```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE
from lightgbm import LGBMClassifier
from sklearn.model_selection import GridSearchCV, cross_validate
from imblearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, confusion_matrix, classification_report, accuracy_
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
import seaborn as sns
# !pip install xgboost
from xgboost import XGBClassifier
```

importing libraries

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

Load data

```
data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/UMAA/data_for_model.csv')
valid_data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/UMAA/all_data_for_final.csv
```

```
data = data.set_index('ID_DEMO')
valid_data = valid_data.set_index('ID_DEMO')
print(data.columns)
```

```
Index(['MARITAL_STATUS', 'GENDER', 'AGE', 'IN_TC_METRO_AREA',
       'IS_CURRENT_TC_EMPLOYEE', 'ATHLETIC_INTEREST', 'TRAVEL_INTEREST',
       'AFFINITY_NETWORK_INTEREST', 'WEB_TOPIC_OPT_INS', 'UMN_event',
       'UMN_member', 'UMN_donor', 'UMN_volun', 'UMN_inform', 'UMN_loyalty',
       'UMN_avg_Annual_score_5_years', 'annual_member', 'life_member',
       'non_member', 'Learning_emails', 'Legislature_emails', 'Social_emails',
       'Sports_emails', 'general_ctr_emails', 'Learning_events',
```

```
       'Legislature_events', 'Networking_events', 'Other_events',
       'Social_events', 'Sports_events', 'total_type_person_events', 'target'],
      dtype='object')
```

Seperate features and Target. Split the data into test and train sets

```
y = data['target'].ravel()
X = data.drop(['target'], axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10, random_state=42, st
```

Do preprocessing.

1. Impute missing values
2. Covers multi category variables into dummies
3. Scale all numeric variables
4. PCA to reduce dimensions
5. SMOTE to remove imbalence

Use Logistic Regression as a baseline classifier

```
numeric_features = ['AGE', 'UMN_event',
       'UMN_member', 'UMN_donor', 'UMN_volun', 'UMN_inform', 'UMN_loyalty',
       'UMN_avg_Annual_score_5_years', 'annual_member', 'life_member',
       'non_member', 'Learning_emails', 'Legislature_emails', 'Social_emails',
       'Sports_emails', 'general_ctr_emails', 'Learning_events',
       'Legislature_events', 'Networking_events', 'Other_events',
       'Social_events', 'Sports_events', 'total_type_person_events']
numeric_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())])

categorical_features = ['MARITAL_STATUS']
categorical_transformer = Pipeline(steps=[
                                        ('imputer', SimpleImputer(strategy='constant', fill
                                        ('onehot', OneHotEncoder(drop='first'))])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
       ('cat', categorical_transformer, categorical_features)
       ], n_jobs=-1)

# Append classifier to preprocessing pipeline.
# Now we have a full prediction pipeline.
pipe = Pipeline(steps=[('preprocessor', preprocessor),
                       ('smt', SMOTE(random_state=42, sampling_strategy=1)),
                       ('pca', PCA(n_components='mle')),
                       #('classifier', LGBMClassifier(boosting_type='gbdt', objective='binary'
                       ('clf', LogisticRegression(n_jobs=-1))])
```

using logistic regression for baseline

```
                    ('clf', LogisticRegression(n_jobs=-1))])
```

```
pipe.steps
```

```
[('preprocessor',
  ColumnTransformer(n_jobs=-1, remainder='drop', sparse_threshold=0.3,
                    transformer_weights=None,
                    transformers=[('num',
                                   Pipeline(memory=None,
                                            steps=[('scaler',
                                                    StandardScaler(copy=True,
                                                                   with_mean=True,
                                                                   with_std=True))],
                                            verbose=False),
                                   ['AGE', 'UMN_event', 'UMN_member', 'UMN_donor',
                                    'UMN_volun', 'UMN_inform', 'UMN_loyalty',
                                    'UMN_avg_Annual_score_5_years',
                                    'annual_member...
                                    'total_type_person_events']),
                                  ('cat',
                                   Pipeline(memory=None,
                                            steps=[('imputer',
                                                    SimpleImputer(add_indicator=False,
                                                                  copy=True,
                                                                  fill_value='missing',
                                                                  missing_values=nan,
                                                                  strategy='constant',
                                                                  verbose=0)),
                                                   ('onehot',
                                                    OneHotEncoder(categories='auto',
                                                                  drop='first',
                                                                  dtype=<class 'numpy.f]
                                                                  handle_unknown='error
                                                                  sparse=True))],
                                            verbose=False),
                                   ['MARITAL_STATUS'])],
                    verbose=False)),
 ('smt',
  SMOTE(k_neighbors=5, kind='deprecated', m_neighbors='deprecated', n_jobs=1,
        out_step='deprecated', random_state=42, ratio=None, sampling_strategy=1,
        svm_estimator='deprecated')),
 ('pca',
  PCA(copy=True, iterated_power='auto', n_components='mle', random_state=None,
      svd_solver='auto', tol=0.0, whiten=False)),
 ('clf',
  LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                     intercept_scaling=1, l1_ratio=None, max_iter=100,
                     multi_class='auto', n_jobs=-1, penalty='l2',
                     random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                     warm_start=False))]
```

Crossvalidation to get a better estimate of accuracy metric

```
scores = cross_validate(pipe, X_train, y_train, scoring='roc_auc', cv=5, n_jobs=-1, return_es
```

```
scores['test_score'].mean()
pipe.fit(X_train, y_train)
```

Accuracy Metrics

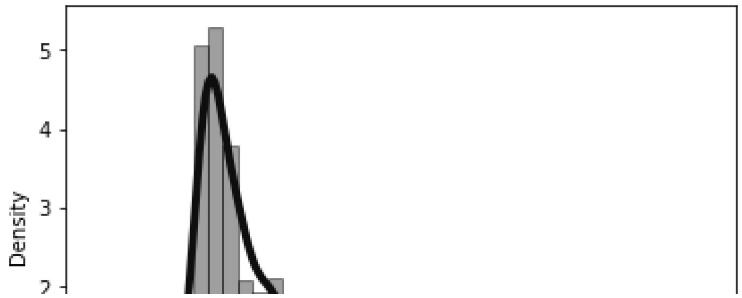
```
y_pred_prob = clf.predict_proba(X_test)
y_pred = clf.predict(X_test)

print(roc_auc_score(y_test, y_pred_prob[:,1]))

print(classification_report(y_test, y_pred))

# Get all measurement
columns_name = ['Accuracy', 'Precision', 'Recall', 'F1-Score']
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy of Model is {0}'.format(round(accuracy,4)))
print(np.unique(y_test, return_counts=True))
print(confusion_matrix(y_test, y_pred))
```

```
                   precision    recall  f1-score   support

               0       1.00      0.84      0.91     50085
               1       0.03      0.70      0.05       303

        accuracy                           0.84     50388
       macro avg       0.51      0.77      0.48     50388
    weighted avg       0.99      0.84      0.91     50388

    Accuracy of [redacted] Model is 0.8402
    (array([0, 1]), array([50085,   303]))
    [[42124  7961]
     [   92   211]]
```

Histogram of predicted probabilities of test set

```
sns.distplot(y_pred_prob[:,1], hist=True, kde=True,
             bins=int(180/5), color = 'darkblue',
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
```

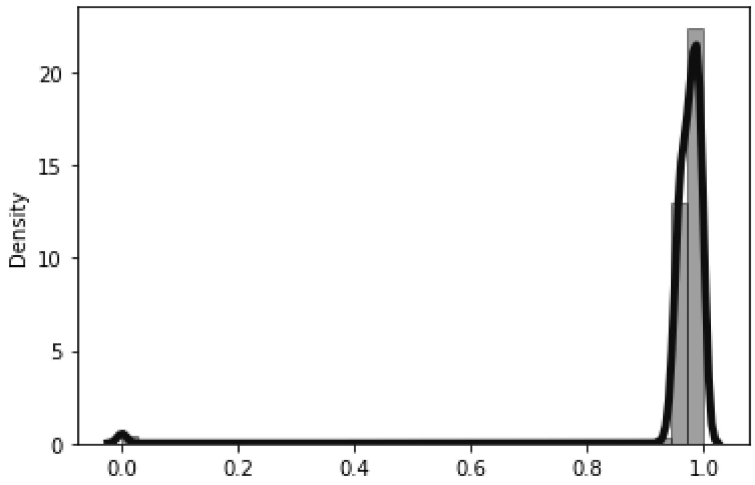
```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2551: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f9bb9ac2710>
```



Histogram of predicted probabilities of New data

```
a= clf.predict_proba(valid_data)[:,1]

sns.distplot(a, hist=True, kde=True,
             bins=int(180/5), color = 'darkblue',
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2551: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7fc8c3f85c88>
```



Testing more Algorithms

```
clfs = []
clfs.append(KNeighborsClassifier(n_neighbors=3))
clfs.append(DecisionTreeClassifier())
clfs.append(RandomForestClassifier())
clfs.append(GradientBoostingClassifier())
clfs.append(SVC())
clfs.append(LogisticRegression())
```

```
    for classifier in clfs:
        pipe.set_params(clf = classifier)
        scores = cross_validate(pipe, X_train, y_train, scoring='roc_auc', cv=5, n_jobs=-1)
        print('---------------------------------')
        print(str(classifier))
        print('---------------------------------')
        for key, values in scores.items():
                print(key,' mean ', values.mean())
                print(key,' std ', values.std())
```

```
----------------------------------
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
----------------------------------
fit_time  mean  16.978074741363525
fit_time  std  2.2851621136792577
score_time  mean  29.086413764953612
score_time  std  4.974008105280437
test_score  mean  0.6682074380456581
test_score  std  0.011775460643893832
----------------------------------
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, 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, presort='deprecated',
                       random_state=None, splitter='best')
----------------------------------
fit_time  mean  74.68439955711365
fit_time  std  5.392215320066669
score_time  mean  0.19594912528991698
score_time  std  0.013061924500099905
test_score  mean  0.555229171633138
test_score  std  0.015362357740034804
----------------------------------
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=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
----------------------------------
fit_time  mean  674.5734558105469
```

Using Gradient Boosting Classifier as it has the highest AUC under ROC score

```
score_time  std  0.19658772999940014
```

```
pipe = Pipeline(steps=[('preprocessor', preprocessor),
                       ('smt', SMOTE(random_state=42, sampling_strategy=1)),
                       ('pca', PCA(n_components='mle')),
                       #('classifier', LGBMClassifier(boosting_type='gbdt', objective='binary'
                       ('clf', XGBClassifier())])
```

```
                              min_samples_leaf=1, min_samples_split=2,
```

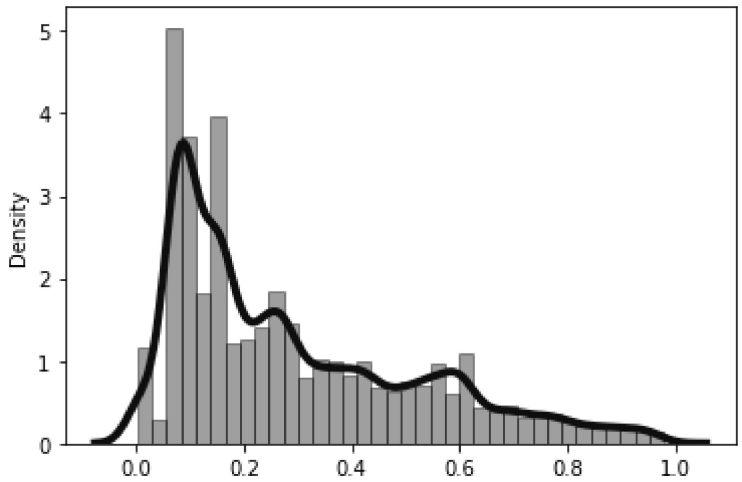
Performace evaluation of Gradient Boosting Classifier

```
                              random_state=None, subsample=1.0, tol=0.0001,
```

```
pipe.fit(X_train, y_train)
y_pred_prob = pipe.predict_proba(X_test)
y_pred = pipe.predict(X_test)

print(roc_auc_score(y_test, y_pred_prob[:,1]))
```

```
print(classification_report(y_test, y_pred))

# Get all measurement
columns_name = ['Accuracy', 'Precision', 'Recall', 'F1-Score']
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy of Model is {0}'.format(round(accuracy,4)))
print(np.unique(y_test, return_counts=True))
print(confusion_matrix(y_test, y_pred))


sns.distplot(y_pred_prob[:,1], hist=True, kde=True,
             bins=int(180/5), color = 'darkblue',
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: F
  warnings.warn(msg, category=FutureWarning)
0.8808422381621211
              precision    recall  f1-score   support

           0       1.00      0.80      0.89     50085
           1       0.02      0.80      0.05       303

    accuracy                           0.80     50388
   macro avg       0.51      0.80      0.47     50388
weighted avg       0.99      0.80      0.88     50388

Accuracy of Model is 0.7985
(array([0, 1]), array([50085,   303]))
[[39993 10092]
 [   62   241]]
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2551: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7fe399c73f60>
```




```
a= pipe.predict_proba(valid_data)[:,1]

sns.distplot(a, hist=True, kde=True
```

```
sns.distplot(a, hist=True, kde=True,
             bins=int(180/5), color = 'darkblue',
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2551: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7fe39a50b358>
```