Deep Mailing - Preparação de Dados

O Objetivo do Deep Mailing é criar uma forma de se prever o CUP de um telefone baseado no seu historico. Para isso iremos criar um dataset que contem as features de cada um dos telefones que gerou CUP e criar um modelo para sua predição usando diversas técnicas.

Esse notebook contém toda a logica de preparação do nosso dataset que será usado nos outros notebooks que implementam algoritmos de detecção de padrões (machine learning) especificos.

Configuração do Kernel em Python

A primeira tarefa no nosso Kernel é a importação dos módulos necessarios e a definição de diversos parametros operacionais como a localização de logs, os arquivos de dados a serem gerados e seus nomes além de configuração especificas de módulos.

Os parametros do nosso Kernel que são definidos hardcoded são:

- **log_location**: Caminho aonde serão gerados os logs desse kernel.
- **arquivo_chamadas**: O caminho completo até o arquivo txt que contem os dados de chamadas
- **arquivo_df_pickled**: O Arquivo pickle que será a serialização de um dataframe pandas com os dados brutos importados apartir do arquivo texto acima.
- **arquivo_df_pickled_norm**: O Arquivo pickle que será a serialização de um dataframe pandas com os dados categorizados discretos explodidos em colunas discretas booleanas, ou seja: CLUSTER será explodido em NORM_CLUSTER_A01, NORM_CLUSTER_A02, NORM_CLUSTER*W01 e assim por diante, e essas colunas NORM** tem apenas os valores possiveis de 0 ou 1.
- **pd.options.display**: Configuracao do pandas para o numero maximo de colunas a ser mostrado quando renderizando o html de um dataframe.
- **num_partitions**: Quando fazendo multiprocessamento dos dataframes, o numero de particoes em que o dataframe original sera repartido.
- **num_cores**: O numero de cores que iremos assumir na maquina para o multiprocessamento de dataframes.
- **Normalizado**: Variavel global que controla se os dados já estao normalizados, se sim, ou seja, detectamos um arquivo *arquivo_df_pickled_norm* no local indicado, então assumimos que não precisamos normalizar o dataframe

In [ ]:
```
import pandas as pd
import dateutil.parser as parser
import os.path
import math
import logging
import numpy as np
from multiprocessing import Pool
from datetime import datetime
```

In [ ]:
```
log_location = "../logs/"
arquivo_chamadas = "../data/mailing_completo.txt.full"
arquivo_df_pickled = "../intermediate/df.pickle"
arquivo_df_pickled_norm = "../intermediate/df.norm.pickle"
```

In [ ]:
```
pd.options.display.max_columns = 50
num_partitions = 8
num_cores = 8
Normalizado = False
```

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

Funções de Conversão

Essas funções são uteis para a conversão dos dados txt para os formats de dados corretos a serem usados nos dataframes do pandas.

In [ ]:
```
logging.debug("Declarando as funcoes globais")

def IsInt(s):
    try:
        int(s)
        return True
    except ValueError:
        return False

def IsIntAndGreaterZero(s):
    return IsInt(s) and int(s)>0
```

In [ ]:
```

def IsFloat(s):
    try:
        float(s)
        return True
    except ValueError:
        return False

def IsDatetime(s):
    try:
        parser.parse(s)
        return True
    except ValueError:
        return False
```

In [ ]:
```
def ConverterInt(val):
    val = val.replace(",",".")
    if IsInt(val):
        return int(val)
    else:
        return 0

def ConverterFloat(val):
    val = val.replace(",",".")
    if IsFloat(val):
        return float(val)
    else:
        return 0
```

In [ ]:
```
def ConverterData(val):
    if IsDatetime(val):
        return parser.parse(val)
    else:
        return val
```

Funções para verificar existencia de arquivos

As funções abaixo são usadas para verificar a existencia de arquivos diversos

In [ ]:
```
def IsCSVDataAvailable():
    return os.path.isfile(arquivo_df_pickled)

def IsNormDataAvailable():
    return os.path.isfile(arquivo_df_pickled_norm)
```

In [ ]:
```
def IsTrainDataAvailable():
    return os.path.isfile(arquivo_df_pickled_norm_train)

def IsNumpyArrayDataAvailable():
    return os.path.isfile(arquivo_df_pickled_norm_train_x + ".npy")
```

Funções para Paralelização via Multiprocessamento

Como existem dataframes muito grandes, é necessario paralelizar o processamento de forma a utilizar toda a capacidade de processamento do computador.

O Algoritmo é bastante simples, e se utiliza da criação de processos filhos para que se possa distribuir a carga de trabalho entre os seus filhos. Abaixo temos a funcao que permite essa serialização e distribuição para os processos filhos. Importante salientar que os dados são serializados e depois desserializados para a execução do processo filho.

In [ ]:
```
def parallelize_dataframe(func, data):
    df = data['df']
    df_split = np.array_split(df, num_partitions)
    pool = Pool(num_cores)
    items = list(data.items())
    chunksize = len(data.items())
    chunks = [items[i:i + chunksize ] for i in range(0, len(items), chunksize)]
    df = pd.concat(pool.map(func, chunks))
    pool.close()
    pool.join()
    return df
```

Funções para Normalizar as Colunas do DataFrame

Abaixo temos as funções que usaremos para desmembrar colunas que tem categorias em varias colunas com valores booleanos.

In [ ]:
```
def func_str(x):
    return str(x)

def func_strip(x):
    return str(x).strip()

def func_start_ALTA(x):
    return str(x).startswith('ALTA')
```

In [ ]:
```
def func_to_execute_column_str(data):
    data = dict(item for item in data)  # Convert back to a dict
    logging.debug("Creating Binary Column:{} in {}".format(data["col"],  data["source
_col"]))
    df = data['df']
    df['NORM_' + data["source_col"] + "_" + data["col"]] = df.apply(lambda row: 1 if
func_str(row[data["source_col"]]) == data["col"] else 0, axis=1)
    return df

def CreateColumnStr(cols, df, source_col):
    for col in cols:
        df = parallelize_dataframe(func_to_execute_column_str, { "df" : df, "source_c
ol" : source_col, "col" : col})
    return df
```

In [ ]:
```
def func_to_execute_column_strip(data):
    data = dict(item for item in data)  # Convert back to a dict
    logging.debug("Creating Binary Column:{} in {}".format(data["col"],  data["source
_col"]))
    df = data['df']
    df['NORM_' + data["source_col"] + "_" + data["col"]] = df.apply(lambda row: 1 if
func_strip(row[data["source_col"]]) == data["col"] else 0, axis=1)
    return df

def CreateColumnStrip(cols, df, source_col):
    for col in cols:
        df = parallelize_dataframe(func_to_execute_column_strip, { "df" : df, "source
_col" : source_col, "col" : col})
    return df
```

In [ ]:
```
def func_to_execute_column_ALTA(data):
    data = dict(item for item in data)  # Convert back to a dict
    logging.debug("Creating Binary Column:{} in {}".format(data["col"],  data["source
_col"]))
    df = data['df']
    df['NORM_' + data["source_col"] + "_" + data["col"]] = df.apply(lambda row: 1 if
func_start_ALTA(row[data["source_col"]]) == data["col"] else 0, axis=1)
    return df

def CreateColumnALTA(cols, df, source_col):
    for col in cols:
        df = parallelize_dataframe(func_to_execute_column_ALTA, { "df" : df, "source_
col" : source_col, "col" : col})
    return df
```

In [ ]:
```
def func_to_execute_numeric(data):
    data = dict(item for item in data)  # Convert back to a dict
    logging.debug("Creating Numeric Column:{} ".format(data["source_col"]))
    df = data['df']
    df['NORM_' + data["source_col"]] = df.apply(lambda row: 1 if IsIntAndGreaterZero(
row[data["source_col"]]) else 0, axis=1)
    return df

def CreateLogColumn(df, source_col):
    df = parallelize_dataframe(func_to_execute_numeric, { "df" : df, "source_col" : s
ource_col})
    return df
```

Tratamento dos dados de cada coluna no Dataframe

Abaixo, temos a declaração de cada tipo de dado da coluna e a função que irá tratar a informação daquela coluna e colocar no formato correto.

In [ ]:
```
logging.debug("Declarando os tipos de dados no dataframe")

df_dtypes = {
    "CPF_CNPJ": "object",
    "CARTEIRA": "object",
    "SEGMENTO": "object",
    "PRODUTO": "object",
    "FILA": "object",
    "STATUS_CONTRATO": "object",
    "PROPENSAO": "object",
    "ORIGEM": "object",
    "DETALHE_ORIGEM": "object",
    "STATUS_BUREAU": "object",
    "STATUS_INTERNA": "object",
    "DDD": "object",
    "TELEFONE": "object",
    "TELRUIM_RENITENCIA": "object",
    "TELRUIM_DISCADOR": "object",
    "STATUS_TELEFONE": "object",
    "OPERADORA": "object",
    "ORIGEM_ULTIMA_ATUALIZACAO": "object",
    "PRIMEIRA_ORIGEM": "object"
}
```

In [ ]:
```
converters = {
    "ATRASO":  ConverterInt,
    "VALOR": ConverterFloat,
    "DT_ENTRADA" : ConverterData,
    "NLOC": ConverterInt,
    "SCORE_C": ConverterInt,
    "SCORE_E": ConverterInt,
    "RENDA": ConverterFloat,
    "DT_DEVOLUCAO": ConverterData,
    "VLRISCO": ConverterFloat,
    "SCORE_ZANC_C": ConverterInt,
    "SCORE_ZANC_E": ConverterInt,
    "SCORE_ZANC": ConverterInt,
    "DATA_PRIMEIRA_ORIGEM": ConverterData,
    "DATA_ULTIMA_ATUALIZACAO": ConverterData,
    "TENTATIVAS": ConverterInt,
    "ULT_ARQ_BUREAU": ConverterData,
    "DATA_MAILING": ConverterData,
    "LIGACOES": ConverterInt,
    "CUP": ConverterInt
}
```

Carga dos dados CSV

Abaixo, carregamos os dados em CSV em um dataframe pandas e apagamos as colunas desnecessárias para o nosso uso, e gravamos o dataframe em um arquivo para reuso pelos algoritmos.

Caso já exista o arquivo de dataframe, lemos o arquivo e criamos o dataframe para normalização

In [ ]:
```
def limpar_df(chamadas):
    del chamadas["CPF_CNPJ"]
    del chamadas["PRODUTO"]
    del chamadas["FILA"]
    del chamadas["STATUS_CONTRATO"]
    del chamadas["DETALHE_ORIGEM"]
    del chamadas["TELEFONE"]
    del chamadas["TELRUIM_RENITENCIA"]
    del chamadas["TELRUIM_DISCADOR"]
    del chamadas["OPERADORA"]
    del chamadas["ORIGEM_ULTIMA_ATUALIZACAO"]
    del chamadas["PRIMEIRA_ORIGEM"]
    del chamadas["ATRASO"]
    del chamadas["VALOR"]
    del chamadas["DT_ENTRADA" ]
    del chamadas["NLOC"]
    del chamadas["SCORE_C"]
    del chamadas["SCORE_E"]
    del chamadas["RENDA"]
    del chamadas["DT_DEVOLUCAO"]
    del chamadas["VLRISCO"]
    del chamadas["SCORE_ZANC_C"]
    del chamadas["SCORE_ZANC_E"]
    del chamadas["SCORE_ZANC"]
    del chamadas["DATA_PRIMEIRA_ORIGEM"]
    del chamadas["DATA_ULTIMA_ATUALIZACAO"]
    del chamadas["ULT_ARQ_BUREAU"]
    return chamadas
```

In [ ]:
```
logging.debug("Carregando dos dados em CSV ou normalizados, dependendo da existencia
 deles...")
if not IsCSVDataAvailable():
    chamadas = pd.read_csv(arquivo_chamadas, sep="|", dtype=df_dtypes, converters = c
onverters)
    logging.debug("CSV carregado, limpando colunas desnecessarias")
    chamadas = limpar_df(chamadas)
    logging.debug("Gravando DataFrame Pandas gerado...")
    chamadas.to_pickle(arquivo_df_pickled)
else:
    if not IsNormDataAvailable():
        chamadas = pd.read_pickle(arquivo_df_pickled)
    else:
        chamadas = pd.read_pickle(arquivo_df_pickled_norm)
        Normalizado = True

logging.debug("Normalizado:{}".format(Normalizado))
```

Calculo das datas maximas e Minimas

Precisamos então calcular as datas minimas e maximas no nosso mailing...

In [ ]:
```
logging.debug("Calculando Datas minimas e maximas...")
data_maxima_mailing = chamadas.DATA_MAILING.max()
data_minima_mailing = chamadas.DATA_MAILING.min()

print("Max:{} Min:{}".format(data_maxima_mailing, data_minima_mailing))
```

Normalizamos a Coluna de Carteiras

In [ ]:
```
logging.debug("Normalizando Carteiras...")
if not Normalizado:
    Carteiras = set([x for x in chamadas.CARTEIRA.unique()[:-1] if len(x) == 3])
    chamadas = CreateColumnStr(Carteiras,chamadas, 'CARTEIRA')
```

Normalizamos a Coluna de Segmentos

In [ ]:
```
logging.debug("Normalizando Segmentos...")
if not Normalizado:
    Segmentos = set([x.strip() for x in chamadas.SEGMENTO.unique()[:-1] if len(x.stri
p()) == 2])
    chamadas = CreateColumnStrip(Segmentos,chamadas, 'SEGMENTO')
```

Normalizamos a Coluna de Chamadas

In [ ]:
```
logging.debug("Normalizando Chamadas...")
if not Normalizado:
    Propensao = set([x[:4] for x in chamadas.PROPENSAO.unique() if str(x).startswith(
"ALTA")])
    chamadas = CreateColumnALTA(Propensao,chamadas, 'PROPENSAO')
```

Normalizamos a Coluna de Origem

In [ ]:
```
logging.debug("Normalizando Origem...")
if not Normalizado:
    Origem = set([x for x in chamadas.ORIGEM.unique()[:-1]])
    chamadas = CreateColumnStr(Origem,chamadas, 'ORIGEM')
```

Normalizamos a Coluna de Status de Bureau

In [ ]:
```
logging.debug("Normalizando StatusBureau...")
if not Normalizado:
    StatusBureau = set([str(x) for x in chamadas.STATUS_BUREAU.unique()])
    chamadas = CreateColumnStr(StatusBureau,chamadas, 'STATUS_BUREAU')
```

Normalizamos a Coluna de Status Interna

In [ ]:
```
logging.debug("Normalizando StatusInterna...")
if not Normalizado:
    StatusInterna = set([str(x) for x in chamadas.STATUS_INTERNA.unique()[:-1]])
    chamadas = CreateColumnStr(StatusInterna,chamadas, 'STATUS_INTERNA')
```

Normalizamos a Coluna de Status de Telefone

In [ ]:
```
logging.debug("Normalizando Telefone...")
if not Normalizado:
    StatusTelefone = set([str(x) for x in chamadas.STATUS_TELEFONE.unique()[:-1]])
    chamadas = CreateColumnStr(StatusTelefone,chamadas, 'STATUS_TELEFONE')
```

Normalizamos a Coluna de DDD

In [ ]:
```
logging.debug("Normalizando DDD...")
if not Normalizado:
    DDD = set([str(x) for x in chamadas.DDD.unique()[:-1]])
    chamadas = CreateColumnStr(DDD,chamadas, 'DDD')
```

Normalizamos as Colunas com as quantidades de tentativas, ligacoes e CUP

In [ ]:
```
logging.debug("Normalizando LIGACOES E TAL...")
if not Normalizado:
    chamadas = CreateLogColumn(chamadas,'TENTATIVAS')
    chamadas = CreateLogColumn(chamadas,'LIGACOES')
    chamadas = CreateLogColumn(chamadas,'CUP')
```

Finalmente removemos as colunas que normalizamos e salvamos o dataframe...

In [ ]:
```
logging.debug("Removendo Campos desnecessarios e pickling...")
if not Normalizado:
    del chamadas['NUMERO']
    del chamadas['TENTATIVAS']
    del chamadas['LIGACOES']
    del chamadas['CUP']
    del chamadas['DDD']
    del chamadas['STATUS_TELEFONE']
    del chamadas['STATUS_INTERNA']
    del chamadas['STATUS_BUREAU']
    del chamadas['ORIGEM']
    del chamadas['SEGMENTO']
    del chamadas['PROPENSAO']
    del chamadas['CARTEIRA']
    chamadas.to_pickle(arquivo_df_pickled_norm)
```