Deep Mailing - XGBoost Model - Reduzindo Dimensões

O objetivo desse notebook é demonstrar a utilizacao do XGBoost para a criacao de arvores de decisão para a predição de CUPS em mailings.

Em primeiro lugar, definimos os imports que iremos usar...

In [1]:
```
import xgboost
import numpy as np
import os
import sys
import logging
import gc
import pickle as pickle
import pandas as pd
import dateutil.parser as parser
import os.path
import math
from sklearn.metrics import accuracy_score
from datetime import datetime
import xgboost as xgb
from xgboost import XGBClassifier
from xgboost import plot_tree
import matplotlib.pyplot as plt
from matplotlib.pylab import rcParams
```

```
/home/ubuntu/git/DeepMailing/env/lib/python3.5/site-packages/sklearn/cross_validatio
n.py:41: DeprecationWarning: This module was deprecated in version 0.18 in favor of
the model_selection module into which all the refactored classes and functions are m
oved. Also note that the interface of the new CV iterators are different from that o
f this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

Abaixo definimos os diretorios e nomes dos arquivos intermediarios.

In [2]:
```
log_location = "../logs/"
arquivo_df_pickled_norm = "../intermediate/df.norm.pickle"
arquivo_df_pickled_norm_train = "../intermediate/df.norm.train.pickle"
arquivo_df_pickled_norm_test = "../intermediate/df.norm.test.pickle"
arquivo_df_pickled_norm_train_x = "../intermediate/df.norm.train.x.pickle.npy"
arquivo_df_pickled_norm_train_y = "../intermediate/df.norm.train.y.pickle.npy"
arquivo_df_pickled_norm_test_x = "../intermediate/df.norm.test.x.pickle.npy"
arquivo_df_pickled_norm_test_y = "../intermediate/df.norm.test.y.pickle.npy"
```

Redefinimos o logger que iremos usar

In [3]:
```
logger = logging.getLogger()
logging.basicConfig(format="%(asctime)-15s %(message)s",
                    level=logging.DEBUG,
                    filename=os.path.join(log_location,'xgboost.log.' + datetime.now
().strftime("%Y%m%d%H%M%S.%f") + '.log'))
```

Criamos uma função para imprimir tanto no log quanto no notebook...

In [4]:
```
def print_log(msg):
    logging.debug(msg)
    print(msg)
```

Carregamos para a memoria o arquivo normalizado e pickled que foi gerado no notebook de "Preparação de Dados".

In [5]:
```
print_log("Carregando Pickling normalizado:{}".format(arquivo_df_pickled_norm))
chamadas = pd.read_pickle(arquivo_df_pickled_norm)
```

```
Carregando Pickling normalizado:../intermediate/df.norm.pickle
```

Verificamos as dimensões do dataframe carregado.... E imprimimos uma amostra do dado que precisamos com apenas as colunas relevantes... Como podemos perceber o nosso modelo considera apenas colunas com valores booleanos (0 ou 1)

In [6]:
```
print_log(chamadas.shape)
chamadas.loc[:, 'NORM_CARTEIRA_A01':'NORM_DDD_87'].head(10)
```

```
(1435423, 103)
```

Out[6]:

	NORM_CARTEIRA_A01	NORM_CARTEIRA_W01	NORM_CARTEIRA_W02	NORM_CARTEIRA_
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	0	0	0	1
4	0	0	0	1
5	0	0	1	0
6	0	0	0	0
7	0	1	0	0
8	0	0	0	1
9	0	0	0	1

10 rows × 98 columns

Para tratarmos as colunas de forma correta, precisamos eliminar os caracteres especiais das colunas, principalmente o espaco e o sinal de - para isso iremos rodar o codigo abaixo para tratar.

In [7]:
```
cols = chamadas.columns
cols = cols.map(lambda x: x.replace(' ', '_').replace('-','menos'))
chamadas.columns = cols
```

Um problema que temos é q temos muito poucos CUPS sendo acionados, então temos q remover linhas não relevantes no arquivo.

As linhas que devemos remover são aquelas em que não houve CUP e cujas features apesar de ativadas (terem valor 1), raramente são determinantes para um CUP.

No exemplo abaixo, eu vou determinar que se em menos de 15% das vezes em q determinada feature foi ativada, foi ativado um CUP tambem, eu vou determinar que essa coluna é irrelevante e que todos os valores com ativação dessa coluna e que não haja CUP seja apagados.

Dessa forma, iremos remover linhas ativações de features que não geraram NORM_CUP =1 e q sabemos q não são relevantes ao modelo.

No final, calculamos o threshold, que é o numero minimo de CUPs ativados para determinarmos se aquela feature é relevante ou não.

In [8]:
```
minima_expressividade = 0.15
dimensionalidade_cup = minima_expressividade / (len(chamadas.query("NORM_CUP==1").ind
ex) /  len(chamadas.index))
dimensionalidade_cup
```

Out[8]: 76.05561638996821

Como já calculamos o numero minimo de CUPs gerados para que uma determinada feature seja gerada, podemos agora determinar quais as colunas que devem ser limpas do dataframe

In [9]:
```
def limpar_coluna_df(dataframe, coluna):
    return dataframe.query("not ({} == 1 and NORM_CUP==0)".format(coluna))

def coluna_tem_cup(dataframe, coluna):
    return len(dataframe.query("{} == 1 and NORM_CUP==1".format(coluna)).index) > dim
ensionalidade_cup

colunas_para_limpar = []
lista_colunas = chamadas.loc[:, 'NORM_CARTEIRA_A01':].columns.values
for coluna in list(lista_colunas):
    if not coluna_tem_cup(chamadas,coluna):
        colunas_para_limpar.append(coluna)
        print_log("Coluna {} foi limpa do dataframe por ser irrelevante...".format(co
luna))
```

```
Coluna NORM_SEGMENTO_FT foi limpa do dataframe por ser irrelevante...
Coluna NORM_PROPENSAO_ALTA foi limpa do dataframe por ser irrelevante...
Coluna NORM_STATUS_BUREAU_Indefinido foi limpa do dataframe por ser irrelevante...
Coluna NORM_STATUS_INTERNA_Novo foi limpa do dataframe por ser irrelevante...
Coluna NORM_STATUS_TELEFONE_Indefinido foi limpa do dataframe por ser irrelevant
e...
Coluna NORM_STATUS_TELEFONE_Validado foi limpa do dataframe por ser irrelevante...
Coluna NORM_STATUS_TELEFONE_Novo foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_63 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_24 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_96 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_22 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_38 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_79 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_69 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_53 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_34 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_94 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_84 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_75 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_88 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_68 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_14 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_32 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_37 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_91 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_92 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_99 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_86 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_97 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_49 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_33 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_61 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_65 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_67 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_44 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_42 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_89 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_77 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_46 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_28 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_27 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_41 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_48 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_98 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_18 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_82 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_16 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_64 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_15 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_66 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_73 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_43 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_12 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_95 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_55 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_13 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_74 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_35 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_93 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_62 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_17 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_54 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_47 foi limpa do dataframe por ser irrelevante...
```

```
Coluna NORM_DDD_45 foi limpa do dataframe por ser irrelevante...
Coluna NORM_DDD_87 foi limpa do dataframe por ser irrelevante...
```

Tendo as colunas irrelevantes, podemos agora limpar elas do nosso dataframe, oque vai fazer com ele fique com menos tamanho e com melhor precisão.

In [10]:
```
for coluna in colunas_para_limpar:
    chamadas = limpar_coluna_df(chamadas,coluna)
    gc.collect()
```

Podemos agora analisar o tamanho final do nosso dataframe:

In [11]:
```
len(chamadas.index)
```

Out[11]:
```
449681
```

Perfeito, então vamos embaralhar os dados e gerar os nossos dados de treinamento e teste. Vamos considerar 70% para treinamento e 30 % para teste. No final, apagamos o dataframe lido para economizar memoria

In [12]:
```
print_log("Criando Pickling de train e teste...")
chamadas = chamadas.sample(int(len(chamadas.index)))
chamadas_train = chamadas.tail(int(len(chamadas.index) * 0.7))
chamadas_test = chamadas.head(int(len(chamadas.index) * 0.3))
del chamadas
```

```
Criando Pickling de train e teste...
```

Criamos uma função para gerar um arquivo de referencia de colunas a serem usadas q que vai ser importante na hora de gerar a arvore de decisao...

In [13]:
```
def create_column_reference(header_chamadas_x,arquivo_df_pickled_norm_train_x):
    print_log("Criando Arquivo de referencia de colunas...")
    with open(arquivo_df_pickled_norm_train_x+".txt","w") as f:
        counter = 0
        lista_header = list(header_chamadas_x.columns.values)
        for header in lista_header:
            f.write("{}-{}\n".format(counter,header))
            counter=counter+1
```

Criamos agora os nossos dataframes de X que são as features e as Y que são os alvos de predição. Também removemos qualquer linha em tentativas seja igual a zero, além de criar um arquivo de referencia com as colunas X que serão usadas no modelo. Esses dataframes serão convertidos para matrizes no formato numpy

In [14]:
```
print_log("Separando colunas em X e Y...")
chamadas_train = chamadas_train[(chamadas_train.NORM_TENTATIVAS > 0)]
create_column_reference(chamadas_train.loc[:, chamadas_train.columns.values[2]:'NORM_
DDD_87'].head(1), arquivo_df_pickled_norm_train_x)
chamadas_train_x = chamadas_train.loc[:, chamadas_train.columns.values[2]:'NORM_DDD_8
7'].as_matrix()
chamadas_train_y = chamadas_train.NORM_CUP.as_matrix()
chamadas_test = chamadas_test[(chamadas_test.NORM_TENTATIVAS > 0)]
chamadas_test_x = chamadas_test.loc[:, chamadas_test.columns.values[2]:'NORM_DDD_87']
.as_matrix()
chamadas_test_y = chamadas_test.NORM_CUP.as_matrix()
```

```
Separando colunas em X e Y...
Criando Arquivo de referencia de colunas...
```

Após a criação das matrizes numpy, gravamos elas em arquivos.

In [15]:
```
print_log("Criando arquivos finais em formato NUMPY para consumo pelo algoritmo...")

np.save(arquivo_df_pickled_norm_train_x,chamadas_train_x)
np.save(arquivo_df_pickled_norm_train_y,chamadas_train_y)
np.save(arquivo_df_pickled_norm_test_x,chamadas_test_x)
np.save(arquivo_df_pickled_norm_test_y,chamadas_test_y)
```

```
Criando arquivos finais em formato NUMPY para consumo pelo algoritmo...
```

Apagamos todos os dados intermediários e rodamos o garbage collector para economizar memória.

In [16]:
```
print_log("Removendo objetos desnecessarios")
del chamadas_train_x
del chamadas_train_y
del chamadas_train
del chamadas_test
del chamadas_test_x
del chamadas_test_y
gc.collect()
```

```
Removendo objetos desnecessarios
```

Out[16]:
```
70
```

Carregamos os objetos numpy em memoria

In [17]:
```
print_log("Carregando objetos numpy")
train_x = np.load(arquivo_df_pickled_norm_train_x)
train_y = np.load(arquivo_df_pickled_norm_train_y)
test_x = np.load(arquivo_df_pickled_norm_test_x)
test_y = np.load(arquivo_df_pickled_norm_test_y)
```

```
Carregando objetos numpy
```

Contamos quantos CUPS existem em treinamento e teste...

In [18]:
```
msg1 = "Train - CUPS Detectados {} num universo de {}".format(len([y for y in train_y
 if y >0]),len(train_y))
msg2 = "Test - CUPS Detectados {} num universo de {}".format(len([y for y in test_y i
f y >0]),len(test_y))
print_log(msg1)
print_log(msg2)
```

```
Train - CUPS Detectados 1989 num universo de 314776
Test - CUPS Detectados 842 num universo de 134903
```

Configuramos os parametros para o XGBoost, especificando que queremos que ele seja o mais exato possivel, que a medida de erro é erro simples e que queremos apenas uma classificacao binaria com um maximo de 1000 interações

In [19]:
```
param = {}
param['eta'] = 0.2
param['objective'] = 'binary:logistic'
param['eval_metric'] = 'error'
param['tree_method'] = 'exact'
param['silent'] = 0
num_round = 1000
```

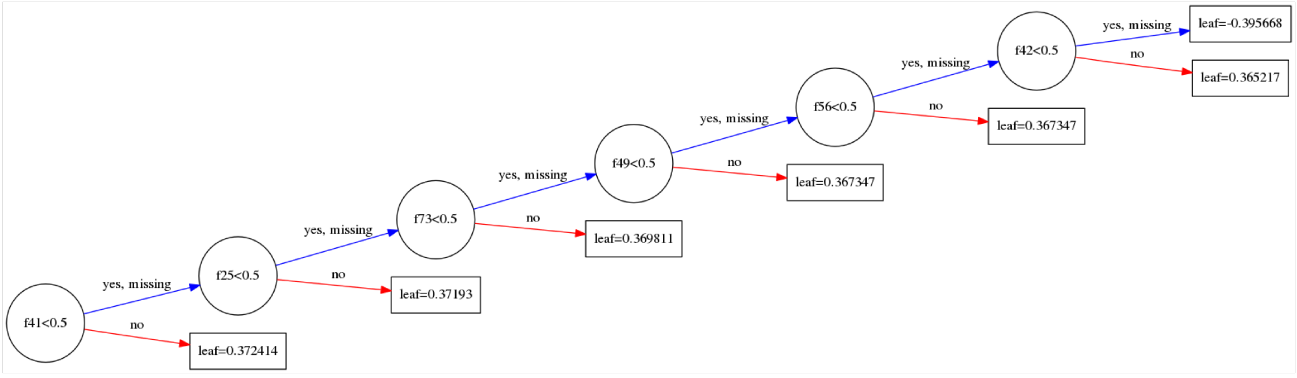
Após a definicão dos paramêtros de teste, criamos as matrizes no formato do XGBoost e treinamos o modelo.

In [20]:
```
gc.collect()
print_log("Starting model for params:{}".format(param))
dtrain = xgb.DMatrix(train_x, train_y)
dtest = xgb.DMatrix(test_x, test_y)
gpu_res = {}
booster = xgb.train(param, dtrain, num_round, evals=[], evals_result=gpu_res)
```

```
Starting model for params:{'silent': 0, 'eval_metric': 'error', 'eta': 0.2, 'tree_me
thod': 'exact', 'objective': 'binary:logistic'}
```

Após o modelo ser treinado, podemos plotar ele... Para verificar que coluna é cada feature no modelo, por favor ver a lista em anexo no final desse notebook.

In [21]:
```
%matplotlib inline
rcParams['figure.figsize'] = 80,50
plot_tree(booster, rankdir='LR')
plt.show()
```



Agora, vamos tentar predizer os dados com o nosso modelo treinado...

In [22]:
```
test_y_pred = booster.predict(dtest)
test_predictions = np.array([value for value in test_y_pred])
```

E Finalmente medir a precisão da nossa predição... Tanto no total quanto em CUPs detectados.

In [23]:
```
accuracy = accuracy_score(test_y, test_predictions.round())
print_log("CUPS Previstos:{}".format(len([x for x in test_predictions if x > 0.5])))
print_log("CUPS na Base Teste:{}".format(len([x for x in test_y if x > 0.5])))
print_log("Accuracy Total:{}".format(accuracy))
print_log("Accuracy em CUPs:{}".format(len([x for x in test_predictions if x > 0.5])
/ len([x for x in test_y if x > 0.5])))
```

```
CUPS Previstos:383
CUPS na Base Teste:842
Accuracy Total:0.9965975552804608
Accuracy em CUPs:0.45486935866983375
```

Após, vamos salvar o modelo gerado em um arquivo para reuso...

In [24]:
```
save_file = "../output/{}.model".format(datetime.now().strftime("%Y%m%d.%H%M%S"))
with open(save_file, 'wb') as fp:
    pickle.dump(booster, fp)
print_log("Model saved as {}".format(save_file))
```

```
Model saved as ../output/20171215.180229.model
```

In [25]:
```
%%bash

cat ../intermediate/df.norm.train.x.pickle.npy.txt
```

0-NORM_CARTEIRA_A01
1-NORM_CARTEIRA_W01
2-NORM_CARTEIRA_W02
3-NORM_CARTEIRA_A02
4-NORM_CARTEIRA_A03
5-NORM_SEGMENTO_CR
6-NORM_SEGMENTO_FA
7-NORM_SEGMENTO_CC
8-NORM_SEGMENTO_LC
9-NORM_SEGMENTO_HC
10-NORM_SEGMENTO_FT
11-NORM_SEGMENTO_FC
12-NORM_SEGMENTO_MA
13-NORM_PROPENSAO_ALTA
14-NORM_ORIGEM_BUREAU
15-NORM_ORIGEM_BASE_INTERNA
16-NORM_STATUS_BUREAU_Indefinido
17-NORM_STATUS_BUREAU_Bom_menos
18-NORM_STATUS_BUREAU_Bom
19-NORM_STATUS_BUREAU_nan
20-NORM_STATUS_INTERNA_Bom
21-NORM_STATUS_INTERNA_Hot
22-NORM_STATUS_INTERNA_Novo
23-NORM_STATUS_INTERNA_Indefinido
24-NORM_STATUS_INTERNA_Validado
25-NORM_STATUS_TELEFONE_Indefinido
26-NORM_STATUS_TELEFONE_Hot
27-NORM_STATUS_TELEFONE_Validado
28-NORM_STATUS_TELEFONE_Bom
29-NORM_STATUS_TELEFONE_Bom_menos
30-NORM_STATUS_TELEFONE_Novo
31-NORM_DDD_63
32-NORM_DDD_24
33-NORM_DDD_96
34-NORM_DDD_22
35-NORM_DDD_38
36-NORM_DDD_79
37-NORM_DDD_69
38-NORM_DDD_53
39-NORM_DDD_34
40-NORM_DDD_94
41-NORM_DDD_84
42-NORM_DDD_75
43-NORM_DDD_88
44-NORM_DDD_68
45-NORM_DDD_14
46-NORM_DDD_32
47-NORM_DDD_85
48-NORM_DDD_37
49-NORM_DDD_91
50-NORM_DDD_92
51-NORM_DDD_99
52-NORM_DDD_86
53-NORM_DDD_97
54-NORM_DDD_49
55-NORM_DDD_33
56-NORM_DDD_61
57-NORM_DDD_65
58-NORM_DDD_67
59-NORM_DDD_44
60-NORM_DDD_42
61-NORM_DDD_89
62-NORM_DDD_19
63-NORM_DDD_77
64-NORM_DDD_46

65-NORM_DDD_28
66-NORM_DDD_27
67-NORM_DDD_51
68-NORM_DDD_71
69-NORM_DDD_41
70-NORM_DDD_48
71-NORM_DDD_98
72-NORM_DDD_18
73-NORM_DDD_82
74-NORM_DDD_11
75-NORM_DDD_16
76-NORM_DDD_64
77-NORM_DDD_15
78-NORM_DDD_66
79-NORM_DDD_73
80-NORM_DDD_43
81-NORM_DDD_12
82-NORM_DDD_95
83-NORM_DDD_83
84-NORM_DDD_55
85-NORM_DDD_13
86-NORM_DDD_21
87-NORM_DDD_74
88-NORM_DDD_35
89-NORM_DDD_93
90-NORM_DDD_81
91-NORM_DDD_62
92-NORM_DDD_17
93-NORM_DDD_54
94-NORM_DDD_47
95-NORM_DDD_31
96-NORM_DDD_45
97-NORM_DDD_87