# Algoritmo ID3 para Classificação de Campanhas de Marketing

Inteligência na Web e Big Data

Douglas Galetti Ribeiro

*Prof. Dr. Fabricio Olivetti de França*

*Santo André - 2018*

Universidade Federal do ABC

# Objetivos

- Implementação árvorde de decisão ID3
- Classificação de desempenho de campanhas de marketing durante a semana

# Pré-Processamento

- Transformação dos dados para serem lidos pelo classificador:

fbz_gld_bonif_suarenda_f2,,2018-05-04 09:39:13,2018-05-04 09:39:22,,1,0,,,,,1,5,,5585987812007
fbz_gld_bonif_suarenda_f2,,2018-05-04 09:39:06,2018-05-04 09:39:16,,1,0,,,,,1,5,,5533988480011
fbz_gld_bonif_suarenda_f2,,2018-05-04 09:54:17,2018-05-04 09:54:30,,1,0,,,,,1,5,,5575988362030

...

fbz_gld_bonif_suarenda_f2,fri,baixo,normal,RUIM
fbz_mov_bonif_marvel_f4,fri,baixo,normal,RUIM
fbz_mov_marvel_f3,fri,alto,normal,BOM
fbz_mov_starwars_f4,fri,alto,normal,BOM
fbz_ups_bonif_gamedom_f4,fri,baixo,normal,RUIM
fbz_zed_revista_f13,fri,alto,normal,BOM

...

# Pré-Processamento

```
OUTPUT_PATH="./"

> ${OUTPUT_PATH}/final_nomsisdn.csv
while read line
do
        DAY=$(echo $line | cut -d '_' -f2)
        CAMP_NAME=$(echo $line | sed -r 's/^[^_]*_[^_]*_(.*)_#.*$/\1/')
        VAR="$CAMP_NAME;$DAY"
        awk -v var=$VAR -F ',' '{print var","$6","$7}' $line | awk -F ',' '{ t[$1]+=1;d[$1]+=$2;a[$1]
+=$3} END { for (i in t) print i","t[i]","d[i]","a[i] }' | awk -F ',' '{printf ("%s,%1d,%d\n",$1,($3
+0.00001)/($2+0.00001)*100, ($4+0.00001)/($3+0.00001)*100)}' | awk -F ',' '{if ($3 < 2) accept =
"baixo"; else if ( $3 >= 2 && $3 <= 3) accept = "normal"; else if ($3 > 3) accept = "alto"; if ($2 <
50) delivery = "baixo"; else if ($2 >= 50 && $2 <= 70) delivery = "normal"; else if ( $2 > 70)
delivery = "alto" ; print $1","delivery","accept; }' |
sed -r 's/(baixo,baixo)/\1,PESSIMO/' |
sed -r 's/(baixo,normal)/\1,RUIM/' |
sed -r 's/(baixo,alto)/\1,BOM/' |
sed -r 's/(normal,baixo)/\1,RUIM/' |
sed -r 's/(normal,normal)/\1,BOM/' |
sed -r 's/(normal,alto)/\1,BOM/' |
sed -r 's/(alto,baixo)/\1,PESSIMO/' |
sed -r 's/(alto,normal)/\1,BOM/' |
sed -r 's/(alto,alto)/\1,EXCELENTE/'  > ${OUTPUT_PATH}/${DAY}_${CAMP_NAME}_nomsisdn.csv

        cat ${OUTPUT_PATH}/${DAY}_${CAMP_NAME}_nomsisdn.csv | sed 's/;/,/g' >> ${OUTPUT_PATH}/
final_nomsisdn.csv

done < lista
```

# Pré-Processamento

```
def processaDados (path, nomeArquivo):

    arquivo = os.path.join(path, nomeArquivo )

    # cria RDD, mapeia arquivo e obtem apenas os valores de interesse
    valores = (sc.textFile(arquivo)).map(lambda x: x.split(',')).map(lambda x: (x[0], int(x[1]), int(x[2]), int(x[3])

    # reduce da campanha com a soma dos totais
    totais = valores.map(lambda x: (x[0],x[1])).reduceByKey(lambda x,y: x+y)

    # reduce da campanha com a soma dos deliveries
    deliv = valores.map(lambda x: (x[0],x[2])).reduceByKey(lambda x,y: x+y)

    # reduce da campanha com a soma dos acceptances
    accept = valores.map(lambda x: (x[0],x[3])).reduceByKey(lambda x,y: x+y)

    # junta valores segundo o grupo (campanha)
    joinValores = totais.join(deliv).join(accept).map(lambda x: (x[0],x[1][0][0],x[1][0][1],x[1][1]))

    # calcula percentagem delivery/total e acceptance/delivery
    calculaPerc = joinValores.map(lambda x: (x[0],int(x[2]/x[1]*100) ,int(x[3]/x[2]*100)))

    # classifica percentual delivery
    deliveryPerc = calculaPerc.map(lambda x: (x[0],x[1] < 50 and 'baixo'
                                    or (x[1] >=50 and x[1] <70 and 'normal')
                                    or (x[1] >= 70 and 'alto'),x[2]))

    # classifica percentual acceptance
    acceptPerc = deliveryPerc.map(lambda x: (x[0],x[1],x[2] < 2 and 'baixo'
                                      or (x[2] >=2 and x[2] <3 and 'normal')
                                      or (x[2] >= 3 and 'alto')))

    # classifica campanha com base nas classificacoes de delivery e acceptance
    classifica = acceptPerc.map(lambda x: (x[0], x[1],x[2],
                            (x[1] == 'baixo' and x[2] == 'baixo' and 'PESSIMO')
                                    or (x[1] == 'baixo' and x[2] == 'normal' and 'RUIM' )
                                    or (x[1] == 'baixo' and x[2] == 'alto' and 'BOM')
                                    or (x[1] == 'normal' and x[2] == 'baixo' and 'RUIM')
                                    or (x[1] == 'normal' and x[2] == 'normal' and 'BOM')
                                    or (x[1] == 'normal' and x[2] == 'alto' and 'BOM')
                                    or (x[1] == 'alto' and x[2] == 'baixo' and 'PESSIMO')
                                    or (x[1] == 'alto' and x[2] == 'normal' and 'BOM')
                                    or (x[1] == 'alto' and x[2] == 'alto' and 'EXCELENTE')
                                   ))
    listaPronta = classifica.map(lambda x: [x[0].split(';'),x[1],x[2],x[3]]).map(lambda x: [x[0][0],x[0][1],x[1],x[2]
    return listaPronta
```

# ID3

- Cálculo da Entropia de um atributo:

$$E(S) = \sum_{i=1}^{c} -P_i log_2 P_i$$

- Cálculo da Entropia de mais atributos:

$$E(T, X) = \sum_{c \in X} P(c) E(c)$$

- Cálculo do Ganho de Informação:

$$GanhoInfo(T, X) = E(T) - E(T, X)$$

# ID3

```python
def calculaEntropia(i, atributos):
    valorEntropia = 0
    for label in atributos.keys():
        probabilidade = atributos[label] / i
        valorEntropia += - probabilidade * math.log(probabilidade, 2)
    return valorEntropia
```

```python
def calculoEntropiaParticao(dados, splitAtributo, atributo):

    entropia = 0
    dadosLinhas = dados['linhas']
    i = len(dadosLinhas)
    particao = particionamentoDados(dados, splitAtributo)

    for valorParticao in particao.keys():
        particionado = particao[valorParticao]
        parteParticionado = len(particionado['linhas'])
        atributosParticionado = obtemDecisao(particionado, atributo)
        entropiaParticionado = calculaEntropia(parteParticionado, atributosParticionado)
        entropia = entropia + parteParticionado / i * entropiaParticionado
    return entropia, particao
```

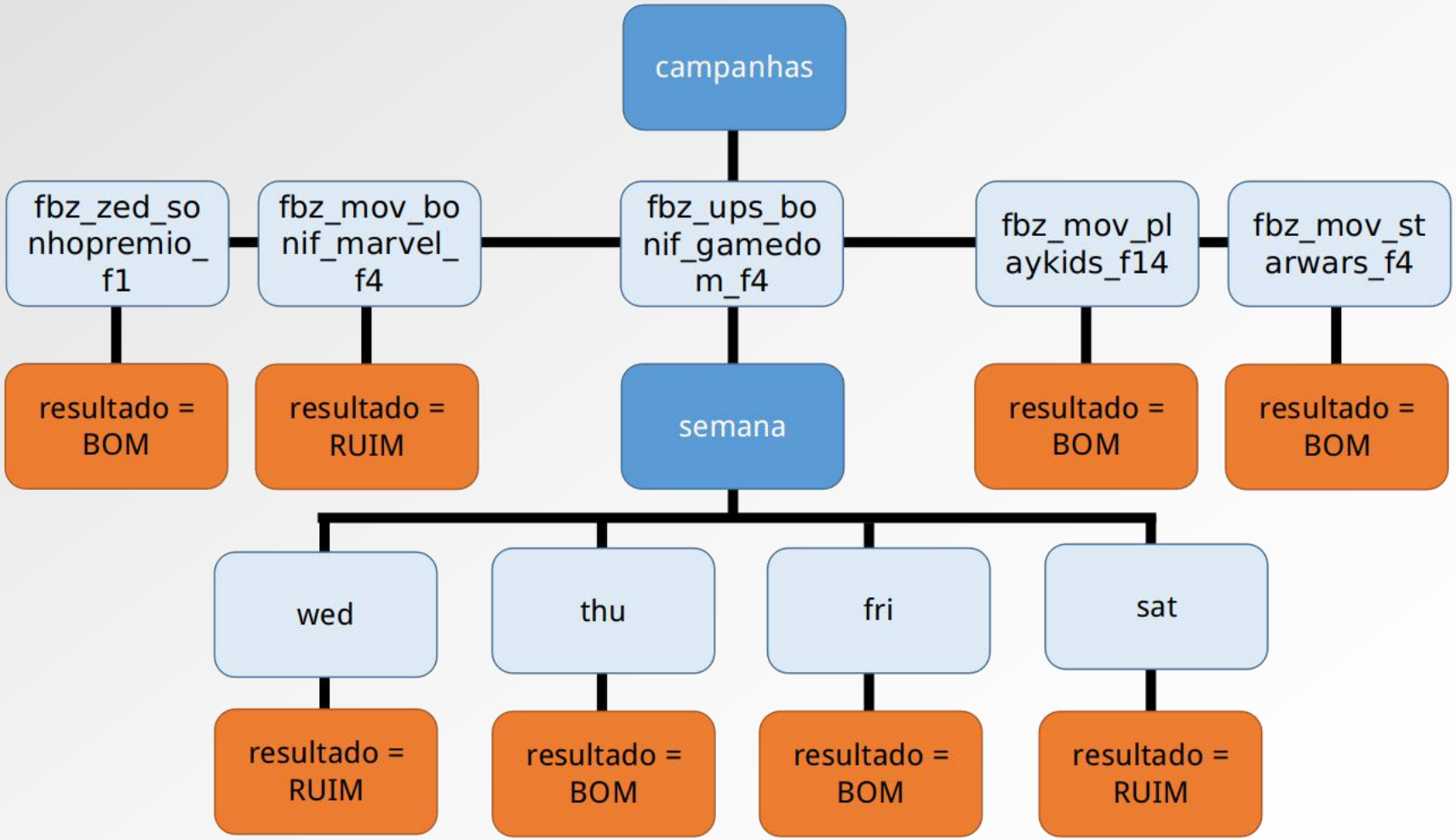
# ID3

```
i = len(amostra['linhas'])
ent = calculaEntropia(i, atributos)
maxInfoGain = None
maxInfoGainAtributo = None
maxInfoGainParticao = None

for restoAtributo in atributosRestantes:
    entropia, particao = calculoEntropiaParticao(amostra, restoAtributo, atributo)
    infoGain = ent - entropia
    if maxInfoGain is None or infoGain > maxInfoGain:
        maxInfoGain = infoGain
        maxInfoGainAtributo = restoAtributo
        maxInfoGainParticao = particao

if maxInfoGain is None:
    node['atributo'] = atributoComum(atributos)
    return node

node['atributo'] = maxInfoGainAtributo
node['nodes'] = {}
atributosRestantesRamos = set(atributosRestantes)
atributosRestantesRamos.discard(maxInfoGainAtributo)
valoresUnicos = unicos[maxInfoGainAtributo]

for valorAtributos in valoresUnicos:
    if valorAtributos not in maxInfoGainParticao.keys():
        node['nodes'][valorAtributos] = {'atributo': atributoComum(atributos)}
        continue
    partition = maxInfoGainParticao[valorAtributos]
    node['nodes'][valorAtributos] = classificador(partition, unicos, atributosRestantesRamos, atributo)
```

# Resultados: Árvore de Decisão

{'atributo': 'CAMPANHA', 'nodes': {'fbz_mov_marvel_f3': {'atributo': 'ACEITE', 'nodes': {'alto': {'atributo': 'EXCELENTE'}, 'normal': {'atributo': 'BOM'}}}, 'fbz_mov_ubook_f5': {'atributo': 'BOM'}, 'fbz_ups_bonif_gamedom_f4': {'atributo': 'SEMANA', 'nodes': {'wed': {'atributo': 'RUIM'}, 'thu': {'atributo': 'BOM'}, 'mon': {'atributo': 'BOM'}, 'fri': {'atributo': 'RUIM'}, 'tue': {'atributo': 'BOM'}, 'sat': {'atributo': 'BOM'}}}, 'fbz_zed_clubfun_f15': {'atributo': 'BOM'}, 'fbz_gld_bonif_suarenda_f2': {'atributo': 'BOM'}, 'fbz_noa_joogos_f3': {'atributo': 'EXCELENTE'}, 'fbz_zed_bonif_revista_f25': {'atributo': 'ACEITE', 'nodes': {'alto': {'atributo': 'EXCELENTE'}, 'normal': {'atributo': 'RUIM'}}}, 'fbz_mov_starwars_f4': {'atributo': 'ACEITE', 'nodes': {'alto': {'atributo': 'EXCELENTE'}, 'normal': {'atributo': 'BOM'}}}, 'fbz_zed_recompensa_f16': {'atributo': 'BOM'}, 'fbz_zed_sonhopremio_f1': {'atributo': 'ACEITE', 'nodes': {'alto': {'atributo': 'EXCELENTE'}, 'normal': {'atributo': 'BOM'}}}, 'fbz_mov_bonif_marvel_f4': {'atributo': 'ACEITE', 'nodes': {'alto': {'atributo': 'BOM'}, 'normal': {'atributo': 'RUIM'}}}, 'fbz_mov_playkids_f14': {'atributo': 'ALCANCE', 'nodes': {'alto': {'atributo': 'EXCELENTE'}, 'normal': {'atributo': 'BOM'}, 'baixo': {'atributo': 'EXCELENTE'}}}, 'fbz_ups_saudeup_t1': {'atributo': 'EXCELENTE'}, 'fbz_twe_appgame_f1': {'atributo': 'BOM'}, 'fbz_zed_revista_f13': {'atributo': 'BOM'}}}

# Resultados: Árvore de Decisão



# Resultados: Desempenho

| Threads | Execução (min) |
|---|---|
| 1 | 5.3 min |
| 2 | 5.0 min |
| 3 | 4.9 min |
| 4 | 4.5 min |

# Conclusão

- Classificação de base de dados com 33 milhões de linhas em menos de cinco minutos.
- Classificação das campanhas com base no desempenho e outros atributos de relevância.
- RDD permite paralelismo e diminuição no tempo de execução do algoritmo.

# OBRIGADO