---Binary Classification problem for the ICR-competition in august 2023---

Placement: Top 23% | 1442/6430

Author: Jakob Lindstrøm

Date: 11.08.2023

Keywords:

Kaggle, ML, competition, bio-data

Data sources:

KAGGLE: https://www.kaggle.com/competitions/icr-identify-age-related-conditions

Intention:

This code was written with the intention of getting placing as high as possible in the ICR - Kaggle competition. This notebook was submitted to the competiton and its performance gave was enough to catch a 1442th place in the competition. The machine learning problem was binary classification. The procedure of creating the model demanded various use of machine learning techniques: such as exploratory data analysis, feature engineering and modelling. I ended up using an RandomForestClassifier-model.

Table of content

Download datasets

The notebook was create in a Kaggel enviroment and therefore the datasets was downloaded using the Kaggle built-in downloader.

In [1]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets pr
# You can also write temporary files to /kaggle/temp/, but they won't be saved outs
```

```
/kaggle/input/icr-identify-age-related-conditions/sample_submission.csv
/kaggle/input/icr-identify-age-related-conditions/greeks.csv
/kaggle/input/icr-identify-age-related-conditions/train.csv
/kaggle/input/icr-identify-age-related-conditions/test.csv
```

Importation of libraries

The libraries below have been used in this notebook

In [2]:
```
import random as rd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import mutual_info_classif

from sklearn.metrics import accuracy_score
from sklearn.metrics import  log_loss

from sklearn.impute import SimpleImputer
from  sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import LabelEncoder


from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.callbacks import EarlyStopping

import warnings
warnings.filterwarnings("ignore")
print('Libraries imported')
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumP
y version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected ver
sion 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/__init__.py:98: U
serWarning: unable to load libtensorflow_io_plugins.so: unable to open file: libte
nsorflow_io_plugins.so, from paths: ['/opt/conda/lib/python3.10/site-packages/tens
orflow_io/python/ops/libtensorflow_io_plugins.so']
caused by: ['/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/libt
ensorflow_io_plugins.so: undefined symbol: _ZN3tsl6StatusC1EN10tensorflow5error4Co
deESt17basic_string_viewIcSt11char_traitsIcEENS_14SourceLocationE']
  warnings.warn(f"unable to load libtensorflow_io_plugins.so: {e}")
/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/__init__.py:104:
UserWarning: file system plugins are not loaded: unable to open file: libtensorflo
w_io.so, from paths: ['/opt/conda/lib/python3.10/site-packages/tensorflow_io/pytho
n/ops/libtensorflow_io.so']
caused by: ['/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/libt
ensorflow_io.so: undefined symbol: _ZTVN10tensorflow13GcsFileSystemE']
  warnings.warn(f"file system plugins are not loaded: {e}")
Libraries imported
```

Target analysis

Below I look at the shape and columns of the dataframe, and the distribution of the target column. I have gotten a lot of inspiration from other peoples code through the competitions code section: https://www.kaggle.com/competitions/icr-identify-age-related-conditions/code. I will do further exploratory analysis in regards to the distributions of the features.

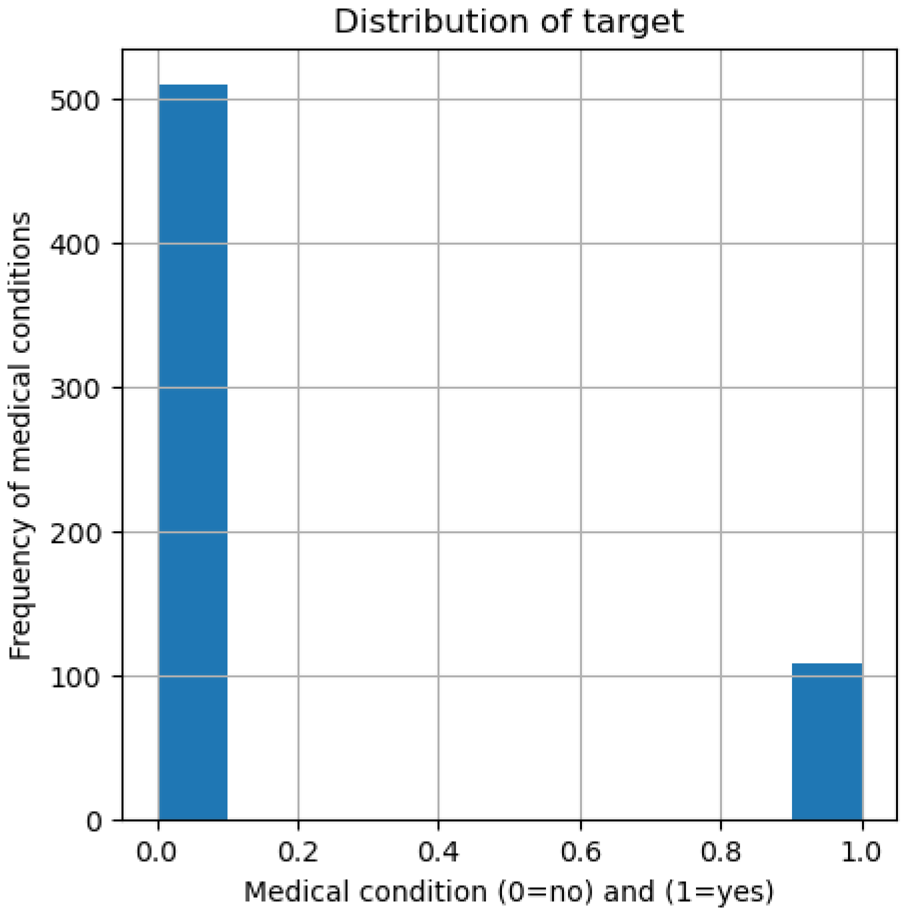
In [3]:

```
df = pd.read_csv("/kaggle/input/icr-identify-age-related-conditions/train.csv")
print('Shape of dataframe: ',df.shape)
print()
print('Columns in dataframe:\n',df.columns)
```

```
Shape of dataframe:  (617, 58)

Columns in dataframe:
 Index(['Id', 'AB', 'AF', 'AH', 'AM', 'AR', 'AX', 'AY', 'AZ', 'BC', 'BD ', 'BN',
       'BP', 'BQ', 'BR', 'BZ', 'CB', 'CC', 'CD ', 'CF', 'CH', 'CL', 'CR', 'CS',
       'CU', 'CW ', 'DA', 'DE', 'DF', 'DH', 'DI', 'DL', 'DN', 'DU', 'DV', 'DY',
       'EB', 'EE', 'EG', 'EH', 'EJ', 'EL', 'EP', 'EU', 'FC', 'FD ', 'FE', 'FI',
       'FL', 'FR', 'FS', 'GB', 'GE', 'GF', 'GH', 'GI', 'GL', 'Class'],
      dtype='object')
```

In [4]:

```
plt.figure(figsize=(5,5))
df['Class'].hist()
plt.xlabel('Medical condition (0=no) and (1=yes)')
plt.ylabel('Frequency of medical conditions')
plt.title('Distribution of target')
plt.show()
```



At the histogram above, one can observe that the target variable is quite imbalanced, this should be taken into account later on.

Feature engineering

My approach of handling data will be by creating functions. Functions will the same input and output will give me the ability of easily checking whther the feature engineering or the machine learning technique will help me generating a better score on the competiton leaderboard.

Now I will start with the feature engineering. First of all I need to handle potential categorical-variables, such variables will be handled by use of onehot encoding.

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 617 entries, 0 to 616
Data columns (total 58 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Id      617 non-null    object 
 1   AB      617 non-null    float64
 2   AF      617 non-null    float64
 3   AH      617 non-null    float64
 4   AM      617 non-null    float64
 5   AR      617 non-null    float64
 6   AX      617 non-null    float64
 7   AY      617 non-null    float64
 8   AZ      617 non-null    float64
 9   BC      617 non-null    float64
 10  BD      617 non-null    float64
 11  BN      617 non-null    float64
 12  BP      617 non-null    float64
 13  BQ      557 non-null    float64
 14  BR      617 non-null    float64
 15  BZ      617 non-null    float64
 16  CB      615 non-null    float64
 17  CC      614 non-null    float64
 18  CD      617 non-null    float64
 19  CF      617 non-null    float64
 20  CH      617 non-null    float64
 21  CL      617 non-null    float64
 22  CR      617 non-null    float64
 23  CS      617 non-null    float64
 24  CU      617 non-null    float64
 25  CW      617 non-null    float64
 26  DA      617 non-null    float64
 27  DE      617 non-null    float64
 28  DF      617 non-null    float64
 29  DH      617 non-null    float64
 30  DI      617 non-null    float64
 31  DL      617 non-null    float64
 32  DN      617 non-null    float64
 33  DU      616 non-null    float64
 34  DV      617 non-null    float64
 35  DY      617 non-null    float64
 36  EB      617 non-null    float64
 37  EE      617 non-null    float64
 38  EG      617 non-null    float64
 39  EH      617 non-null    float64
 40  EJ      617 non-null    object 
 41  EL      557 non-null    float64
 42  EP      617 non-null    float64
 43  EU      617 non-null    float64
 44  FC      616 non-null    float64
 45  FD      617 non-null    float64
 46  FE      617 non-null    float64
 47  FI      617 non-null    float64
 48  FL      616 non-null    float64
 49  FR      617 non-null    float64
 50  FS      615 non-null    float64
 51  GB      617 non-null    float64
 52  GE      617 non-null    float64
 53  GF      617 non-null    float64
 54  GH      617 non-null    float64
 55  GI      617 non-null    float64
 56  GL      616 non-null    float64
 57  Class   617 non-null    int64  
```

```
dtypes: float64(55), int64(1), object(2)
memory usage: 279.7+ KB
```

In [6]:

```
# onehot encoding of categorical variables
def onehot(data):
    train = data[0]
    test = data[1]

    cols = train.dtypes == 'object'
    object_cols = list(cols[cols].index)

    OH_encoder = OneHotEncoder(handle_unknown='ignore',sparse=False)
    encoded_train = pd.DataFrame(OH_encoder.fit_transform(train[object_cols]))
    encoded_test = pd.DataFrame(OH_encoder.transform(test[object_cols]))

    encoded_train.index = train.index
    encoded_test.index = test.index
    train = train.drop(object_cols,axis=1)
    test = test.drop(object_cols,axis=1)

    OHE_train = pd.concat([train,encoded_train],axis=1)
    OHE_test = pd.concat([test,encoded_test],axis=1)
    OHE_train.columns = OHE_train.columns.astype(str)
    OHE_test.columns = OHE_test.columns.astype(str)

    return [OHE_train,OHE_test]
```

After handling categorical variables, I need to take a decision about how I should handle missing values. I have choosen to impute nan's with a mean-strategy and creating a extra binary column that tells whether the observation had a missing value.

In [7]:

```
df.isnull().sum()
```

Out[7]:

```
Id        0
AB        0
AF        0
AH        0
AM        0
AR        0
AX        0
AY        0
AZ        0
BC        0
BD        0
BN        0
BP        0
BQ       60
BR        0
BZ        0
CB        2
CC        3
CD        0
CF        0
CH        0
CL        0
CR        0
CS        0
CU        0
CW        0
DA        0
DE        0
DF        0
DH        0
DI        0
DL        0
DN        0
DU        1
DV        0
DY        0
EB        0
EE        0
EG        0
EH        0
EJ        0
EL       60
EP        0
EU        0
FC        1
FD        0
FE        0
FI        0
FL        1
FR        0
FS        2
GB        0
GE        0
GF        0
GH        0
GI        0
GL        1
Class     0
dtype: int64
```

In [8]:

```
def nan_ext(data):
    train = data[0]
    test = data[1]
```

```
    nan_cols = [col for col in train.columns if train[col].isnull().any()]
    for cols in nan_cols:
        train[cols+'_nan'] = train[cols].isnull()
        test[cols+'_nan'] = test[cols].isnull()

    mean_imputer = SimpleImputer()
    imputed_train = pd.DataFrame(mean_imputer.fit_transform(train))
    imputed_test = pd.DataFrame(mean_imputer.transform(test))


    imputed_train.columns = train.columns
    imputed_test.columns = test.columns

    data = [imputed_train,imputed_test]
    return data
```

Now I will create a function that can help me with choosing the most valuable features, this will be done by use of mutual information.

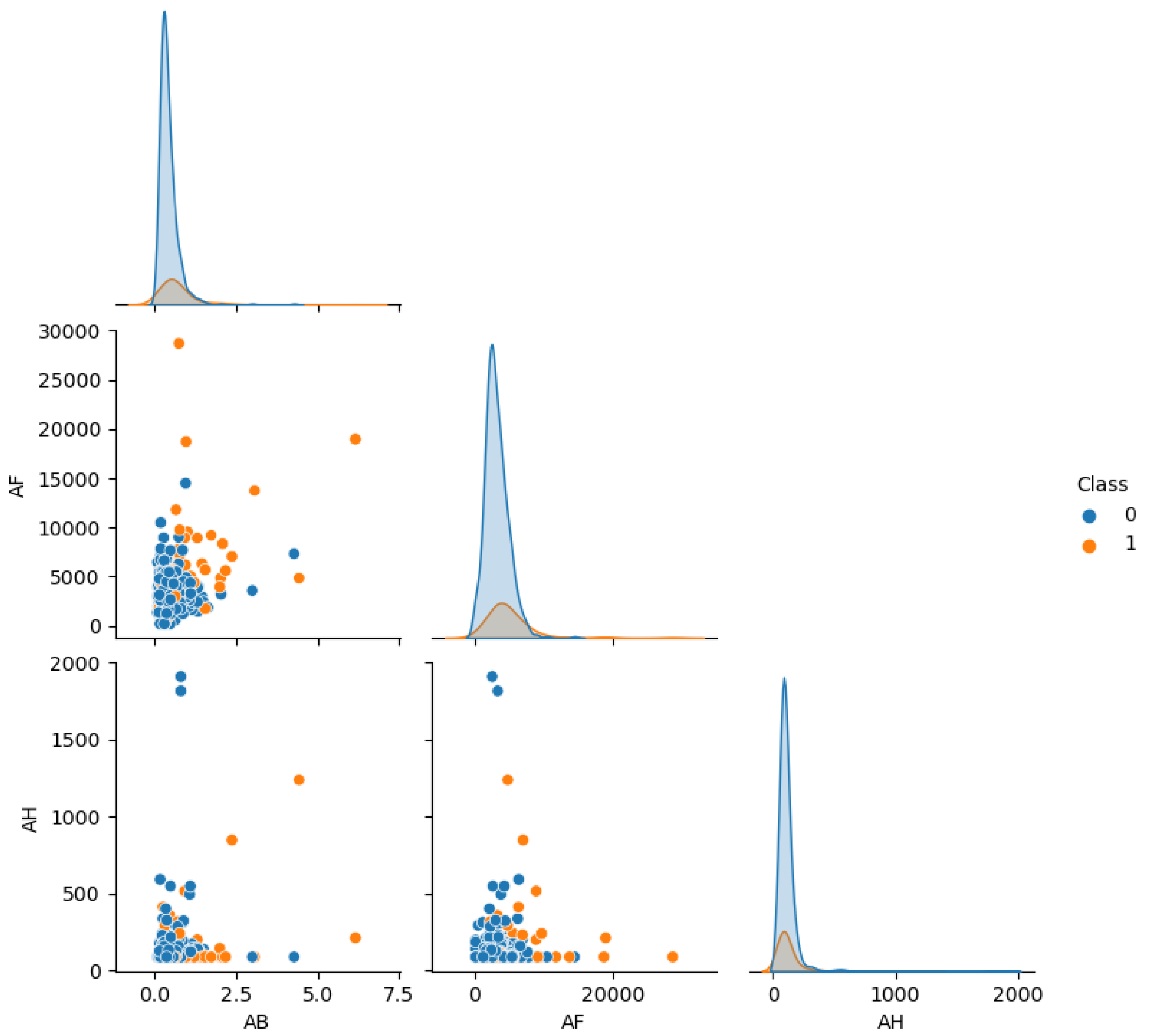
In [9]:
```
def mutual_info(df):
    mi_scores = mutual_info_classif(df[X], df[y] ,discrete_features=False, random_s
    mi_scores = pd.Series(mi_scores, index=df[X].columns)
    mi_scores = mi_scores.sort_values(ascending=False)
    return  mi_scores
```

The code takes generates pairplots, that can help in getting valuebale insights regarding the distributions of the variables in regard to the target variable.

In [10]:
```
def plot_vars(data, start, end):

    y = data['Class']

    X = data.drop('Class',axis=1).drop('Id', axis=1)
    plotdata = pd.concat([X.iloc[:,start:end],y], axis=1)

    return sns.pairplot(plotdata, hue="Class", corner=True)
```

In [11]:
```
example = plot_vars(df,0,3)
example
```

Out[11]:
```
<seaborn.axisgrid.PairGrid at 0x7f868d9109d0>
```



Based on running the pairplot function for many different variables I created a new function that can handle variables that have two distributions within itself.

In [12]:
```
def binary(data,threshhold='',name=''):
    train =  data[0]
    test = data[1]
    train[name+'_bin'] = np.where(train[name]>=threshhold,0,1)
    test[name+'_bin'] = np.where(test[name]>=threshhold,0,1)

    return[train,test]
```

Furthermore I handle potential outliers by setting all outlier-values to 1.49* IQR. TO next time I will add a new binary column that can tell whther such a process has been done on a observation or not.

In [13]:
```
def outliers(data):
    train = data[0]
    test = data[1]

    X = data[0].columns
    for i in range(0,len(X),1):
        name = X[i]

        #All outliers seems to be  high, no low outliers
        IQR = (np.percentile(train[name],75)-np.percentile(train[name],25))
        above_threshhold = np.percentile(IQR,50) + IQR * 1.5
```

```
        train[name] = np.where(train[name] >= above_threshhold, above_threshhold*1.
        test[name] = np.where(test[name] >= above_threshhold, above_threshhold*1.2,

    return [train,test]
```

The function below calculates the metric used in the competitoin, such that I can see how different models and techniques responds in accordance to the metric.

In [14]:
```
def competition_log_loss(y_true, y_pred):
    N_0 = np.sum(1 - y_true)
    N_1 = np.sum(y_true)
    p_1 = np.clip(y_pred, 1e-15, 1 - 1e-15)
    p_0 = 1 - p_1
    log_loss_0 = -np.sum((1 - y_true) * np.log(p_0)) / N_0
    log_loss_1 = -np.sum(y_true * np.log(p_1)) / N_1
    return (log_loss_0 + log_loss_1)/2
```

As known from the histogram of the target variable; the target variable is imbalanced. Therefore I choose to handle this problem by use of oversampling.

In [15]:
```
def oversampling(df):
    ones = df.loc[df.Class==1]
    zeros = df.loc[df.Class==0]
    ratio = zeros.shape[0]/ones.shape[0]

    concats = [zeros]
    for i in range(0,int(ratio),1):
        concats.append(ones)
    dont_care, additional_ones = train_test_split(ones, test_size=ratio-int(ratio)
    concats.append(additional_ones)
    df = pd.concat(concats)

    return df
```

The competiton also gave us metadata. The function below preprocesses the metadata.

In [16]:
```
def greeks_le(df):
    le = LabelEncoder()
    cols = df.columns
    le = LabelEncoder()

    df['Alpha'] = le.fit_transform(df['Alpha'])
    return df
```

In [17]:
```
def rename(data):
    train = data[0]
    test = data[1]

    train = train.rename(columns = {'0':'STH1','1':'STH2'})
    test = test.rename(columns = {'0':'STH1','1':'STH2'})
    return [train,test]
```

Models for metadata and data

The two functions below creates code for predicting the metadata and the target variable.

In [18]:
```
def pred_indicators(data, test):
    train = data
    test = test

    pred_cols = train.columns[69:][::-1]
    predictions = []
    X_cols =  pd.Series(train.columns.drop(train.columns[69:]))

    for i in range(0,len(pred_cols),1):

        model = RandomForestClassifier(random_state=42,max_depth=5,n_estimators=5)
        model.fit(train[X_cols],train[pred_cols[i]])

        train_pred = model.predict(train[X_cols])
        train[pred_cols[i]] = train_pred

        test_pred = model.predict(test[X_cols])
        test[pred_cols[i]] = test_pred

        X_cols = pd.Series(X_cols).append(pd.Series(pred_cols[i]))


    return [train, test]
```

In [19]:
```
def good_rcf_model(df='', features=''):
    param_dist = {'n_estimators': [rd.randint(10,42)],
             'max_depth':[rd.randint(8,30)]}

    rfc = RandomForestClassifier(random_state=42)

    rand_search = RandomizedSearchCV(rfc,
                                      random_state=42,
                                    param_distributions=param_dist,
                                    n_iter=5,
                                    cv=25,
                                     scoring='neg_log_loss')

    rand_search.fit(df[features],df[y])
    best = rand_search.best_estimator_

    return best
```

Competition submission phase

The first code below works as the pipeline for my model. The other code lines uses the preprocessed data, predicts it and submits it to the competition.

In [20]:
```
work = pd.read_csv('/kaggle/input/icr-identify-age-related-conditions/train.csv')

del work['Id']

greeks = pd.read_csv("/kaggle/input/icr-identify-age-related-conditions/greeks.csv"
del greeks['Epsilon']
del greeks['Id']
del greeks['Delta']

check = pd.read_csv('/kaggle/input/icr-identify-age-related-conditions/test.csv')
check_id = check['Id']
del check['Id']
```

```
data = pd.concat([work,greeks],axis=1)
data = oversampling(data)
train_Class = data['Class'].reset_index()
del train_Class['index']
del data['Class']

run_now = data.loc[:,:data.columns[55]]
run_later = data.loc[:,data.columns[56]:]

data = onehot([run_now,check])

data = nan_ext(data)
data = binary(data,threshhold=15,name='GL')
data =  binary(data,threshhold=240,name='BQ')
data = binary(data,threshhold=12.5,name='CW ')
data = outliers(data)
data = rename(data)

run_later = onehot([greeks_le(run_later),greeks_le(run_later)])[0]
run_later['Alpha']  = np.where(run_later['Alpha']>=0.5, 1, 0)

part_one = run_later.reset_index()
del part_one['index']
part_two = data[0].reset_index()
del part_two['index']

df = pd.concat([part_two, part_one], axis=1)
test = data[1]
data = pred_indicators(df, test)
```

In [21]:
```
train = data[0]
train['Class'] = train_Class
test = data[1]

X = train.columns.drop('Class')
y =  'Class'
sub_feat = mutual_info(train).index[:]
```

In [22]:
```
print(good_rcf_model(df=train, features=sub_feat))
```

```
RandomForestClassifier(max_depth=18, n_estimators=28, random_state=42)
```

In [23]:
```
#21,20 gives = 0.006334
final_sub_model = RandomForestClassifier(random_state=42, max_depth=3, n_estimators
final_sub_model.fit(train[sub_feat],train[y])

pred_prob_train = final_sub_model.predict_proba(train[sub_feat])
log_loss = competition_log_loss(train[y], pd.Series([i[1] for i in pred_prob_train]

print('Train accuracy: ',accuracy_score(final_sub_model.predict(train[sub_feat]), t
print('Train LogLoss:', log_loss)
```

```
Train accuracy:  0.9587426326129665
Train LogLoss: 0.23636051163027844
```

In [24]:
```
final_test_pred_prob = final_sub_model.predict_proba(test[sub_feat])

sample_submission = pd.DataFrame({'Id':check_id}).reset_index()
del sample_submission['index']
sample_submission['class_0'] = [i[0] for i in final_test_pred_prob]
sample_submission['class_1'] = [i[1] for i in final_test_pred_prob]
```

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