Calcul parallèle

Implémentation du TF-IDF en Map/Reduce

Nous avons choisi d'implémenter le TF IDF à l'aide de deux *jobs* Map/Reduce consécutifs.

1er Map/Reduce

mapper.py :

```
#!/usr/bin/python

import sys
import os

finalMap = []

infile = sys.stdin

for line in infile:
    line = line.strip().lower()
    line = (line.replace(".", "")
        .replace("'","")
        .replace("?","")
        .replace(",","")
        .replace("-","")
        .replace("!","")
        .replace("","")
        .replace(";","")
        .replace("\"",""))

    for word in line.split():
        finalMap.append(tuple([os.environ['map_input_file']+'_'+word, 1]))
for i in sorted(finalMap):
    print "%-20s - %d" % (i[0], i[1])
```

Le script ci-dessus lit les données à analyser ligne par ligne. Il commence par nettoyer chaque ligne des caractères spéciaux et stocke, dans la variable intermédiaire finalMap (liste de tuple), la clef (composée du nom du fichier et du mot en question séparés par '_') et une valeur 1.
La sortie ressemble à cela :

```
hdfs://vmhadoopmaster:9000/user/user147/data/csvfile10.csv_adipisci - 1
hdfs://vmhadoopmaster:9000/user/user147/data/csvfile10.csv_adipisci - 1
hdfs://vmhadoopmaster:9000/user/user147/data/csvfile10.csv_adipisci - 1
hdfs://vmhadoopmaster:9000/user/user147/data/csvfile10.csv_adipisci - 1
hdfs://vmhadoopmaster:9000/user/user147/data/csvfile10.csv_adipisci - 1
hdfs://vmhadoopmaster:9000/user/user147/data/csvfile10.csv_adipisci - 1
hdfs://vmhadoopmaster:9000/user/user147/data/csvfile10.csv_adipisci - 1
hdfs://vmhadoopmaster:9000/user/user147/data/csvfile10.csv_adipisci – 1
```

reducer.py :

```
#!/usr/bin/python

# The reducer
from __future__ import division

from operator import itemgetter
import sys

current_key = None
current_count = 0
key = None
finalMap = []
lexie = {}

for line in sys.stdin:
    key, _ = line.split('- ', 1)
    key = key.strip()
    lexie[line.split("_")[0]] = lexie.get(line.split("_")[0], 0) + 1

    if current_key == key:
        current_count += 1
    else:
        if current_key:

            finalMap.append(tuple([current_key, current_count]))

        current_count = 1
        current_key = key

if current_key == key:
    finalMap.append(tuple([current_key, current_count]))

for i in finalMap:
    file = i[0].split("_")[0]
    print "%-20s - %.6f" % (i[0], i[1]/lexie[file])
```

Le premier reducer va agir comme un *word count* par clef (composé du nom du fichier et du mot). La principale différence est que la variable `lexie` va également compter le nombre de mots par fichier. A la fin, le nombre d'occurrences d'un certain mot trouvé dans un document est divisé par le nombre de mots dans ce même document. Cela nous permet d'obtenir le TF. La sortie ressemble à cela :

```
hdfs://vmhadoopmaster:9000/user/user147/data/csvfile2.csv_voluptatemdolorem - 0.000014
hdfs://vmhadoopmaster:9000/user/user147/data/csvfile2.csv_voluptatemeius - 0.000007
hdfs://vmhadoopmaster:9000/user/user147/data/csvfile2.csv_voluptatemest - 0.000014
hdfs://vmhadoopmaster:9000/user/user147/data/csvfile2.csv_voluptatemetincidunt - 0.000021
hdfs://vmhadoopmaster:9000/user/user147/data/csvfile2.csv_voluptatemlabore - 0.000007
hdfs://vmhadoopmaster:9000/user/user147/data/csvfile2.csv_voluptatemneque - 0.000021
hdfs://vmhadoopmaster:9000/user/user147/data/csvfile2.csv_voluptatemnon - 0.000007
hdfs://vmhadoopmaster:9000/user/user147/data/csvfile2.csv_voluptatemquaerat - 0.000028
hdfs://vmhadoopmaster:9000/user/user147/data/csvfile2.csv_voluptatemquisquam - 0.000007
hdfs://vmhadoopmaster:9000/user/user147/data/csvfile2.csv_voluptatemsed - 0.000014
hdfs://vmhadoopmaster:9000/user/user147/data/csvfile2.csv_voluptatemvelit - 0.000007
hdfs://vmhadoopmaster:9000/user/user147/data/csvfile2.csv_voluptatemvoluptatem - 0.000007
```

Nous notons que la sortie est donc triée avec le nom du fichier puis avec le mot. La valeur est le TF du mot dans le document donné.

2ème Map/Reduce

mapper2.py :

```
#!/usr/bin/python

from __future__ import print_function
import sys
import os

finalMap = []

infile = sys.stdin

for line in infile:
    key = line.split(' - ')
    tf = key[1]
    key = key[0]
    splitted_key = key.split("_")
    word = splitted_key[1]
    doc = splitted_key[0]

    finalMap.append(tuple([doc, word, tf, "1"]))

for i in finalMap:
    print (i[1]+'_'+i[0]+"-"+i[2].strip()+"-"+i[3])
```

Le but de ce mapper est d'intervertir le nom du fichier avec le mot afin que la sortie soit triée alphabétiquement par le mot. Nous ajoutons également un 1.
La sortie :

```
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile24.csv-0.000007-1
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile25.csv-0.000006-1
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile26.csv-0.000010-1
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile27.csv-0.000011-1
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile28.csv-0.000006-1
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile29.csv-0.000006-1
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile2.csv-0.000007-1
voluptatemvoluptatem_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile10.csv-0.000007-1
voluptatemvoluptatem_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile11.csv-0.000008-1
voluptatemvoluptatem_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile12.csv-0.000007-1
voluptatemvoluptatem_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile13.csv-0.000006-1
voluptatemvoluptatem_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile14.csv-0.000006-1
```

reducer2.py

```
#!/usr/bin/python

# The reducer
from __future__ import division
from operator import itemgetter
import sys
import math
```

```
current_key = None
current_count = 0
key = None
finalMap = []
document_set = set()
file_tf = []

for line in sys.stdin:

    splitted_line = line.split('_')
    key = splitted_line[0].strip()
    file, tf, _ = splitted_line[1].split("-")
    document_set.add(file)

    if current_key == key:
        current_count += 1
        file_tf.append(tuple([file, tf]))
    else:
        if current_key:
            try :
                for i in range(current_count):
                    finalMap.append(tuple([current_key, file_tf[i][0], file_tf[i][1],
current_count]))
            except:
                print "%s - %r" % (current_count, file_tf)

        current_count = 1
        current_key = key
        file_tf = []
        file_tf.append(tuple([file, tf]))

if current_key == key:
    for i in range(current_count):
        finalMap.append(tuple([current_key, file_tf[i][0], file_tf[i][1], current_count]))

#print finalMap
for i in finalMap:
    number_of_docs = len(document_set)
    print "%s_%s-%s-%s-%s-%.6f-%.12f" % (i[0], i[1], i[2], i[3],
                                                 str(number_of_docs),
                                                 math.log(number_of_docs/int(i[3])),
                                                 float(i[2]) *
math.log(number_of_docs/int(i[3])))
```

Le dernier reducer va réutiliser le principe du *wordcount* afin de déterminer dans combine de documents différents un mot est présent. Cependant nous avons besoin du nombre de documents. Nous avons également besoin de compter dans combien de documents apparaît un mot. Il faut bien sûr conserver le TF qui peut être différent d'un document à l'autre.
La finale :

```
voluptatemtempora_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile10.csv-0.000006-20-22-0.095310-0.000000571861
voluptatemtempora_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile11.csv-0.000004-20-22-0.095310-0.000000381241
voluptatemtempora_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile12.csv-0.000003-20-22-0.095310-0.000000285931
voluptatemtempora_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile13.csv-0.000009-20-22-0.095310-0.000000857792
voluptatemtempora_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile14.csv-0.000007-20-22-0.095310-0.000000667171
voluptatemtempora_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile15.csv-0.000008-20-22-0.095310-0.000000762481
voluptatemtempora_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile16.csv-0.000008-20-22-0.095310-0.000000762481
voluptatemtempora_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile17.csv-0.000006-20-22-0.095310-0.000000571861
voluptatemtempora_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile18.csv-0.000011-20-22-0.095310-0.000001048412
voluptatemtempora_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile19.csv-0.000006-20-22-0.095310-0.000000571861
voluptatemtempora_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile20.csv-0.000004-20-22-0.095310-0.000000381241
voluptatemtempora_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile21.csv-0.000008-20-22-0.095310-0.000000762481
voluptatemtempora_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile22.csv-0.000007-20-22-0.095310-0.000000667171
voluptatemtempora_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile23.csv-0.000004-20-22-0.095310-0.000000381241
voluptatemtempora_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile24.csv-0.000006-20-22-0.095310-0.000000571861
voluptatemtempora_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile25.csv-0.000009-20-22-0.095310-0.000000857792
voluptatemtempora_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile26.csv-0.000008-20-22-0.095310-0.000000762481
voluptatemtempora_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile27.csv-0.000004-20-22-0.095310-0.000000381241
voluptatemtempora_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile28.csv-0.000006-20-22-0.095310-0.000000571861
voluptatemtempora_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile29.csv-0.000008-20-22-0.095310-0.000000762481
voluptatemut_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile10.csv-0.000011-20-22-0.095310-0.000001048412
```

```
voluptatemut_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile11.csv-0.000007-20-22-0.095310-0.000000667171
voluptatemut_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile12.csv-0.000002-20-22-0.095310-0.000000190620
voluptatemut_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile13.csv-0.000009-20-22-0.095310-0.000000857792
voluptatemut_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile14.csv-0.000006-20-22-0.095310-0.000000571861
voluptatemut_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile15.csv-0.000007-20-22-0.095310-0.000000667171
voluptatemut_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile16.csv-0.000006-20-22-0.095310-0.000000571861
voluptatemut_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile17.csv-0.000008-20-22-0.095310-0.000000762481
voluptatemut_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile18.csv-0.000006-20-22-0.095310-0.000000571861
voluptatemut_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile19.csv-0.000010-20-22-0.095310-0.000000953102
voluptatemut_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile20.csv-0.000010-20-22-0.095310-0.000000953102
voluptatemut_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile21.csv-0.000006-20-22-0.095310-0.000000571861
voluptatemut_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile22.csv-0.000007-20-22-0.095310-0.000000667171
voluptatemut_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile23.csv-0.000009-20-22-0.095310-0.000000857792
voluptatemut_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile24.csv-0.000004-20-22-0.095310-0.000000381241
voluptatemut_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile25.csv-0.000006-20-22-0.095310-0.000000571861
voluptatemut_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile26.csv-0.000009-20-22-0.095310-0.000000857792
voluptatemut_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile27.csv-0.000003-20-22-0.095310-0.000000285931
voluptatemut_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile28.csv-0.000003-20-22-0.095310-0.000000285931
voluptatemut_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile29.csv-0.000008-20-22-0.095310-0.000000762481
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile10.csv-0.000009-21-22-0.046520-0.000000418680
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile11.csv-0.000010-21-22-0.046520-0.000000465200
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile12.csv-0.000005-21-22-0.046520-0.000000232600
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile13.csv-0.000006-21-22-0.046520-0.000000279120
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile14.csv-0.000009-21-22-0.046520-0.000000418680
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile15.csv-0.000006-21-22-0.046520-0.000000279120
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile16.csv-0.000006-21-22-0.046520-0.000000279120
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile17.csv-0.000006-21-22-0.046520-0.000000279120
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile18.csv-0.000008-21-22-0.046520-0.000000372160
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile19.csv-0.000006-21-22-0.046520-0.000000279120
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile20.csv-0.000009-21-22-0.046520-0.000000418680
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile21.csv-0.000008-21-22-0.046520-0.000000372160
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile22.csv-0.000008-21-22-0.046520-0.000000372160
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile23.csv-0.000004-21-22-0.046520-0.000000186080
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile24.csv-0.000007-21-22-0.046520-0.000000325640
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile25.csv-0.000006-21-22-0.046520-0.000000279120
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile26.csv-0.000010-21-22-0.046520-0.000000465200
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile27.csv-0.000011-21-22-0.046520-0.000000511720
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile28.csv-0.000006-21-22-0.046520-0.000000279120
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile29.csv-0.000006-21-22-0.046520-0.000000279120
voluptatemvelit_hdfs://vmhadoopmaster:9000/user/user147/data/csvfile2.csv-0.000007-21-22-0.046520-0.000000325640
```

Cette sortie est de la forme :

Mot_fichier-TF-nombre de docs dans lequel le mot est présent-nombre de docs au total-IDF-TF*IDF

La sortie finale permet de faire une recherche sur un mot donné afin de repérer quel fichier a le score le plus élevé.

Implémentation avec PySpark

Nous commençons par générer un fichier de cette forme :

```
Hac ita persuasione reducti intra moenia
Haec igitur Epicuri non probo inquam. De cetero
Quibus ita sceleste patratis Paulus cruore
Horum adventum praedocti speculationibus fidis
Et licet quocumque oculos flexeris feminas adfatim
Et quia Montius inter dilancinantium manus
Quam ob rem ut ii qui superiores sunt submittere
Quis enim aut eum diligat quem metuat aut eum a
Has autem provincias quas Orontes ambiens amnis
Eminuit autem inter humilia supergressa iam
Quam ob rem cave Catoni anteponas ne istum quidem
Pandente itaque viam fatorum sorte tristissima
Circa hos dies Lollianus primae lanuginis
Orientis vero limes in longum protentus et rectum
Intellectum est enim mihi quidem in multis et
Ex turba vero imae sortis et paupertinae in
Altera sententia est quae definit amicitiam
Cuius acerbitati uxor grave accesserat incentivum
Dum haec in oriente aguntur Arelate hiemem agens
Accedebant enim eius asperitati ubi inminuta vel
Constituendi autem sunt qui sint in amicitia fines
Post haec indumentum regale quaerebatur et
Et hanc quidem praeter oppida multa duae civitates
Mensarum enim voragines et varias voluptatum
Utque proeliorum periti rectores primo catervas
Quam quidem partem accusationis admiratus sum et
```

```
Et quoniam mirari posse quosdam peregrinos
Quam ob rem id primum videamus si placet
Superatis Tauri montis verticibus qui ad solis
Auxerunt haec vulgi sordidioris audaciam quod cum
Quid qui se etiam nunc subsidiis patrimonii aut
Cum saepe multa tum memini domi in hemicyclio
Sed cautela nimia in peiores haeserat plagas ut
Latius iam disseminata licentia onerosus bonis
Procedente igitur mox tempore cum adventicium
Dein Syria per speciosam interpatet diffusa
Erat autem diritatis eius hoc quoque indicium nec
Novitates autem si spem adferunt ut tamquam in
Postremo ad id indignitatis est ventum ut cum
Utque aegrum corpus quassari etiam levibus solet
```

PySpark s'attend à recevoir un document par ligne. Contrairement à la première partie où nous avons placé un document par fichier, ici chaque document est une ligne.

Nous plaçons le fichier sur hdfs :

```
user147@vmhadoopmaster:~/project$ hdfs dfs -ls data/csv_pyspark.csv
-rw-r--r--   3 user147 cluster       1910 2019-10-15 13:04 data/csv_pyspark.csv
```

Afin d'implémenter le TF-IDF avec Spark, nous profitons de la bibliothèque pyspark afin d'implémenter le calcul : https://spark.apache.org/docs/2.2.0/mllib-feature-extraction.html

Le code, tiré de la documentation en ligne est le suivant :

```
from pyspark.mllib.feature import HashingTF, IDF

# Load documents (one per line).

documents = sc.textFile("data/csv_pyspark.csv").map(lambda line: line.split(" "))

hashingTF = HashingTF()

tf = hashingTF.transform(documents)

# While applying HashingTF only needs a single pass to the data, applying IDF needs two pas
ses:# First to compute the IDF vector and second to scale the term frequencies by IDF.

tf.cache()

idf = IDF().fit(tf)

tfidf = idf.transform(tf)
```

Nous plaçons le code sur le serveur :

```
from pyspark.mllib.feature import HashingTF, IDF

# Load documents (one per line).
documents = sc.textFile("data/csv_pyspark.csv").map(lambda line: line.split(" "))

hashingTF = HashingTF()
tf = hashingTF.transform(documents)

# While applying HashingTF only needs a single pass to the data, applying IDF needs two passes:
# First to compute the IDF vector and second to scale the term frequencies by IDF.
tf.cache()
idf = IDF().fit(tf)
tfidf = idf.transform(tf)
```

Une fois que ce code a tourné, l'objet tfidf peut être utilisé pour trouver le document le plus approprié pour un mot clef donné :

```
>>> print tfidf
MapPartitionsRDD[12] at mapPartitions at PythonMLLibAPI.scala:1335
```

Afin de trouver le document le plus approprié pour un mot (par exemple pour *admiratus*), nous procédons de cette manière :

```
keywordTF = hashingTF.transform(["admiratus"])
keywordTF
SparseVector(1048576, {668803: 1.0})
keywordHashValue = int(keywordTF.indices[0])
keywordRelevance = tfidf.map(lambda x: x[keywordHashValue])
zippedResults = keywordRelevance.zip(documents)
print zippedResults
org.apache.spark.api.java.JavaPairRDD@61ee9473
print zippedResults.max()
(3.0204248861443626, [u'Quam', u'quidem', u'partem', u'accusationis', u'admiratus', u'sum',
u'et'])
```

Le document le plus *relevant* est donc la ligne suivante (ligne 26) :

```
Quam quidem partem accusationis admiratus sum et
```