Hyperparameter Tunning:

Classifiers:

1. ***XGB***

* from xgboost import XGBClassifier
* xgb\_classifier.get\_params()
* Default parameters:
* {'objective': 'multi:softprob',
* 'use\_label\_encoder': True,
* 'base\_score': 0.5,
* 'booster': 'gbtree',
* 'colsample\_bylevel': 1,
* 'colsample\_bynode': 1,
* 'colsample\_bytree': 1,
* 'gamma': 0,
* 'gpu\_id': -1,
* 'importance\_type': 'gain',
* 'interaction\_constraints': '',
* 'learning\_rate': 0.300000012,
* 'max\_delta\_step': 0,
* 'max\_depth': 6,
* 'min\_child\_weight': 1,
* 'missing': nan,
* 'monotone\_constraints': '()',
* 'n\_estimators': 100,
* 'n\_jobs': 8,
* 'num\_parallel\_tree': 1,
* 'random\_state': 100,
* 'reg\_alpha': 0,
* 'reg\_lambda': 1,
* 'scale\_pos\_weight': None,
* 'subsample': 1,
* 'tree\_method': 'exact',
* 'validate\_parameters': 1,
* 'verbosity': None}

param\_grid={

'learning\_rate':[0.5,0.1,0.01,0.001,1],

'max\_depth': range(5,200,25),

'n\_estimators':range(10,300,50), #[100,200,250,300,350],

'min\_child\_weight' : np.arange(1,20,2),

'gamma' : [0.1, 0.2, 0.3],

'subsample': np.arange(0.5, 1, 0.1)

}

grid\_xgb = RandomizedSearchCV(estimator=xgb\_classifier , param\_distributions = param\_grid, n\_jobs=-1, verbose = 3, cv = 10,random\_state = 100, n\_iter = 10, scoring='f1\_macro')

grid\_xgb.fit(xtrain, ytrain)

best\_parameters = grid\_xgb.best\_params\_

print(best\_parameters)

1. ***Random Forest Classifier***

* sklearn.ensemble import RandomForestClassifier
* Default parameters:
* {'bootstrap': True,
* 'ccp\_alpha': 0.0,
* 'class\_weight': None,
* 'criterion': 'entropy',
* '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': 90,
* 'n\_jobs': None,
* 'oob\_score': False,
* 'random\_state': 100,
* 'verbose': 0,
* 'warm\_start': False}

param\_grid = {

"n\_estimators" : [90,100,115,130],

'criterion': ['gini', 'entropy'],

'max\_depth' : range(2,20,1),

'min\_samples\_leaf' : range(1,10,1),

'min\_samples\_split': range(2,10,1),

'max\_features' : ['auto','log2']

}

grid\_rf = RandomizedSearchCV(estimator=rf\_classifier , param\_distributions = param\_grid, n\_jobs=-1, verbose = 3, cv = 10,random\_state = 100, n\_iter = 12, scoring='f1\_macro')

grid\_rf.fit(xtrain, ytrain)

best\_parameters = grid\_rf.best\_params\_

print(best\_parameters)

1. ***LIGHT GRADIENT BOOSTING***

* pip install lightgbm
* import lightgbm as lgb
* {'boosting\_type': 'gbdt',
* 'class\_weight': None,
* 'colsample\_bytree': 1.0,
* 'importance\_type': 'split',
* 'learning\_rate': 0.15,
* 'max\_depth': -1,
* 'min\_child\_samples': 20,
* 'min\_child\_weight': 0.001,
* 'min\_split\_gain': 0.0,
* 'n\_estimators': 100,
* 'n\_jobs': -1,
* 'num\_leaves': 31,
* 'objective': 'multi',
* 'random\_state': 1,
* 'reg\_alpha': 0.0,
* 'reg\_lambda': 0.0,
* 'silent': True,
* 'subsample': 1.0,
* 'subsample\_for\_bin': 200000,
* 'subsample\_freq':

from scipy.stats import randint as sp\_randint

from scipy.stats import uniform as sp\_uniform

param\_grid = {

'num\_leaves': sp\_randint(6, 50),

'min\_child\_samples': sp\_randint(100, 500),

'min\_child\_weight': [1e-5, 1e-3, 1e-2, 1e-1, 1, 1e1, 1e2, 1e3, 1e4],

'subsample': sp\_uniform(loc=0.2, scale=0.8),

'colsample\_bytree': sp\_uniform(loc=0.4, scale=0.6),

'reg\_alpha': [0, 1e-1, 1, 2, 5, 7, 10, 50, 100],

'reg\_lambda': [0, 1e-1, 1, 5, 10, 20, 50, 100],

'max\_depth': sp\_randint(5,200,25), #add

'n\_estimators':sp\_randint(10,300,50), #add

}

grid\_lgb = RandomizedSearchCV(estimator=lgb\_classifier , param\_distributions = param\_grid, n\_jobs=-1, verbose = 3, cv = 10, random\_state = 100, n\_iter = 12, scoring='f1\_macro')#'neg\_mean\_absolute\_error')

grid\_lgb.fit(xtrain, ytrain)

best\_parameters = grid\_lgb.best\_params\_

print(best\_parameters)

1. ***CATBOOST***

* pip install catboost
* import catboost as cb

param\_grid = {#'depth': [4, 7, 10],

#'learning\_rate' : [0.03, 0.1, 0.15],

'l2\_leaf\_reg': [1,4,9],

'iterations': range(50,300,50),

#added

#'learning\_rate': np.arange(0.03, 0.1, 0.05),

'max\_depth': np.arange(3, 15, 1),

'colsample\_bylevel': np.arange(0.3, 0.8, 0.1),

}

grid\_cb = RandomizedSearchCV(estimator=cb\_classifier , param\_distributions = param\_grid, n\_jobs=-1, verbose = 3, cv = 10, random\_state = 100, n\_iter = 12, scoring='f1\_macro')#'neg\_mean\_absolute\_error')

grid\_cb.fit(xtrain, ytrain)

best\_parameters = grid\_cb.best\_params\_

print(best\_parameters)

**CLUSTERING APPROACH**

**Simple Approach**

def xgb\_function(xtrain, ytrain):

#xgb\_classifier = XGBClassifier()

xgbclf = XGBClassifier(random\_state = 100)

xgbclf.fit(xtrain, ytrain)

return xgbclf

def lgb\_function(xtrain, ytrain):

lgbclf = lgb.LGBMClassifier(colsample\_bytree=0.9640178917734248,

min\_child\_samples=149,

min\_child\_weight=0.001,

num\_leaves = 20,

reg\_alpha = 50,

reg\_lambda = 100,

subsample = 0.34289945963435475)

lgbclf.fit(xtrain, ytrain)

return lgbclf

def cb\_function(xtrain, ytrain):

cbclf = cb.CatBoostClassifier(depth = 4,

learning\_rate = 0.1,

l2\_leaf\_reg = 4,

iterations = 300)

cbclf.fit(xtrain, ytrain)

return cbclf

for i in list\_of\_clusters:

cluster\_data = train\_data[train\_data['Clusters']==i]

cluster\_features = cluster\_data.drop(['Clusters', 'churn\_risk\_score'], axis = 1) # X

cluster\_label = cluster\_data['churn\_risk\_score'] # Y

xtrain, xtest, ytrain, ytest = train\_test\_split(cluster\_features, cluster\_label, test\_size = 1/3, random\_state = 355)

xgb\_fun = xgb\_function(xtrain, ytrain)

xgb\_pred = xgb\_fun.predict(xtest)

xgb\_score = 100 \* f1\_score(ytest, xgb\_pred, average="macro")

lgb\_fun = lgb\_function(xtrain, ytrain)

lgb\_pred = lgb\_fun.predict(xtest)

lgb\_score = 100 \* f1\_score(ytest, lgb\_pred, average="macro")

cb\_fun = cb\_function(xtrain, ytrain)

cb\_pred = cb\_fun.predict(xtest)

cb\_score = 100 \* f1\_score(ytest, cb\_pred, average="macro")

if ((xgb\_score > lgb\_score) & (xgb\_score > cb\_score)):

print('CLUSTER ',i, ' XGB SCORE: ', xgb\_score)

elif ((lgb\_score > xgb\_score) & (lgb\_score > cb\_score)):

print('CLUSTER ',i, ' LGB SCORE: ', lgb\_score)

else:

print('CLUSTER ',i, ' CB SCORE: ', cb\_score)

"""#for xgb

xgb\_fun = xgb\_function(xtrain, ytrain)

xgb\_pred = xgb\_fun.predict(xtest)

xgb\_score = 100 \* f1\_score(ytest, xgb\_pred, average="macro")

print('CLUSTER ',i, ' XGB SCORE: ', xgb\_score)

#LGB Hyperparameter tuned

lgb\_fun = lgb\_function(xtrain, ytrain)

lgb\_pred = lgb\_fun.predict(xtest)

lgb\_score = 100 \* f1\_score(ytest, lgb\_pred, average="macro")

print('CLUSTER ',i, ' LGB SCORE: ', lgb\_score)

#Cat Boost Hypertuned

cb\_fun = cb\_function(xtrain, ytrain)

cb\_pred = cb\_fun.predict(xtest)

cb\_score = 100 \* f1\_score(ytest, cb\_pred, average="macro")

print('CLUSTER ',i, ' CB SCORE: ', cb\_score)"""

**Hyper tuning Approach**

#def xgb\_function(xtrain, ytrain):

param\_grid={

'learning\_rate':[0.5,0.1,0.01,0.001,1],

'max\_depth': range(5,200,25),

'n\_estimators':range(10,300,50), #[100,200,250,300,350],

'min\_child\_weight' : np.arange(1,20,2),

'gamma' : [0.1, 0.2, 0.3],

'subsample': np.arange(0.5, 1, 0.1)

}

xgb\_classifier = XGBClassifier()

grid\_xgb = RandomizedSearchCV(estimator=xgb\_classifier , param\_distributions = param\_grid, n\_jobs=-1, verbose = 3, cv = 10,

random\_state = 100, n\_iter = 10, scoring='f1\_macro')

grid\_xgb.fit(xtrain, ytrain)

#parameter values

learning\_rate = grid\_xgb.best\_params\_['learning\_rate']

max\_depth = grid\_xgb.best\_params\_['max\_depth']

n\_estimators = grid\_xgb.best\_params\_['n\_estimators']

min\_child\_weight = grid\_xgb.best\_params\_['min\_child\_weight']

gamma = grid\_xgb.best\_params\_['gamma']

subsample = grid\_xgb.best\_params\_['subsample']

xgb\_classifier = XGBClassifier(learning\_rate = learning\_rate,

max\_depth = max\_depth,

n\_estimators = n\_estimators,

gamma = gamma,

min\_child\_weight = min\_child\_weight,

subsample = subsample)

xgb\_classifier.fit(xtrain, ytrain)

return xgb\_classifier

def lgb\_function(xtrain, ytrain):

param\_grid = {

'num\_leaves': sp\_randint(6, 50),

'min\_child\_samples': sp\_randint(100, 500),

'min\_child\_weight': [1e-5, 1e-3, 1e-2, 1e-1, 1, 1e1, 1e2, 1e3, 1e4],

'subsample': sp\_uniform(loc=0.2, scale=0.8),

'colsample\_bytree': sp\_uniform(loc=0.4, scale=0.6),

'reg\_alpha': [0, 1e-1, 1, 2, 5, 7, 10, 50, 100],

'reg\_lambda': [0, 1e-1, 1, 5, 10, 20, 50, 100]

}

lgb\_classifier = LGBMClassifier()

grid\_lgb = RandomizedSearchCV(estimator=lgb\_classifier , param\_distributions = param\_grid, n\_jobs=-1, verbose = 3, cv = 10,

random\_state = 100, n\_iter = 12, scoring='f1\_macro')#'neg\_mean\_absolute\_error')

grid\_lgb.fit(xtrain, ytrain)

#parameter values

num\_leaves = grid\_lgb.best\_params\_['num\_leaves']

min\_child\_samples = grid\_lgb.best\_params\_['min\_child\_samples']

min\_child\_weight = grid\_lgb.best\_params\_['min\_child\_weight']

subsample = grid\_lgb.best\_params\_['subsample']

colsample\_bytree = grid\_lgb.best\_params\_['colsample\_bytree']

reg\_alpha = grid\_lgb.best\_params\_['reg\_alpha']

reg\_lambda = grid\_lgb.best\_params\_['reg\_lambda']

lgb\_classifier = LGBMClassifier(num\_leaves = num\_leaves,

min\_child\_samples = min\_child\_samples,

min\_child\_weight = min\_child\_weight,

subsample = subsample,

colsample\_bytree = colsample\_bytree,

reg\_alpha = reg\_alpha,

reg\_lambda = reg\_lambda)

lgb\_classifier.fit(xtrain, ytrain)

return lgb\_classifier

def cb\_function(xtrain, ytrain):

param\_grid = {'depth': [4, 7, 10],

'learning\_rate' : [0.03, 0.1, 0.15],

'l2\_leaf\_reg': [1,4,9],

'iterations': [300]

}

cb\_classifier = CatBoostClassifier()

grid\_cb = RandomizedSearchCV(estimator=cb\_classifier , param\_distributions = param\_grid, n\_jobs=-1, verbose = 3, cv = 10,

random\_state = 100, n\_iter = 12, scoring='f1\_macro')

grid\_cb.fit(xtrain, ytrain)

#parameter values

depth = grid\_cb.best\_params\_['depth']

learning\_rate = grid\_cb.best\_params\_['learning\_rate']

l2\_leaf\_reg = grid\_cb.best\_params\_['l2\_leaf\_reg']

iterations = grid\_cb.best\_params\_['iterations']

lgb\_classifier = CatBoostClassifier(depth = depth,

learning\_rate = learning\_rate,

l2\_leaf\_reg = l2\_leaf\_reg,

iterations = iterations)

cb\_classifier.fit(xtrain, ytrain)

return cb\_classifier

for i in list\_of\_clusters:

cluster\_data = train\_data[train\_data['Clusters']==i]

cluster\_features = cluster\_data.drop(['Clusters', 'churn\_risk\_score'], axis = 1) # X

cluster\_label = cluster\_data['churn\_risk\_score'] # Y

xtrain, xtest, ytrain, ytest = train\_test\_split(cluster\_features, cluster\_label, test\_size = 1/3, random\_state = 355)

xgb\_fun = xgb\_function(xtrain, ytrain)

xgb\_pred = xgb\_fun.predict(xtest)

xgb\_score = 100 \* f1\_score(ytest, xgb\_pred, average="macro")

lgb\_fun = lgb\_function(xtrain, ytrain)

lgb\_pred = lgb\_fun.predict(xtest)

lgb\_score = 100 \* f1\_score(ytest, lgb\_pred, average="macro")

cb\_fun = cb\_function(xtrain, ytrain)

cb\_pred = cb\_fun.predict(xtest)

cb\_score = 100 \* f1\_score(ytest, cb\_pred, average="macro")

if ((xgb\_score > lgb\_score) & (xgb\_score > cb\_score)):

print('CLUSTER ',i, ' XGB SCORE: ', xgb\_score)

elif ((lgb\_score > xgb\_score) & (lgb\_score > cb\_score)):

print('CLUSTER ',i, ' LGB SCORE: ', lgb\_score)

else:

print('CLUSTER ',i, ' CB SCORE: ', cb\_score)

"""#for xgb

xgb\_fun = xgb\_function(xtrain, ytrain)

xgb\_pred = xgb\_fun.predict(xtest)

xgb\_score = 100 \* f1\_score(ytest, xgb\_pred, average="macro")

print('CLUSTER ',i, ' XGB SCORE: ', xgb\_score)

#LGB Hyperparameter tuned

lgb\_fun = lgb\_function(xtrain, ytrain)

lgb\_pred = lgb\_fun.predict(xtest)

lgb\_score = 100 \* f1\_score(ytest, lgb\_pred, average="macro")

print('CLUSTER ',i, ' LGB SCORE: ', lgb\_score)

#Cat Boost Hypertuned

cb\_fun = cb\_function(xtrain, ytrain)

cb\_pred = cb\_fun.predict(xtest)

cb\_score = 100 \* f1\_score(ytest, cb\_pred, average="macro")

print('CLUSTER ',i, ' CB SCORE: ', cb\_score)"""