In [4]:
```
!ls Machine-Learning--Projects/Projects/'Projects for Submission'/'Project 2 - Income Qualification'
```

```
'Dataset for the project.zip'  'Income Qualification.txt'
```

In [5]:
```
!unzip Machine-Learning--Projects/Projects/'Projects for Submission'/'Project 2 - Income Qualification'/'Dataset for the project.zip'
```

```
Archive:  Machine-Learning--Projects/Projects/Projects for Submission/Project 2 - Income Qualification/Dataset for the project.zip
   creating: Dataset for the project/
  inflating: Dataset for the project/test.csv
  inflating: Dataset for the project/train.csv
```

In [8]:
```
!mv 'Dataset for the project' data
!ls data
```

```
test.csv  train.csv
```

In [10]:
```
!pip install catboost category_encoders
```

```
Collecting catboost
  Downloading https://files.pythonhosted.org/packages/5a/8a/a867c35770291646b085e9248814eb32dbe2aa824715b08e40cd92d0a83e/catboost-0.15.1-cp36-none-manylinux1_x86_64.whl (61.0MB)
     |████████████████████████████████| 61.1MB 433kB/s
Collecting category_encoders
  Downloading https://files.pythonhosted.org/packages/6e/a1/f7a22f144f33be78afeb06bfa78478e8284a64263a3c09b1ef54e673841e/category_encoders-2.0.0-py2.py3-none-any.whl (87kB)
     |████████████████████████████████| 92kB 27.1MB/s
Requirement already satisfied: graphviz in /usr/local/lib/python3.6/dist-packages (from catboost) (0.10.1)
Requirement already satisfied: pandas>=0.19.1 in /usr/local/lib/python3.6/dist-packages (from catboost) (0.24.2)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from catboost) (1.12.0)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.6/dist-packages (from catboost) (1.16.4)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.6/dist-packages (from category_encoders) (0.21.2)
Requirement already satisfied: statsmodels>=0.6.1 in /usr/local/lib/python3.6/dist-packages (from category_encoders) (0.9.0)
Requirement already satisfied: scipy>=0.19.0 in /usr/local/lib/python3.6/dist-packages (from category_encoders) (1.3.0)
Requirement already satisfied: patsy>=0.4.1 in /usr/local/lib/python3.6/dist-packages (from category_encoders) (0.5.1)
Requirement already satisfied: pytz>=2011k in /usr/local/lib/python3.6/dist-packages (from pandas>=0.19.1->catboost) (2018.9)
Requirement already satisfied: python-dateutil>=2.5.0 in /usr/local/lib/python3.6/dist-packages (from pandas>=0.19.1->catboost) (2.5.3)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages (from scikit-learn>=0.20.0->category_encoders) (0.13.2)
Installing collected packages: catboost, category-encoders
Successfully installed catboost-0.15.1 category-encoders-2.0.0
```

In [11]:
```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import datetime
import gc
import numpy as np
import os
import pandas as pd
from tqdm import tqdm
import warnings
warnings.filterwarnings(action='ignore',category = DeprecationWarning)
warnings.simplefilter(action='ignore',category = DeprecationWarning)


from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import f1_score
from sklearn.model_selection import KFold, RepeatedKFold, GroupKFold
from sklearn.utils.class_weight import compute_sample_weight
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import ADASYN
import category_encoders as ce
import lightgbm as lgb
from xgboost import XGBClassifier
from catboost import CatBoostClassifier

from time import time

import scipy.stats as st
from sklearn.pipeline import Pipeline
from tempfile import mkdtemp
from shutil import rmtree

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifie
r
import xgboost as xgb

from sklearn.metrics import confusion_matrix, accuracy_score, f1_score

from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.model_selection import RandomizedSearchCV,GridSearchCV
from sklearn.model_selection import train_test_split

import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline


import os
print(os.listdir("data"))

import warnings

def fxn():
    warnings.warn("deprecated", DeprecationWarning)

with warnings.catch_warnings(record=True) as w:
    # Cause all warnings to always be triggered.
    warnings.simplefilter("always")
    # Trigger a warning.
    fxn()
    # Verify some things
    assert len(w) == 1
    assert issubclass(w[-1].category, DeprecationWarning)
    assert "deprecated" in str(w[-1].message)
```

```
['train.csv', 'test.csv']
```

In [0]:
```python
def dprint(*args, **kwargs):
    print("[{}] ".format(datetime.datetime.now().strftime("%Y-%m-%d %H:%M")) + \
        " ".join(map(str,args)), **kwargs)

id_name = 'Id'
target_name = 'Target'
```

In [0]:
```python
# Load data
train = pd.read_csv('data/train.csv')
test = pd.read_csv('data/test.csv')
```

In [14]:
```python
train['is_test'] = 0
test['is_test'] = 1
df_all = pd.concat([train, test], axis=0)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: FutureWarnin
g: Sorting because non-concatenation axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  This is separate from the ipykernel package so we can avoid doing imports u
ntil
```

In [15]:
```python
dprint('Clean features...')
cols = ['dependency']
for c in tqdm(cols):
    x = df_all[c].values
    strs = []
    for i, v in enumerate(x):
        try:
            val = float(v)
        except:
            strs.append(v)
            val = np.nan
        x[i] = val
    strs = np.unique(strs)

    for s in strs:
        df_all[c + '_' + s] = df_all[c].apply(lambda x: 1 if x == s else 0)

    df_all[c] = x
    df_all[c] = df_all[c].astype(float)
dprint("Done.")
```

```
100%|██████████| 1/1 [00:00<00:00, 14.06it/s]

[2019-06-05 11:08] Clean features...
[2019-06-05 11:08] Done.
```

In [16]:

```
dprint("Extracting features...")
def extract_features(df):
    df['bedrooms_to_rooms'] = df['bedrooms']/df['rooms']
    df['rent_to_rooms'] = df['v2a1']/df['rooms']
    df['rent_to_bedrooms'] = df['v2a1']/df['bedrooms']
    df['tamhog_to_rooms'] = df['tamhog']/df['rooms'] # tamhog - size of the ho
usehold
    df['tamhog_to_bedrooms'] = df['tamhog']/df['bedrooms']
    df['r4t3_to_tamhog'] = df['r4t3']/df['tamhog'] # r4t3 - Total persons in t
he household
    df['r4t3_to_rooms'] = df['r4t3']/df['rooms'] # r4t3 - Total persons in the
household
    df['r4t3_to_bedrooms'] = df['r4t3']/df['bedrooms']
    df['rent_to_r4t3'] = df['v2a1']/df['r4t3']
    df['v2a1_to_r4t3'] = df['v2a1']/(df['r4t3'] - df['r4t1'])
    df['hhsize_to_rooms'] = df['hhsize']/df['rooms']
    df['hhsize_to_bedrooms'] = df['hhsize']/df['bedrooms']
    df['rent_to_hhsize'] = df['v2a1']/df['hhsize']
    df['qmobilephone_to_r4t3'] = df['qmobilephone']/df['r4t3']
    df['qmobilephone_to_v18q1'] = df['qmobilephone']/df['v18q1']


extract_features(train)
extract_features(test)
dprint("Done.")
```

```
[2019-06-05 11:08] Extracting features...
[2019-06-05 11:08] Done.
```

In [0]:

```
from sklearn.preprocessing import LabelEncoder

def encode_data(df):

    yes_no_map = {'no': 0, 'yes': 1}

    df['dependency'] = df['dependency'].replace(yes_no_map).astype(np.float32)

    df['edjefe'] = df['edjefe'].replace(yes_no_map).astype(np.float32)
    df['edjefa'] = df['edjefa'].replace(yes_no_map).astype(np.float32)

    df['idhogar'] = LabelEncoder().fit_transform(df['idhogar'])
```

In [18]:

```
dprint("Encoding Data....")
encode_data(train)
encode_data(test)
dprint("Done...")
```

```
[2019-06-05 11:08] Encoding Data....
[2019-06-05 11:08] Done...
```

In [0]:
```python
def do_features(df):
    feats_div = [('children_fraction', 'r4t1', 'r4t3'),
                 ('working_man_fraction', 'r4h2', 'r4t3'),
                 ('all_man_fraction', 'r4h3', 'r4t3'),
                 ('human_density', 'tamviv', 'rooms'),
                 ('human_bed_density', 'tamviv', 'bedrooms'),
                 ('rent_per_person', 'v2a1', 'r4t3'),
                 ('rent_per_room', 'v2a1', 'rooms'),
                 ('mobile_density', 'qmobilephone', 'r4t3'),
                 ('tablet_density', 'v18q1', 'r4t3'),
                 ('mobile_adult_density', 'qmobilephone', 'r4t2'),
                 ('tablet_adult_density', 'v18q1', 'r4t2'),
                 #('', '', ''),
                ]

    feats_sub = [('people_not_living', 'tamhog', 'tamviv'),
                 ('people_weird_stat', 'tamhog', 'r4t3')]

    for f_new, f1, f2 in feats_div:
        df['fe_' + f_new] = (df[f1] / df[f2]).astype(np.float32)
    for f_new, f1, f2 in feats_sub:
        df['fe_' + f_new] = (df[f1] - df[f2]).astype(np.float32)

    # aggregation rules over household
    aggs_num = {'age': ['min', 'max', 'mean'],
                'escolari': ['min', 'max', 'mean']
               }
    aggs_cat = {'dis': ['mean']}
    for s_ in ['estadocivil', 'parentesco', 'instlevel']:
        for f_ in [f_ for f_ in df.columns if f_.startswith(s_)]:
            aggs_cat[f_] = ['mean', 'count']
    # aggregation over household
    for name_, df_ in [('18', df.query('age >= 18'))]:
        df_agg = df_.groupby('idhogar').agg({**aggs_num, **aggs_cat}).astype(n
p.float32)
        df_agg.columns = pd.Index(['agg' + name_ + '_' + e[0] + "_" + e[1].upp
er() for e in df_agg.columns.tolist()])
        df = df.join(df_agg, how='left', on='idhogar')
        del df_agg
    # do something advanced above...

    # Drop SQB variables, as they are just squres of other vars
    df.drop([f_ for f_ in df.columns if f_.startswith('SQB') or f_ == 'agesq'
], axis=1, inplace=True)
    # Drop id's
    df.drop(['Id', 'idhogar'], axis=1, inplace=True)
    # Drop repeated columns
    df.drop(['hhsize', 'female', 'area2'], axis=1, inplace=True)
    return df
```

In [20]:
```python
dprint("Do_feature Engineering....")
train = do_features(train)
test = do_features(test)
dprint("Done....")
```

```
[2019-06-05 11:08] Do_feature Engineering....
[2019-06-05 11:08] Done....
```

In [21]:
```
dprint("Fill Na value....")
train = train.fillna(0)
test = test.fillna(0)
dprint("Done....")
```

```
[2019-06-05 11:08] Fill Na value....
[2019-06-05 11:08] Done....
```

In [22]:
```
train.shape,test.shape
```

Out[22]:
```
((9557, 221), (23856, 220))
```

In [0]:
```
cols_to_drop = [
    id_name,
    target_name,
]
X = train.drop(cols_to_drop, axis=1, errors='ignore')
y = train[target_name].values
```

In [24]:
```
X.shape,y.shape
```

Out[24]:
```
((9557, 220), (9557,))
```

In [0]:
```
params = {
        'min_child_weight': [1, 5, 10],
        'gamma': [0.5, 1, 1.5, 2, 5],
        'subsample': [0.6, 0.8, 1.0],
        'colsample_bytree': [0.6, 0.8, 1.0],
        'max_depth': [3, 4, 5]
        }
xgb = XGBClassifier(learning_rate=0.02, n_estimators=100, objective='multi:sof
tmax',booster='gbtree',
                    silent=True, nthread=1)

folds = 3
param_comb = 5
skf = StratifiedKFold(n_splits=folds, shuffle = True, random_state = 42)
```

In [26]:
```
random_search = RandomizedSearchCV(xgb, param_distributions=params, n_iter=par
am_comb, scoring='accuracy', n_jobs=4, cv=skf.split(X,y), verbose=0, random_st
ate=1001 )
random_search.fit(X, y)
```

Out[26]:
```
RandomizedSearchCV(cv=<generator object _BaseKFold.split at 0x7fcb62263f68>,
                   error_score='raise-deprecating',
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.02, max_delta_step
=0,

                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=1,
                                           objectiv...
                                           reg_lambda=1, scale_pos_weight=1,
                                           seed=None, silent=True, subsample=
1,

                                           verbosity=1),
                   iid='warn', n_iter=5, n_jobs=4,
                   param_distributions={'colsample_bytree': [0.6, 0.8, 1.0],
                                        'gamma': [0.5, 1, 1.5, 2, 5],
                                        'max_depth': [3, 4, 5],
                                        'min_child_weight': [1, 5, 10],
                                        'subsample': [0.6, 0.8, 1.0]},
                   pre_dispatch='2*n_jobs', random_state=1001, refit=True,
                   return_train_score=False, scoring='accuracy', verbose=0)
```

In [27]:
```
print('\n All results:')
print(random_search.cv_results_)
print('\n Best estimator:')
print(random_search.best_estimator_)
print('\n Best normalized gini score for %d-fold search with %d parameter comb
inations:' % (folds, param_comb))
print(random_search.best_score_ * 2 - 1)
print('\n Best hyperparameters:')
print(random_search.best_params_)
results = pd.DataFrame(random_search.cv_results_)
results.to_csv('xgb-random-grid-search-results-01.csv', index=False)
```

```
 All results:
{'mean_fit_time': array([38.54292218, 64.78864948, 58.65247647, 40.0707616 ,
45.34571632]), 'std_fit_time': array([0.23574511, 0.02759083, 0.13724321, 0.0
6338125, 4.70875725]), 'mean_score_time': array([0.18842101, 0.27431075, 0.26
77687 , 0.26250656, 0.14133581]), 'std_score_time': array([0.00767652, 0.0041
8096, 0.00247599, 0.00629999, 0.05404488]), 'param_subsample': masked_array(d
ata=[1.0, 0.6, 0.8, 1.0, 0.8],
             mask=[False, False, False, False, False],
       fill_value='?',
            dtype=object), 'param_min_child_weight': masked_array(data=[5, 1,
5, 5, 1],
             mask=[False, False, False, False, False],
       fill_value='?',
            dtype=object), 'param_max_depth': masked_array(data=[3, 5, 5, 5,
4],
             mask=[False, False, False, False, False],
       fill_value='?',
            dtype=object), 'param_gamma': masked_array(data=[5, 1.5, 1, 5,
1],
             mask=[False, False, False, False, False],
       fill_value='?',
            dtype=object), 'param_colsample_bytree': masked_array(data=[1.0,
0.8, 0.8, 0.6, 1.0],
             mask=[False, False, False, False, False],
       fill_value='?',
            dtype=object), 'params': [{'subsample': 1.0, 'min_child_weight':
5, 'max_depth': 3, 'gamma': 5, 'colsample_bytree': 1.0}, {'subsample': 0.6,
'min_child_weight': 1, 'max_depth': 5, 'gamma': 1.5, 'colsample_bytree': 0.
8}, {'subsample': 0.8, 'min_child_weight': 5, 'max_depth': 5, 'gamma': 1, 'co
lsample_bytree': 0.8}, {'subsample': 1.0, 'min_child_weight': 5, 'max_depth':
5, 'gamma': 5, 'colsample_bytree': 0.6}, {'subsample': 0.8, 'min_child_weigh
t': 1, 'max_depth': 4, 'gamma': 1, 'colsample_bytree': 1.0}], 'split0_test_sc
ore': array([0.67524318, 0.74929401, 0.74144964, 0.72921243, 0.70442422]), 's
plit1_test_score': array([0.69020716, 0.74795982, 0.74199623, 0.73697426, 0.7
2473321]), 'split2_test_score': array([0.69723618, 0.75659548, 0.75125628, 0.
74340452, 0.72738693]), 'mean_test_score': array([0.68755886, 0.75128178, 0.7
4489903, 0.7365282 , 0.71884483]), 'std_test_score': array([0.00917161, 0.003
79517, 0.00449903, 0.00580233, 0.01025747]), 'rank_test_score': array([5, 1,
2, 3, 4], dtype=int32)}

 Best estimator:
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.8, gamma=1.5,
              learning_rate=0.02, max_delta_step=0, max_depth=5,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=1, objective='multi:softprob', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=True, subsample=0.6, verbosity=1)

 Best normalized gini score for 3-fold search with 5 parameter combinations:
0.5025635659725856

 Best hyperparameters:
{'subsample': 0.6, 'min_child_weight': 1, 'max_depth': 5, 'gamma': 1.5, 'cols
ample_bytree': 0.8}
```

In [31]:
```
clf = RandomForestClassifier(n_estimators=120, max_features="sqrt", min_sample
s_leaf=3, n_jobs=-1, class_weight='balanced_subsample')
params={'n_estimators': list(range(40,61, 1))}
gs = GridSearchCV(clf, params, cv=5)
gs.fit(X,y)
```

Out[31]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True,
                                              class_weight='balanced_subsampl
e',
                                              criterion='gini', max_depth=Non
e,
                                              max_features='sqrt',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=3,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=120, n_jobs=-1,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'n_estimators': [40, 41, 42, 43, 44, 45, 46, 47, 48,
                                          49, 50, 51, 52, 53, 54, 55, 56, 57,
                                          58, 59, 60]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [0]:
```
preds=gs.predict(X)
```

In [34]:
```
from sklearn.metrics import classification_report
print(classification_report(y, preds))
```

```
              precision    recall  f1-score   support

           1       0.98      0.97      0.98       755
           2       0.98      0.97      0.97      1597
           3       0.95      0.99      0.97      1209
           4       1.00      0.99      1.00      5996

    accuracy                           0.99      9557
   macro avg       0.97      0.98      0.98      9557
weighted avg       0.99      0.99      0.99      9557
```

In [36]:
```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y, preds))
```

```
[[ 736   18    0    1]
 [  16 1544   33    4]
 [   0    6 1197    6]
 [   2    9   35 5950]]
```

In [37]:
```
print(gs.best_params_)
print(gs.best_score_)
print(gs.best_estimator_)
```

```
{'n_estimators': 54}
0.6302186878727635
RandomForestClassifier(bootstrap=True, class_weight='balanced_subsample',
                       criterion='gini', max_depth=None, max_features='sqrt',
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=3,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       n_estimators=54, n_jobs=-1, oob_score=False,
                       random_state=None, verbose=0, warm_start=False)
```

In [38]:
```
cvres = gs.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(mean_score), params)
```

```
0.7875775073095224 {'n_estimators': 40}
0.7901639757081144 {'n_estimators': 41}
0.7882415134196364 {'n_estimators': 42}
0.7910242524742912 {'n_estimators': 43}
0.7935994874554066 {'n_estimators': 44}
0.7865804471461157 {'n_estimators': 45}
0.7855821215158134 {'n_estimators': 46}
0.7884406062066681 {'n_estimators': 47}
0.7916853677192155 {'n_estimators': 48}
0.7898328505308921 {'n_estimators': 49}
0.7883078832689032 {'n_estimators': 50}
0.7904287759755035 {'n_estimators': 51}
0.7885733068061447 {'n_estimators': 52}
0.7897003615841358 {'n_estimators': 53}
0.7938631417774499 {'n_estimators': 54}
0.7877767679091204 {'n_estimators': 55}
0.7902963869324554 {'n_estimators': 56}
0.7839821562673545 {'n_estimators': 57}
0.7937972364067184 {'n_estimators': 58}
0.7899653172572795 {'n_estimators': 59}
0.7935335601873839 {'n_estimators': 60}
```

In [0]:

In [0]: