Engenharia de Computação

Especialização Lato Sensu em Ciência de Dados e Analytics

Aula Prática

{ MapReduce }

Prof. Jairson Rodrigues
jairson.rodrigues@univasf.edu.br

{ prática }

Entrada: seis livros em formato txt obtidos através do projeto Gutenberg - https://www.gutenberg.org/

prática 1: wordcount "turbinado" + topk - http://bit.ly/2m9AMUz

prática 2: TF-IDF

```
$ cp /vagrant/resources/tfidf.sh ~/
```

prática 3: stop words

```
$ cp /vagrant/resources/stopwords.sh ~/
```

{ exercício - wordcount }

Através de jobs MapReduce, considerando o conjunto de textos do projeto gutenberg (seis livros) retorne as 20 palavras mais relevantes, com mais de quatro caracteres, em ordem decrescente, sem diferenciar maiúsculas ou minúsculas, excluindo *stop words*.

{ wc_map_v1.py }

```
#!/usr/bin/env python
import sys

# toma linhas da entrada padrao
for line in sys.stdin:
    # remove espacos em branco
    line = line.strip()
    # divide linha em palavras
    words = line.split()

    for word in words:
        # escreve resultado na saida padrao
        # que sera entrada para a fase reduce
        # delimitador: tab
        print '%s\t%s' % (word, 1)
```

{ executando jobs }

```
#!/bin/bash

# job MapReduce / Word Count v1.
# Prof. Jairson Rodrigues
# Especialização em Ciência de Dados e Analytics

cd ~/code/wc

export STREAM_JAR=/home/vagrant/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.7.4.jar

hadoop fs -rm -r /poli/output/wc1

hadoop jar $STREAM_JAR      \
 -files wc_map_v1.py,wc_reduce.py      \
 -mapper wc_map_v1.py      \
 -reducer wc_reduce.py      \
 -input /poli/gutenberg      \
 -output /poli/output/wc1

hadoop fs -text /poli/output/wc1/part* | more
```

{ wc_map_v1.py }

```
Papelão!        1
Papo.   1
Papéis  1
Para    100
Parado  1
Paraguay,       1
Paraguay;       2
Paraiso.        1
Paraná, 2
Paraná; 1
Pararam 1
Pararam.        1
Parava, 1
Paravam,        1
Pare,   2
Parece  19
```

...limpar palavras!

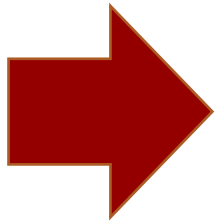
{ exercício - wc_map_v2.py }

- Reescrever o mapper para descartar caracteres não alfabéticos
- Dica: Em python, utilize as funções filter() e isalpha()

  Ex:

  word = "loremipsum;+=]"

  new_word = filter(str.isalpha, word)

resultado  "loremipsum"

...e o reducer?

{ wc_map_v2.py }

```
#!/usr/bin/env python
import sys

for line in sys.stdin:
    # divide a entrada em linhas
    words = line.split()

    for word in words:
        # escreve na saida padrao
        # a entrada para a fase reduce
        # delimitador: \t
        flt = filter(str.isalpha, word)
        if flt != '':
            print '%s\t%s' % (flt, 1)
```

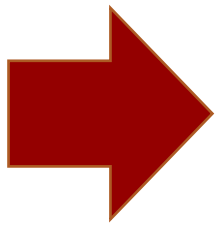
{ exercício - wc_map_v3.py }

- Reescrever o mapper para não diferenciar maiúsculas ou minúsculas
- Dica: Em python, utilize a funcao lower()

  Ex:

  word = "ABCdef"

  word.lower()

  resultado  "abcdef"

{ wc_map_v3.py }

```
#!/usr/bin/env python
import sys

# toma linhas da entrada padrao
for line in sys.stdin:

    # divide linha em palavras
    words = line.split()

    for word in words:
        # escreve resultado na saida padrao
        # que sera entrada para a fase reduce
        # delimitador: tab

        # exclui caracteres nao alfabeticos
        flt = filter(str.isalpha, word)
        if flt != '':
            print '%s\t%s' % (flt.lower(), 1)
```

{ exercício - wc_map_v4.py }

- Eliminando palavas irrelevantes
- Dica:

  Codificação de strings: `# -*- coding: latin-1 -*-`

  Considere a lista a seguir:

```
irrelevantes = ['a', 'à', 'agora', 'ainda', 'alguém', 'algum', 'alguma', 'algumas', 'alguns', 'ampla',
'amplas', 'amplo', 'amplos', 'ante', 'antes', 'ao', 'aos', 'após', 'aquela', 'aquelas', 'aquele', 'aqueles',
'aquilo', 'as', 'até', 'cada', 'com', 'como', 'contra', 'contudo', 'da', 'daquele', 'daqueles', 'das', 'de',
'deve', 'do', 'dos', 'e', 'é', 'e', 'ela', 'elas', 'ele', 'eles', 'em', 'enquanto', 'entre', 'era', 'essa',
'essas', 'esse', 'esses', 'esta', 'está', 'estas', 'este', 'estes', 'for', 'há', 'isso', 'isto', 'já', 'la',
'lá', 'lhe', 'lhes', 'lo', 'mas', 'me', 'na', 'não', 'nas', 'nem', 'nessa', 'nessas', 'nesta', 'nestas',
'no', 'nos', 'nós', 'nossa', 'nossas', 'nosso', 'nossos', 'num', 'numa', 'nunca', 'o', 'os', 'ou', 'para',
'pela', 'pelas', 'pelo', 'pelos', 'per', 'perante', 'pois', 'por', 'porém', 'quais', 'qual', 'quando',
'quanto', 'quantos', 'que', 'quem', 'são', 'se', 'seja', 'sejam', 'sem', 'sendo', 'será', 'serão', 'seu',
'seus', 'si', 'sido', 'só', 'sob', 'sobre', 'sua', 'suas', 'talvez', 'também', 'tampouco', 'te', 'tem',
'tendo', 'tenha', 'ter', 'teu', 'teus', 'ti', 'toda', 'todas', 'todavia', 'todo', 'todos', 'tu', 'tua',
'tuas', 'um', 'uma', 'umas', 'uns', 'vendo', 'ver', 'vez', 'vindo', 'vir', 'vos', 'vós']
```

{ wc_map_v4.py }

```
#!/usr/bin/env python
# -*- coding: latin-1 -*-
import sys
import string

irrelevantes = ['a', 'à', 'agora', 'ainda',... 'vos', 'vós']

def relevante(word):
    if word in irrelevantes or word == "":
        return False
    else:
        return True

for line in sys.stdin:

    words = line.split()

    for word in words:

        flt = filter(str.isalpha, word)
        if relevante(flt) and len(flt) >=5:
            print '%s\t%s' % (flt.lower(), 1)
```

{ saida - wc_map_v4.py / wc_reduce }

```
[...]
claridade      13
clarins         1
claro           4
classe          3
classes         3
classicas       1
classico        2
classicos       3
classifica      1
[...]
```

Contagem de palavras maiores que quatro caracteres, excluídas palavras irrelevantes, sem diferenciar maiúsculas ou minúsculas

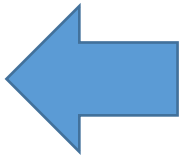
FALTA ORDENAR top 20

{ top k - estratégia }

- Utilizar wc_map_v4.py como mapper
- Criar um reducer que:
  - armazena uma estrutura (key, value) em memória para cada palavra (word) enviada ao reducer
  - incrementa cada contador (by key)
  - ao final, ordena essa lista por (by value / contador) em ordem inversa
  - escreve na saída padrão as primeiras K entradas dessa lista

{ topk_reduce.py }

```
#!/usr/bin/python
import sys
import os

lista = {}

def lista_inversa(lista):
    lista_inversa = []
    for key, value in lista.items():
        lista_inversa.append((value, key))
    return sorted(lista_inversa, reverse = True)

for line in sys.stdin:
    word, count = line.strip().split("\t", 1)
    try:
        count = int(count)
    except ValueError:
        continue

    if word not in lista:
        lista[word] = count
    else:
        lista[word] += count

lista_inversa = lista_inversa(lista)
for word in lista_inversa[:20]:
    print ("%s\t%d" % (word[1], word[0]))
```



função em python

{ saída - topk map/reduce }

- mais458
- alteza 194
- vossa 187
- elle 156
- tinha 137
- ento 133
- muito 132
- meu 128
- foi 127
- ella 126
- depois 122
- minha 114
- ser 112
- homem 112
- vida 108
- tudo 102
- assim 100
- mundo 95
- porque 93
- olhos 93

{ exercício - palavras relevantes }

Através de jobs MapReduce, sobre um conjunto de livros do projeto gutenberg (três livros), exiba as palavras mais relevantes por livro.

Utilize a técnica TF - IDF (Term Frequency / Inverse Document Frequency)

{ TF-IDF }

- Métrica utilizada em algoritmos de recuperação de informação
- Medida estatística utilizada para avaliar quão importante um termo é para um documento dentro de uma coleção de documentos
- Aumenta com o número de vezes que uma palavra aparece em um documento
- Mas é afetada pela frequência da palavra na coleção.
- Muito utilizado em engenhos de busca para *ranking* das páginas web, listas de stop words etc

{ TF-IDF }

- D: Coleção de documentos (corpus)
- d: Documento no corpus
- t: termo
- $f_{(t,d)}$ = frequência do termo t no documento d
- $idf_{(t,D)}$ = frequência inversa do termo t, no corpus D
- TF-IDF = medida estatística da importância do termo

* O algoritmo TF-IDF possui muitas variações.
Os slides e sua implementação foram baseadas em http://spark.apache.org/docs/latest/ml-features.html#tf-idf

{ TF-IDF * }

$$tf_{(t,d)} = \frac{num\ de\ vezes\ que\ o\ termo\ aparece\ no\ documento}{total\ de\ termos\ do\ documento}$$

$$idf_{(t,D)} = log \frac{num\ de\ documentos}{num\ de\ documentos\ que\ possuem\ o\ termo}$$

$$tfidf_{(t,d)} = tf_{(t,d)} * idf_{(t,D)}$$

* O algoritmo TF-IDF possui muitas variações.
Os slides e sua implementação foram baseadas em http://spark.apache.org/docs/latest/ml-features.html#tf-idf

{ TF-IDF / implementação }

- Contabilizar frequência de termos (TF)
- Contabilizar frequência de documentos (DF)
- Contabilizar tamanho do corpus e total de palavras por documento (NTODC)
- Calcular tudo considerando as saídas de DF e NTDOC como entradas

{ tf_map.py }

```
#!/usr/bin/env python
# -*- coding: latin-1 -*-
import sys
import os
import string

irrelevantes = ['a', 'à', ... , 'vos', 'vós']

def relevante(word):
    if word in irrelevantes or word == "":
        return False
    else:
        return True

def tfmapper():
  for line in sys.stdin:
    words = line.strip().split()

    for word in words:
        word_f = filter(str.isalpha, word)
        if relevante(word_f) and len(word_f) >=3:
          print "%s\t%s\t1" % (word_f.lower(), os.getenv('mapreduce_map_input_file','???'))

if __name__ == '__main__':
  tfmapper()
```

{ tf_reduce.py }

```
#!/usr/bin/env python
import sys

def tfreducer():
  prefixo_atual = None
  contador_atual = None

  for line in sys.stdin:
    word,arquivo,contador  = line.strip().split('\t')
    prefixo = '%s\t%s' % (word,arquivo)

    if prefixo_atual == None:
      prefixo_atual = prefixo
      contador_atual = eval(contador)
    elif prefixo_atual == prefixo:
      contador_atual += eval(contador)
    else:
      print "%s\t%s" % (prefixo_atual,contador_atual)
      prefixo_atual = prefixo
      contador_atual = eval(contador)

  print "%s\t%s" % (prefixo_atual,contador_atual)

if __name__=='__main__':
  tfreducer()
```

{ saída tf => map/reduce }

[...]		
carro	.../doc5.txt	2
carro	.../doc3.txt	2
carta	.../doc1.txt	5
carta	.../doc5.txt	17
carta	.../doc3.txt	8
carta	.../doc6.txt	4
cartaxo	.../doc6.txt	1
casa	.../doc3.txt	14
casa	.../doc5.txt	32
casa	.../doc4.txt	4
casa	.../doc1.txt	5
casa	.../doc6.txt	1
[...]		

frequência bruta

termo

documento

{ df_map.py }

```
#!/usr/bin/env python
import sys
import os

def dfmapper():
  for line in sys.stdin:
    print "%s\t1" % line.strip()

if __name__ == '__main__':
  dfmapper()
```

{ df_reduce.py }

```
#!/usr/bin/env python
import sys

def dfreducer():
  word_atual = None
  contador_atual = None
  space = []

  for line in sys.stdin:
    word,arquivo,freq,contador = line.strip().split()
    prefixo = "%s\t%s\t%s" %(word,arquivo,freq)

    if word == None:
      word_atual = word
      contador_atual = eval(contador)
      space.append(prefixo)
    elif word_atual == word:
      contador_atual += eval(contador)
      space.append(prefixo)
    else:
      for item in space:
        print "%s\t%d" % (item,contador_atual)

      word_atual = word
      contador_atual = eval(contador)
      space = [prefixo]

  for item in space:
    print "%s\t%d" % (item,contador_atual)

if __name__=='__main__':
  dfreducer()
```

{ saída df => map/reduce }

```
[ ... ]
carro      .../doc3.txt 2      2
carro      .../doc5.txt 2      2
carta      .../doc6.txt 4      4
carta      .../doc1.txt 5      4
carta      .../doc5.txt 17     4
carta      .../doc3.txt 8      4
cartaxo    .../doc6.txt 1      1
casa       .../doc3.txt 14     5
casa       .../doc5.txt 32     5
casa       .../doc4.txt 4      5
casa       .../doc1.txt 5      5
casa       .../doc6.txt 1      5[ ... ]
```

frequência bruta

#num documentos que contêm o termo

termo

documento

{ ntdoc_map.py }

```
#!/usr/bin/env python
# -*- coding: latin-1 -*-
import sys
import os
import string

irrelevantes = ['a', 'à', ..., 'vos', 'vós']

def relevante(word):
    if word in irrelevantes or word == "":
        return False
    else:
        return True

def ntdocmapper():
  for line in sys.stdin:
    words = line.strip().split()
    for word in words:
        word_f = filter(str.isalpha, word)
        if relevante(word_f) and len(word_f) >=3:
                print "%s\t1" % (os.getenv('mapreduce_map_input_file','???'))

if __name__ == '__main__':
  ntdocmapper()
```

{ ntdoc_reduce.py }

```
#!/usr/bin/env python
import sys

files_list = []
D = 0 # tamanho do corpus

def ntdocreducer():
  curFileName = None
  curcount = None

  for line in sys.stdin:
        filename, count = line.strip().split("\t")

        if (curFileName == None):
                curcount = eval(count)
                curFileName = filename
        elif (curFileName == filename):
                curcount += eval(count)
        else:
                files_list.append((curFileName, curcount))
                curcount = eval(count)
                curFileName = filename

  # append da ultima chave encontrada
  files_list.append((curFileName, curcount))

  D = len(files_list)
  for values in files_list:
        # nome_do_arquivo       total_termos    tam_corpus
        print "%s\t%s\t%s" % (values[0], values[1], D)

if __name__=='__main__':
  ntdocreducer()
```

{ saída ntdoc => map/reduce }

tamanho do corpus

hdfs://192.168.4.221:9000/poli/input/tfidf/doc1.txt	8181	6
hdfs://192.168.4.221:9000/poli/input/tfidf/doc2.txt	4710	6
hdfs://192.168.4.221:9000/poli/input/tfidf/doc3.txt	10117	6
hdfs://192.168.4.221:9000/poli/input/tfidf/doc4.txt	5433	6
hdfs://192.168.4.221:9000/poli/input/tfidf/doc5.txt	12045	6
hdfs://192.168.4.221:9000/poli/input/tfidf/doc6.txt	6354	6

documento

termos por documento

{ TF-IDF / cálculo para o termo: carta }

doc_id:termo	f	tpd	D	tD	TF (f / tpd)	IDF (log D+1/tD+1)	TF-IDF (TF * IDF)
doc1:carta	5	8181	6	4	0,00061	0,14612803567	0,000089
doc2:carta	-	4710	6	4	-	0,14612803567	-
doc3:carta	8	10117	6	4	0,00079	0,14612803567	0,000248
doc4:carta	-	5433	6	4	-	0,14612803567	-
doc5:carta	**17**	**12045**	**6**	**4**	**0,00141**	**0,14612803567**	**0,000457**
doc6:carta	4	6354	6	4	0,00063	0,14612803567	0,000092

* D: tamanho do corpus
* tD: número de documentos onde o termo ocorre
* f: frequência bruta
* tpd: total de termos do documento
* TF: frequência do termo normalizada
* IDF: frequência inversa do termo
* TF-IDF: medida estatística da importância do termo na coleção

{ TF-IDF / cálculo para o termo: carta }

doc_id:termo	f	tpd	TF-IDF
doc5:carta	17	12045	0.000457
doc3:carta	8	10117	0.000248
doc6:carta	4	6354	0.000092
doc1:carta	5	8181	0.000089

{ tfidf_map.py }

```
#!/usr/bin/env python
import sys

for line in sys.stdin:
        line = line.strip()
        splits = line.split("\t")
        doc_id, tpd, D, termo, f, tD = ("_", "_","_", "_", "_","_")

        # doc_id        tpd     D
        # doc1.txt      8181    6
        # doc2.txt      4710    6
        # ...
        if len(splits) == 3:
                doc_id = splits[0]
                tpd = splits[1]
                D = splits[2]

        # termo     doc_id              f       tD
        # carta doc6.txt        4       4
        # carta doc1.txt        5       4
        # carta doc5.txt        17      4
        # ...
        else:
                termo = splits[0]
                doc_id = splits[1]
                f = splits[2]
                tD = splits[3]

        print '%s\t%s\t%s\t%s\t%s\t%s' % (doc_id,tpd,D,termo,f,tD)
    # doc1.txt  8181     6     -1          -1    -1
    # doc1.txt    -1    -1     carta        4     4
    # doc1.txt    -1    -1     casa        12     3
    # doc1.txt    -1    -1     catavento    3     5
    # ...
```

{ tfidf_reduce.py }

```
#!/usr/bin/env python

import sys
import math

tf, idf, tfidf, tpd_atual, D_atual = (0,0,0,0,0)


for line in sys.stdin:
        line = line.strip()
        doc_id,tpd,D,termo,f,tD = line.split("\t")

        # linha de controle: split[3] { termo } = "_"
        if termo == "_": # controle
            tpd_atual = float(eval(tpd))
            D_atual = float(eval(D))
        else: # dados
                f = float(eval(f))
                tD = float(eval(tD))
                tf = f/tpd_atual
                idf = math.log((D_atual+1)/(tD+1), 10)

                tfidf = tf * idf
                print "%s\t%f\t%s" % (termo, tfidf, doc_id)
```

{ último passo / ordenando por termo }

```
#!/usr/bin/env python
import sys

for line in sys.stdin:
        line = line.strip()
        termo,tfidf,doc_id = line.split("\t")

        print '%s\t%s\t%s' % (termo,tfidf,doc_id)
```

map()

reduce()

```
#!/usr/bin/env python
import sys

for line in sys.stdin:
        line = line.strip()
        termo,tfidf,doc_id = line.split("\t")

        print '%s\t%s\t%s' % (termo,tfidf,doc_id)
```

{ saída tfidf/sort => map/reduce }

- carta 0.000092 .../doc6.txt
- carta 0.000457 .../doc5.txt
- carta 0.000248 .../doc3.txt
- carta 0.000089 .../doc1.txt

TF => DF => NTDOC => TFIDF => SORT