.856

We will be using this best XGBoost estimator in stacking.

#Stacking

We are using 5 first level models here: Logistic Regression, Random Forest, CatBoost Classifier, ExtraTrees Classifier, KNN and XGBoost.

And second level model: Logistic Regression

In [ ]:

```
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from catboost import CatBoostClassifier
from sklearn.neighbors import KNeighborsClassifier

y_pred_dataframe_cv = pd.DataFrame()
y_pred_dataframe_test = pd.DataFrame()

seed = 42
kf = StratifiedKFold(n_splits=5,shuffle=True,random_state=seed)
cat_features = list(range(25))

cat = CatBoostClassifier(custom_metric=['AUC'], early_stopping_rounds=100, eval_metric='AUC
model = LogisticRegression(C = 0.5, class_weight = 'balanced')
rf = RandomForestClassifier(n_estimators = 400, min_samples_split = 10, min_samples_leaf =
extc = ExtraTreesClassifier(n_jobs = -1)
knn = KNeighborsClassifier(n_neighbors = 11)
xgb = xgboost.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.4, gamma=0.4,
              learning_rate=0.05, max_delta_step=0, max_depth=15,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)

mean_auc_log = []
mean_auc_rf = []
mean_auc_cat = []
mean_auc_ext = []
mean_auc_knn = []
mean_auc_xgb = []

for train_index,test_index in kf.split(fts, Y):

    #Logistic Regression
    X_train = fts[train_index,:]
    y_train = Y.iloc[train_index]
    X_cv = fts[test_index,:]
    y_cv = Y.iloc[test_index]

    # train model and make predictions
    model.fit(X_train, y_train)
    preds = model.predict_proba(X_cv)[:, 1]

    for idx, k in enumerate(test_index):
        y_pred_dataframe_cv.at[k, 'Id'] = k
        y_pred_dataframe_cv.at[k, 'Logistic Regression'] = preds[idx]

    # compute AUC metric for this CV fold
    fpr, tpr, thresholds = metrics.roc_curve(y_cv, preds)
    roc_auc = metrics.auc(fpr, tpr)

    mean_auc_log.append(roc_auc)

    #Random Forest
    X_train_rf = X_freq_train[train_index, :]
    y_train_rf = Y.iloc[train_index]
    X_cv_rf = X_freq_train[test_index, :]
    y_cv_rf = Y.iloc[test_index]
```

```
# train model and make predictions
rf.fit(X_train_rf, y_train_rf)
rf_preds = rf.predict_proba(X_cv_rf)[:, 1]

for idx, k in enumerate(test_index):
    y_pred_dataframe_cv.at[k, 'Random Forest'] = rf_preds[idx]

# compute AUC metric for this CV fold
fpr, tpr, thresholds = metrics.roc_curve(y_cv_rf, rf_preds)
roc_auc = metrics.auc(fpr, tpr)

mean_auc_rf.append(roc_auc)

#CatBoost Classifier
X_train_cat = X_train_all.iloc[train_index]
y_train_cat = Y.iloc[train_index]
X_cv_cat = X_train_all.iloc[test_index]
y_cv_cat = Y.iloc[test_index]

# train model and make predictions
cat.fit(X_train_cat, y_train_cat, cat_features=cat_features, eval_set=(X_cv_cat, y_cv_c

cat_preds = cat.predict_proba(X_cv_cat)[:, 1]

for idx, k in enumerate(test_index):
    y_pred_dataframe_cv.at[k, 'CatBoost'] = cat_preds[idx]

# compute AUC metric for this CV fold
fpr, tpr, thresholds = metrics.roc_curve(y_cv_cat, cat_preds)
roc_auc = metrics.auc(fpr, tpr)

mean_auc_cat.append(roc_auc)

#ExtraTrees Classifier
X_train_ext = X_train_all_categorical[train_index,:]
y_train_ext = Y.iloc[train_index]
X_cv_ext = X_train_all_categorical[test_index,:]
y_cv_ext = Y.iloc[test_index]

# train model and make predictions
extc.fit(X_train_ext, y_train_ext)

extc_preds = extc.predict_proba(X_cv_ext)[:, 1]

for idx, k in enumerate(test_index):
    y_pred_dataframe_cv.at[k, 'ExtraTrees Classifier'] = extc_preds[idx]

# compute AUC metric for this CV fold
fpr, tpr, thresholds = metrics.roc_curve(y_cv_ext, extc_preds)
roc_auc = metrics.auc(fpr, tpr)

mean_auc_ext.append(roc_auc)

#KNN
X_train = fts[train_index,:]
y_train = Y.iloc[train_index]
X_cv = fts[test_index,:]
y_cv = Y.iloc[test_index]

# train model and make predictions
```

```
    knn.fit(X_train, y_train)
    preds = knn.predict_proba(X_cv)[:, 1]

    for idx, k in enumerate(test_index):
        y_pred_dataframe_cv.at[k, 'Id'] = k
        y_pred_dataframe_cv.at[k, 'KNN'] = preds[idx]

    # compute AUC metric for this CV fold
    fpr, tpr, thresholds = metrics.roc_curve(y_cv, preds)
    roc_auc = metrics.auc(fpr, tpr)

    mean_auc_knn.append(roc_auc)

    #XGBoost Classifier
    X_train_xgb = X_freq_train[train_index, :]
    y_train_xgb = Y.iloc[train_index]
    X_cv_xgb = X_freq_train[test_index, :]
    y_cv_xgb = Y.iloc[test_index]

    # train model and make predictions
    xgb.fit(X_train_xgb, y_train_xgb)
    xgb_preds = xgb.predict_proba(X_cv_xgb)[:, 1]

    for idx, k in enumerate(test_index):
        y_pred_dataframe_cv.at[k, 'XGBoost'] = xgb_preds[idx]

    # compute AUC metric for this CV fold
    fpr, tpr, thresholds = metrics.roc_curve(y_cv_xgb, xgb_preds)
    roc_auc = metrics.auc(fpr, tpr)

    mean_auc_xgb.append(roc_auc)

print(f"AUC for Logistic Regression: {np.mean(mean_auc_log)}")
print(f"AUC for Random Forest: {np.mean(mean_auc_rf)}")
print(f"AUC for CatBoost Classifier: {np.mean(mean_auc_cat)}")
print(f"AUC for ExtraTrees Classifier: {np.mean(mean_auc_ext)}")
print(f"AUC for KNN: {np.mean(mean_auc_knn)}")
print(f"AUC for XGBoost: {np.mean(mean_auc_xgb)}")

preds_test = model.predict_proba(fts_test)[:, 1]
rf_preds_test = rf.predict_proba(X_freq_test)[:, 1]
cat_preds_test = cat.predict_proba(X_test_all)[:, 1]
extc_preds_test = extc.predict_proba(X_test_all_categorical)[:, 1]
knn_preds_test = knn.predict_proba(fts_test)[:, 1]
xgb_preds_test = xgb.predict_proba(X_freq_test)[:, 1]

for idx, k in enumerate(preds_test):
    y_pred_dataframe_test.at[idx, 'CatBoost'] = cat_preds_test[idx]
    y_pred_dataframe_test.at[idx, 'Random Forest'] = rf_preds_test[idx]
    y_pred_dataframe_test.at[idx, 'Logistic Regression'] = preds_test[idx]
    y_pred_dataframe_test.at[idx, 'ExtraTrees Classifier'] = extc_preds_test[idx]
    y_pred_dataframe_test.at[idx, 'KNN'] = knn_preds_test[idx]
    y_pred_dataframe_test.at[idx, 'XGBoost'] = xgb_preds_test[idx]
```

```
MetricVisualizer(layout=Layout(align_self='stretch', height='500px'))
```

```
MetricVisualizer(layout=Layout(align_self='stretch', height='500px'))
```

```
MetricVisualizer(layout=Layout(align_self='stretch', height='500px'))
```

```
MetricVisualizer(layout=Layout(align_self='stretch', height='500px'))
```

```
MetricVisualizer(layout=Layout(align_self='stretch', height='500px'))
```

```
AUC for Logistic Regression: 0.8900040702780189
AUC for Random Forest: 0.8667048293384092
AUC for CatBoost Classifier: 0.8961264121735797
AUC for ExtraTrees Classifier: 0.847721312126477
AUC for KNN: 0.8398592082911522
AUC for XGBoost: 0.8676945849343817
```

In [ ]:

```
y_pred_dataframe_cv.sort_index(axis = 0, inplace= True)
```

In [ ]:

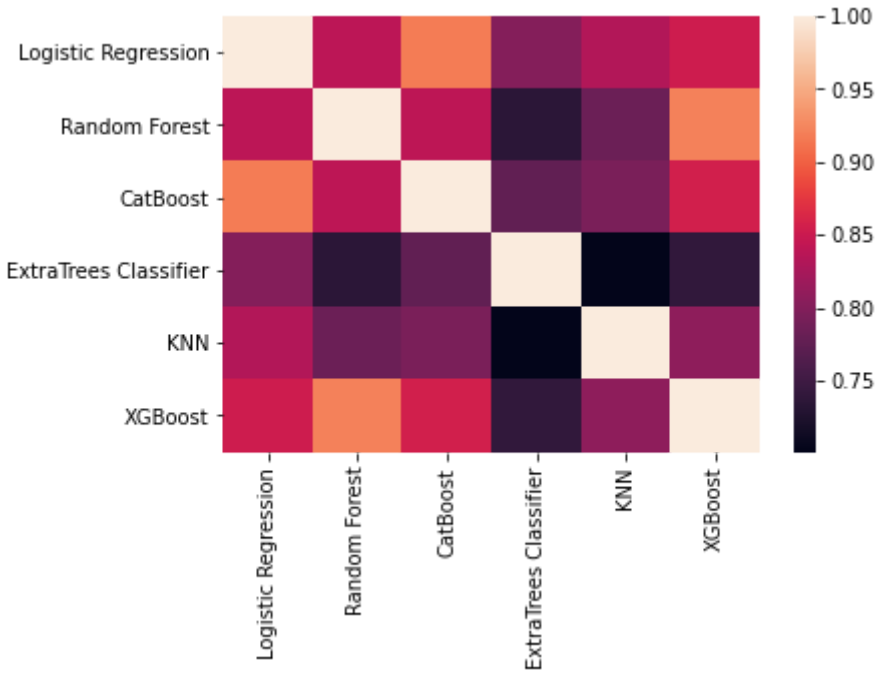
```
y_pred_dataframe_cv = y_pred_dataframe_cv.drop('Id', axis = 1)
```

In [ ]:

```
sns.heatmap(y_pred_dataframe_cv.corr())
```

Out[60]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9d50bbb3c8>
```



Bad News! Looks like Our 2 best models i.e. Logistic Regression and CatBoost Classifier are correlated. XGBoost and Random Forest results are also correlated.

Let's hope for the best.

In [ ]:

```
y_pred_dataframe_test.to_csv('test_stack.csv', index = False)
```

In [ ]:

```
y_pred_dataframe_cv.to_csv('cv_stack.csv', index = False)
```

In [ ]:

```
y_pred_dataframe_cv.head()
```

Out[72]:

	Logistic Regression	Random Forest	CatBoost	ExtraTrees Classifier	KNN	XGBoost
0	0.993932	0.993559	0.995736	0.96	1.000000	0.976608
1	0.949798	0.991467	0.981159	0.94	1.000000	0.954883
2	0.940770	0.976312	0.989623	0.88	0.818182	0.961382
3	0.991040	0.998940	0.989898	1.00	1.000000	0.977093
4	0.973816	0.985039	0.989069	0.97	1.000000	0.956162

We have got our meta-features.

##Final Submission

Using Logistic Regression as a Meta-Model and fitting it on our Meta Features obtained from other first level classifiers.

And later hyperparameter tuning it.

In [ ]:

```
for c in [0.0001, 0.00001, 0.001, 0.01, 0.1, 1]:
    mean_auc = []

    for i in range(10):

        seed = 42
        kf = StratifiedKFold(n_splits=5,shuffle=True,random_state=i*seed)

        for train_index,test_index in kf.split(y_pred_dataframe_cv, Y):
            X_train = y_pred_dataframe_cv.iloc[train_index]
            y_train = Y.iloc[train_index]
            X_cv = y_pred_dataframe_cv.iloc[test_index]
            y_cv = Y.iloc[test_index]

            stack = LogisticRegression(C = c)
            stack.fit(X_train, y_train)
            preds = stack.predict_proba(X_cv)[:, 1]

            # compute AUC metric for this CV fold
            fpr, tpr, thresholds = metrics.roc_curve(y_cv, preds)
            roc_auc = metrics.auc(fpr, tpr)

            mean_auc.append(roc_auc)

    print(f"For C: {c}, AUC: {np.mean(mean_auc)}")
```

```
For C: 0.0001, AUC: 0.895037088579687
For C: 1e-05, AUC: 0.8950340324985734
For C: 0.001, AUC: 0.8950860075040105
For C: 0.01, AUC: 0.8955698449250694
For C: 0.1, AUC: 0.8971255765129232
For C: 1, AUC: 0.894973536410306
```

####C = 0.1 gives the best AUC Score of 0.8971

##FFS

In [ ]:

```
stack_featurewise=[]

for f in y_pred_dataframe_cv.columns:
    stack_featurewise.append(y_pred_dataframe_cv[f].values)
```

In [ ]:

```
#Forward Feature Selection

import warnings
warnings.filterwarnings("ignore")

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import sklearn.metrics as metrics

selected_features = []
best_auc = [0, 0.5]

while best_auc[-1] > best_auc[-2]:
    auc_scores = []

    for idx, f in enumerate(stack_featurewise):

        if idx not in selected_features:
            feats = selected_features + [idx]
            ft = np.hstack([stack_featurewise[j].reshape(-1, 1) for j in feats])
            auc = cv_loop(ft, Y, LogisticRegression(C = 0.1, class_weight = 'balanced'))
            auc_scores.append((idx, auc))

    best_auc.append(max(auc_scores,key=lambda x:x[1])[1])
    selected_features.append(max(auc_scores,key=lambda x:x[1])[0])

    print(f"Feature Set: {selected_features}, AUC: {best_auc[-1]}")
```

```
Feature Set: [2], AUC: 0.8996787412004743
Feature Set: [2, 0], AUC: 0.9014986807387864
Feature Set: [2, 0, 3], AUC: 0.9012687981370109
```

As expected, Logistic Regression and CatBoost Classifier gave best scores above also. And here also we can see that feature 2, 0 are from those classifiers only.

##HyperParameter Tuning again on Selected Features

In [ ]:

```
y_pred_dataframe_cv_selected = y_pred_dataframe_cv.iloc[:, [2, 0]]
y_pred_dataframe_test_selected = y_pred_dataframe_test.iloc[:, [2, 0]]
```

In [ ]:

```
for c in [0.0001, 0.00001, 0.001, 0.01, 0.1, 1]:
    mean_auc = []

    for i in range(10):

        seed = 42
        kf = StratifiedKFold(n_splits=5,shuffle=True,random_state=i*seed)

        for train_index,test_index in kf.split(y_pred_dataframe_cv_selected, Y):
            X_train = y_pred_dataframe_cv_selected.iloc[train_index]
            y_train = Y.iloc[train_index]
            X_cv = y_pred_dataframe_cv_selected.iloc[test_index]
            y_cv = Y.iloc[test_index]

            stack = LogisticRegression(C = c, class_weight = 'balanced')
            stack.fit(X_train, y_train)
            preds = stack.predict_proba(X_cv)[:, 1]

            # compute AUC metric for this CV fold
            fpr, tpr, thresholds = metrics.roc_curve(y_cv, preds)
            roc_auc = metrics.auc(fpr, tpr)

            mean_auc.append(roc_auc)

    print(f"For C: {c}, AUC: {np.mean(mean_auc)}")
```

```
For C: 0.0001, AUC: 0.8973771785728016
For C: 1e-05, AUC: 0.8973952587889923
For C: 0.001, AUC: 0.8972493707010734
For C: 0.01, AUC: 0.8969918093492042
For C: 0.1, AUC: 0.8970142949073767
For C: 1, AUC: 0.897197262029795
```

###C = 0.00001 is best with AUC Score of 0.89739

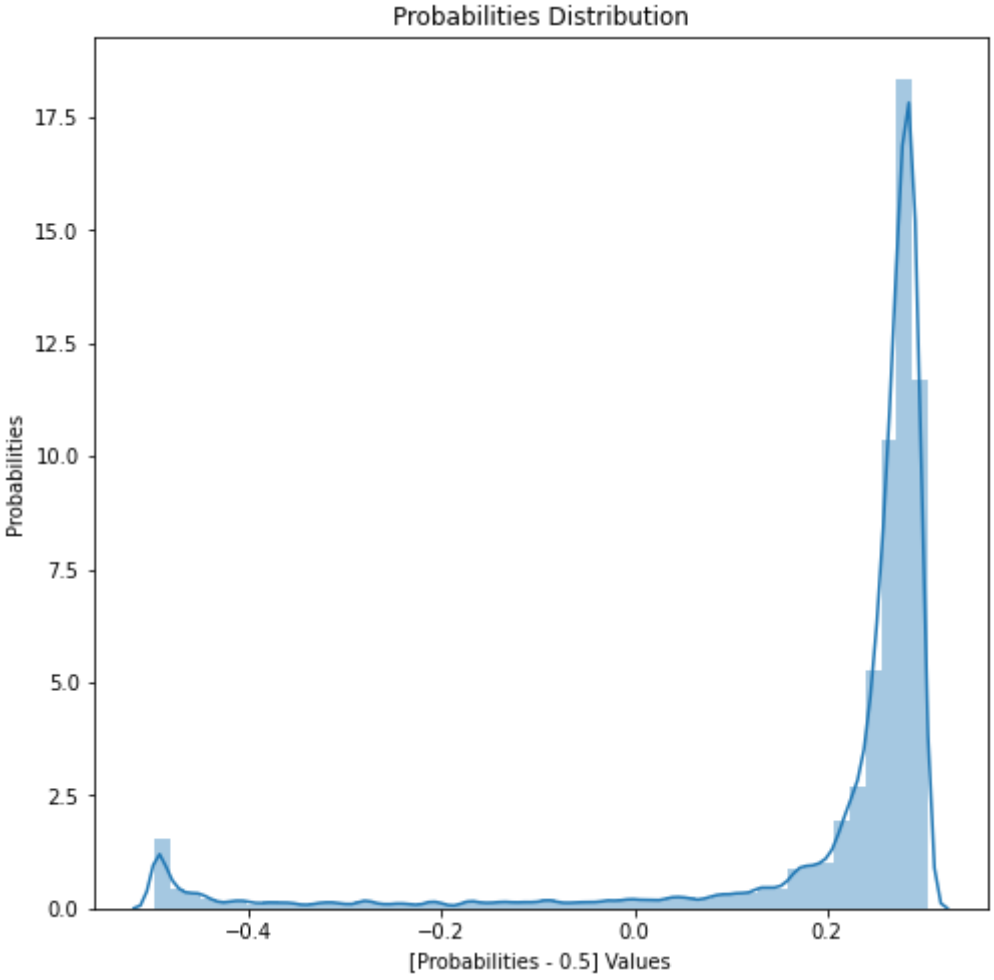
Let's check our predictions plot.

In [ ]:

```
plt.figure(figsize=(8, 8))
sns.distplot(preds-0.5)
plt.xlabel('[Probabilities - 0.5] Values')
plt.ylabel('Probabilities')
plt.title('Probabilities Distribution')
```

Out[93]:

```
Text(0.5, 1.0, 'Probabilities Distribution')
```



We can see the data imbalance here. Very less datapoints are predicted as class 0 compared to class 1.

##Final Submission: Stacked Model with Selected Features

In [ ]:

```
stack = LogisticRegression(C = 0.00001, class_weight = 'balanced')
stack.fit(y_pred_dataframe_cv_selected, Y)

stack_preds = stack.predict_proba(y_pred_dataframe_test_selected)[:, 1]

submission = pd.DataFrame(stack_preds, columns = ['Action'])
submission['Id'] = range(1, len(stack_preds)+1)

submission.to_csv('cat + log + rf + xt+ knn_final_02-07.csv', index = False)
```

In [ ]:

```
submission.head()
```

Out[86]:

	Action	Id
0	0.499191	1
1	0.503872	2
2	0.504184	3
3	0.504121	4
4	0.504218	5

###Meh. Private AUC: 0.89837, Public AUC: 0.90338

AUC has dropped a little.

#Summary

In [1]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Model", "Cross Validation AUC", "Submission- Public AUC"]

x.add_row(["Base Model: Logistic Regression", 0.8497, "-"])
x.add_row(["Logistic Regression with Hybrid Features", 0.8825, "-"])
x.add_row(["Logistic Regression with Forward Feature Selection", 0.8943, 0.9052])
x.add_row(["Random Forest", 0.8575, "-"])
x.add_row(["CatBoost", 0.8961, 0.8964])
x.add_row(["ExtraTrees", 0.8560, "-"])
x.add_row(["KNN", 0.8398, "-"])
x.add_row(["XGBoost", 0.8569, "-"])
x.add_row(["Stacked Model: Logistic Regression + CatBoost Classifier", 0.8974, 0.9034])

print(x)
```

```
+-------------------------------------------------------------+----------------
------+------------------------+
|                          Model                              | Cross Validatio
n AUC | Submission- Public AUC |
+-------------------------------------------------------------+----------------
------+------------------------+
|             Base Model: Logistic Regression                 |        0.8497
|           -             |
|         Logistic Regression with Hybrid Features            |        0.8825
|           -             |
|    Logistic Regression with Forward Feature Selection       |        0.8943
|         0.9052          |
|                       Random Forest                         |        0.8575
|           -             |
|                          CatBoost                           |        0.8961
|         0.8964          |
|                         ExtraTrees                          |        0.856
|           -             |
|                            KNN                              |        0.8398
|           -             |
|                          XGBoost                            |        0.8569
|           -             |
| Stacked Model: Logistic Regression + CatBoost Classifier    |        0.8974
|         0.9034          |
+-------------------------------------------------------------+----------------
------+------------------------+
```

#Best Submission:

##AUC on Public LeaderBoard: 0.90520 and on Private LeaderBoard: 0.89930

###Which is Top 15%.

#Conclusion

This was a very interesting challenge for a begineer. I learnt alot from this exercise. Tried various encoding techniques, machine learning models and compared their performance.

Learnt more about concepts like Cross-Validation, How to handle imbalanced data, Forward Feature Selection etc.

However the AUC score can be improved further. Here are some of the keys for further improvement-

1. Stacked model not performing well due to overfitting. Possible reason might be data leakage.
2. We can hyperparameter tune Random Forest, ExtraTrees and XGBoost more rigorously.

Anyways, I will consider this as a good try as this was my first hand experience with kaggle. Its fierce!