There are three models used in this notebook

- XGBClassifier
- CatBoostClassifier
- RandomForestClassifier

Our data is treated in such a way that the data used for the xgbclassifier and the catboostclassifier are left with the null values missing but the data used for the randomforest classifier have nul values filled with -999

Training Data

- X : used for the xgbclassifier and the catboostclassifier
- X_fill : used for the randomforestclassifier
- y : The target columns for both features

Testing Data

- test : Used for prediction by the catboost and xgboost classifier
- test_fill : Used for prediction by the RandomforestClassifier

In [1]:

```
# Importing lbraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Working with test data

In [2]:

```
# Importing our data
df = pd.read_csv("Train.csv")
df.head()
```

Out[2]:

	Applicant_ID	form_field1	form_field2	form_field3	form_field4	form_field5	form_field6	form_field7	form_field8	form_field9	...
0	Apcnt_1000000	3436.0	0.28505	1.6560	0.0	0.000	0.0	10689720.0	252072.0	4272776.0	...
1	Apcnt_1000004	3456.0	0.67400	0.2342	0.0	0.000	0.0	898979.0	497531.0	9073814.0	...
2	Apcnt_1000008	3276.0	0.53845	3.1510	0.0	6.282	NaN	956940.0	NaN	192944.0	...
3	Apcnt_1000012	3372.0	0.17005	0.5050	0.0	0.000	192166.0	3044703.0	385499.0	3986472.0	...
4	Apcnt_1000016	3370.0	0.77270	1.1010	0.0	0.000	1556.0	214728.0	214728.0	1284089.0	...

5 rows × 52 columns

In [3]:

```
# Dropping Applicant column
df.drop("Applicant_ID", axis = 1, inplace = True)
```

In [4]:

```
from sklearn.preprocessing import LabelEncoder
encoder1 = LabelEncoder()
encoder2 = LabelEncoder()
```

In [5]:

```
# Encoding our categorical columns
df["form_field47"] = encoder1.fit_transform(df["form_field47"])
df["default_status"] = encoder2.fit_transform(df["default_status"])
```

In [6]:

```
from sklearn.preprocessing import StandardScaler
```

In [7]:

```
# scaler for data with missing values
scaler = StandardScaler()

# Scaler for data without missing values
scaler1 = StandardScaler()
```

In [8]:

```
# creating our features and target columns. this contains missing data to be used for xgboost and
catclassifier
X = df.drop("default_status", axis = 1)
y = df["default_status"].values
```

In [9]:

```
# Creating a new features Dataset and filling nan with -999
# This is to be used for the random forest classifier
X_fill = X.fillna(-999)
```

In [10]:

```
# Scaling the features for the catboost and xgboost models
# This features contain missing values
X = scaler.fit_transform(X)

# Scaling the features for the random forest classifier
# This contains no missing features
X_fill = scaler1.fit_transform(X_fill)
```

In [11]:

```
from sklearn.metrics import classification_report, confusion_matrix, mean_absolute_error,
mean_squared_error
```

Importing our models

In [12]:

```
# Model 1: CatBoostClassifier
from catboost import CatBoostClassifier
cbc = CatBoostClassifier()
```

In [13]:

```
# Model 2: XGBClassifier
from xgboost import XGBClassifier
xgb = XGBClassifier()
```

In [14]:

```
# Model 3: RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
```

In [15]:

```
# Model 4: LGBMClassifier
from lightgbm import LGBMClassifier
lgb = LGBMClassifier(max_depth=-1,
                     random_state=314,
                     silent=True,
                     metric='None',
                     # n_jobs=4,
                     n_estimators=5000)
```

Importing our search mechanism

In [16]:

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.model_selection import StratifiedKFold
```

In [17]:

```
folds      = 10
skf = StratifiedKFold(n_splits=folds, shuffle = True, random_state = 1001)
```

Looking for best parametrs of the XGBClassifier

In [50]:

```
# Parameters for xgboost

params = {"min_child_weight"   : [1, 2, 3, 4],
          "gamma"              : [0.8, 0.9, 1, 1.1, 1.2],
          "subsample"          : [ 0.6, 0.7, 0.8, 0.9],
          "colsample_bytree"   : [0.3, 0.4, 0.5],
          "max_depth"          : [3, 4, 5, 6, 7, 8, 9]
        }

# Seaching through xgboost

folds = 10
param_comb = 10

random_search = RandomizedSearchCV(xgb,
                                   param_distributions = params,
                                   n_iter              = param_comb,
                                   scoring             = "f1",
                                   n_jobs              = -1,
                                   cv                  = skf.split(X,y),
                                   verbose             = 3,
                                   random_state        = 1001 )

# Fitting the randomized search to our dataset
random_search.fit(X, y)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  24 tasks      | elapsed:  3.9min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 16.1min finished
```

Out[50]:

```
RandomizedSearchCV(cv=<generator object _BaseKFold.split at 0x000002446E616D60>,
                   estimator=XGBClassifier(base_score=None, booster=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None, gamma=None,
                                           gpu_id=None, importance_type='gain',
                                           interaction_constraints=None,
                                           learning_rate=None,
                                           max_delta_step=None, max_depth=None,
                                           min_child_weight=None, mis...
```

```
                                    random_state=None, reg_alpha=None,
                                    reg_lambda=None,
                                    scale_pos_weight=None,
                                    subsample=None, tree_method=None,
                                    validate_parameters=None,
                                    verbosity=None),
                   n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.3, 0.4, 0.5],
                                        'gamma': [0.8, 0.9, 1, 1.1, 1.2],
                                        'max_depth': [3, 4, 5, 6, 7, 8, 9],
                                        'min_child_weight': [1, 2, 3, 4],
                                        'subsample': [0.6, 0.7, 0.8, 0.9]},
                   random_state=1001, scoring='f1', verbose=3)
```

In [52]:

```
# Viewing the best parameters for the XGBClassifier
random_search.best_params_
```

Out[52]:

```
{'subsample': 0.7,
 'min_child_weight': 3,
 'max_depth': 3,
 'gamma': 1.1,
 'colsample_bytree': 0.4}
```

Searching for best parameters for the CatBoostClassifier

In [37]:

```
# Parameters for catboost

params2 = {"iterations"                : [500, 600],
           "learning_rate"             : [0.02, 0.03, 0.04],
           "depth"                     : [4, 5, 6, 8, 9, 10],
           "loss_function"             : ["Logloss", "CrossEntropy"],
           "l2_leaf_reg"               : np.logspace(-20, -19, 3),
           "leaf_estimation_iterations" : [10],
           "eval_metric"               : ["Accuracy"],
#           "use_best_model"            : ['True'],
           "logging_level"             : ["Silent"],
           "random_seed"               : [42]
          }

# Searching through catboostclassifier

folds = 10
param_comb = 10

random_search2 = RandomizedSearchCV(cbc,
                                    param_distributions = params2,
                                    n_iter              = param_comb,
                                    scoring             = "f1",
                                    n_jobs              = -1,
                                    cv                  = skf.split(X,y),
                                    verbose             = 3,
                                    random_state        = 1001)

# Fitting the search to out data with nan
random_search2.fit(X, y)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  24 tasks      | elapsed: 23.2min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 97.1min finished
```

Out[37]:

```
RandomizedSearchCV(cv=<generator object _BaseKFold.split at 0x0000019A0D60EF90>,
                   estimator=<catboost.core.CatBoostClassifier object at 0x0000019A09C8FF10>,
```

```
                   n_jobs=-1,
                   param_distributions={'depth': [4, 5, 6, 8, 9, 10],
                                        'eval_metric': ['Accuracy'],
                                        'iterations': [500, 600],
                                        'l2_leaf_reg': array([1.00000000e-20, 3.16227766e-20, 1.000
0000e-19]),
                                        'leaf_estimation_iterations': [10],
                                        'learning_rate': [0.02, 0.03, 0.04],
                                        'logging_level': ['Silent'],
                                        'loss_function': ['Logloss',
                                                          'CrossEntropy'],
                                        'random_seed': [42]},
                   random_state=1001, scoring='f1', verbose=3)
```

In [38]:

```
# Viewing the best parameters from the catboost search
random_search2.best_params_
```

Out[38]:

```
{'random_seed': 42,
 'loss_function': 'CrossEntropy',
 'logging_level': 'Silent',
 'learning_rate': 0.03,
 'leaf_estimation_iterations': 10,
 'l2_leaf_reg': 1e-20,
 'iterations': 500,
 'eval_metric': 'Accuracy',
 'depth': 6}
```

Looking for best parameters for the RandomForestClassifier

In [33]:

```
# Parameters for random forest

params3 = {"bootstrap"         : [True, False],
           "max_depth"         : [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
           "max_features"      : ["auto", "sqrt"],
           "min_samples_leaf"  : [1, 2, 4],
           "min_samples_split" : [2, 5, 10],
           "n_estimators"      : [130, 180, 230],
           "criterion"         : ["gini", "entropy"] }

# Searching through random forest

folds = 10
param_comb = 10

random_search3 = RandomizedSearchCV(rfc,
                                    param_distributions = params3,
                                    n_iter              = param_comb,
                                    scoring             = 'f1',
                                    n_jobs              = -1,
                                    cv                  = skf.split(X,y),
                                    verbose             = 3,
                                    random_state        = 1001)

# Fitting our randomized search to the data
random_search3.fit(X_fill, y)

# Printing the best parameters from the search
random_search3.best_params_
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  24 tasks      | elapsed: 10.9min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 47.9min finished
```

Out[33]:

```
{'n_estimators': 130,
 'min_samples_split': 5,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 40,
 'criterion': 'gini',
 'bootstrap': False}
```

Looking for parameters of the LGBMClassifier

In [25]:

```
# Parameters for lgbmclassifier

from scipy.stats import randint as sp_randint
from scipy.stats import uniform as sp_uniform

params4 ={"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]}

# Searching through lgbmclassifier

folds = 10
param_comb = 10

random_search4 = RandomizedSearchCV(lgb,
                                    param_distributions = params4,
                                    n_iter              = param_comb,
                                    scoring             = "roc_auc",
                                    n_jobs              = -1,
                                    refit               = True,
                                    cv                  = skf.split(X,y),
                                    verbose             = 3,
                                    random_state        = 1001)

# Fitting the search to out data with nan
random_search4.fit(X, y)

# Pinting the best params from the search
print(random_search4.best_params_)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  24 tasks      | elapsed: 14.7min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 82.1min finished

{'colsample_bytree': 0.4165178996588055, 'min_child_samples': 297, 'min_child_weight': 1e-05, 'num
_leaves': 28, 'reg_alpha': 50, 'reg_lambda': 100, 'subsample': 0.7353436507509516}
```

In [26]:

```
random_search4.best_score_
```

Out[26]:

```
0.8393271505029833
```

In [ ]:

```

```

In [ ]:

Test Data

In [18]:

```
# importing our test data

test = pd.read_csv("Test.csv")
```

In [19]:

```
# Dropping the applicant id column
test.drop("Applicant_ID", axis = 1, inplace = True)
```

In [20]:

```
# Encoding form field 47
test["form_field47"] = encoder1.transform(test["form_field47"])
```

In [21]:

```
# filling the test data to be used for the random forest classifier
test_fill = test.fillna(-999)
```

In [22]:

```
# Scaling the test data containing missing values
test = scaler.transform(test)

# Scaling the test data with nan filled with -999
test_fill = scaler1.transform(test_fill)
```

Creating a new model and working with best parameters from the randomized search of each model

model.predict_proba returns two columns but we will be needing the secodn column, therefore, we will be adding [: ,1] to our predict_proba to extract the second column, we ca also use [: ,-1] to extract the last column which is the second column

XGBClassifier

In [32]:

```
# Creating a NEW XGBClassifier with the best parameters gotten from the random search
xgb1 = XGBClassifier(subsample        = 0.8,
                     min_child_weight = 4,
                     max_depth        = 4,
                     gamma            = 1.1,
                     colsample_bytree = 0.5
                     )

# Fitting the classifier to our target and feature columns
xgb1.fit(X, y)
```

Out[32]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.5, gamma=1.1, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=4,
              min_child_weight=4, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=0.8,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [33]:

```
# Making predictions with the new XGBClassifier
xgbp = xgb1.predict_proba(test)[:,1]
```

CatBoostClassifier

In [34]:

```
# Creating a NEW CatBoostClassifier with the best parameters gotten from the random search
cbc1 = CatBoostClassifier(random_seed                = 42,
                          loss_function              = "CrossEntropy",
                          logging_level              = "Silent",
                          learning_rate              = 0.03,
                          leaf_estimation_iterations = 10,
                          l2_leaf_reg                = 1e-20,
                          iterations                 = 500,
                          eval_metric                = "Accuracy",
                          depth                      = 6
                          )

# Fitting the classifier to our target and feature columns
cbc1.fit(X,y)
```

Out[34]:

```
<catboost.core.CatBoostClassifier at 0x2098e46ae50>
```

In [35]:

```
# Making predictions with the new CatBoostClassifier
cbcp = cbc1.predict_proba(test)[:,1]
```

RandomForestClassifier

In [36]:

```
# Creating a NEW RandomForestClassifier with the best parameters gotten from the random search
rfc1 = RandomForestClassifier(n_estimators      = 130,
                              min_samples_split = 5,
                              min_samples_leaf  = 1,
                              max_features      = "auto",
                              max_depth         = 40,
                              criterion         = "gini",
                              bootstrap         = False)
# Fitting the classifier to our target and feature columns
rfc1.fit(X_fill, y)
```

Out[36]:

```
RandomForestClassifier(bootstrap=False, max_depth=40, min_samples_split=5,
                       n_estimators=130)
```

In [37]:

```
# making predictions with the new RandomForestClassifier
rfcp = rfc1.predict_proba(test_fill)[:,1]
```

LGBMClassifier

In [39]:

```
lgb1 = LGBMClassifier(colsample_bytree = 0.4165178996588055,
                      min_child_samples = 297,
                      min_child_weight = 1e-05,
                      num_leaves = 28,
```

```
                        reg_alpha = 50,
                        reg_lambda = 100,
                        subsample = 0.7353436507509516,
                        max_depth=-1,
                        random_state=314,
                        silent=True,
                        metric='None',
                        n_estimators=5000,
                        n_jobs= -4
                       )

# Fitting the classifier to the target anf feature column
lgb1.fit(X, y)
```

Out[39]:

```
LGBMClassifier(colsample_bytree=0.4165178996588055, metric='None',
               min_child_samples=297, min_child_weight=1e-05, n_estimators=5000,
               n_jobs=-4, num_leaves=28, random_state=314, reg_alpha=50,
               reg_lambda=100, subsample=0.7353436507509516)
```

In [40]:

```
# Making predictions with our classifier
lgbp = lgb1.predict_proba(test)[:,1]
```

In [ ]:

```

```

Prediction Table

In [48]:

```
mixed = pd.DataFrame({# "xgboost"  : xgbp,
                      "catboost" : cbcp,
                      "forest"   : rfcp,
                      "Lightgbm" : lgbp
                     })
mixed.head()
```

Out[48]:

	catboost	forest	Lightgbm
0	0.304480	0.320513	0.263665
1	0.427942	0.307051	0.392060
2	0.358523	0.387308	0.446228
3	0.775943	0.714744	0.836372
4	0.173978	0.196154	0.138847

In [49]:

```
mixed = mixed.mean(axis = 1)
```

In [50]:

```
mixed
```

Out[50]:

```
0         0.296219
1         0.375684
2         0.397353
3         0.775686
4         0.169660
            ...
```

```
23995    0.653132
23996    0.245827
23997    0.274194
23998    0.557089
23999    0.230929
Length: 24000, dtype: float64
```

Submission

In [51]:

```
samp = pd.read_csv("SampleSubmission.csv")
```

In [52]:

```
samp["default_status"] = mixed
```

In [53]:

```
samp.head()
```

Out[53]:

	Applicant_ID	default_status
0	Apcnt_1000032	0.296219
1	Apcnt_1000048	0.375684
2	Apcnt_1000052	0.397353
3	Apcnt_1000076	0.775686
4	Apcnt_1000080	0.169660

In [54]:

```
samp.to_csv("the_submit23.csv", index = False)
```

In [ ]:

```

```

In [ ]:

```

```