Zhihua Zhang & KaiXiang Wang CSOC 560

Homework 2

1. Identifying and removing stop words

The word count mapper function and reducer function are created to get the word count output. The input files include three txt files: Halmet.txt, The_Tragedie_of_Macbeth.txt, and The_Tragedy_of_Romeo_and_Juliet.txt, and the output is in the form of "word count". Since in this step we just need to identify stop words, so we just print stop words out to the output.

mapper_stopwords.py

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

#works_path = r'works/'

# tokenize function
def tokenize(text):
    min_length = 3   # typically, words with length smaller than 3 is not meaningful
    words = map(lambda word: word.lower(), text.strip().split(' '))  # case insensitive
    p = re.compile('^[a-zA-Z]+$')  # remove the mess up
    filtered_words = list(filter(lambda token: p.match(token) and len(token)>=min_length, words))
    return filtered_words

# get the word + count
for line in sys.stdin:
    words = tokenize(line.decode('utf-8'))
    if words:
        for word in words:
            print('%s\t%s' % (word,1))

```

reducer_stopwords.py

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

previous = None
wc_list = {}
sum = 0
# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    key, value = line.split('\t',1)
    # by key (here: word) before it is passed to the reducer
    if key == previous:
        sum += int(value)
    else:
        if previous:
            wc_list[previous] = sum
        sum = int(value)
        previous = key
if previous == key:  # the last word
    wc_list[previous] = sum
sorted_wc_list = sorted(wc_list.items(),key=lambda (k,v):v)
# output stop words (count > 150)
for key,value in sorted_wc_list:
    if value > 150:
        print key + '\t' + str(value)

```

We wrote a tokizen function to get the cleaned words. What the function dose is keeping words that only contains characters, changing them to lowercase and removing those with less than three characters.

```
# tokenize function
def tokenize(text):
    min_length = 3   # typically, words with length smaller than 3 is not meaningful
    words = map(lambda word: word.lower(), text.strip().split(' '))  # case insensitive
    p = re.compile('^[a-zA-Z]+$')  # remove the mess up
    filtered_words = list(filter(lambda token: p.match(token) and len(token)>=min_length, words))
    return filtered_words
```

In order to identify stop words, we simply take the words with count bigger than 150 as stop words, and print them out to the output. In addition, we transfer the output from HDFS to the local file system and name it as stopwords.txt, which will be used in step 2.

```
# output stop words (count > 150)
for key,value in sorted_wc_list:
    if value > 150:
        print key + '\t' + str(value)
```

```
zhihua@namenode:~$ hadoop fs -cat /tmp/stopwords/part-00000
when    151
should  152
must    152
like    153
then    163
she     163
their   166
thee    168
let     174
was     174
hath    176
would   177
more    178
how     182
may     194
project 195
which   204
good    217
enter   220
they    234
him     243
her     245
from    277
shall   283
all     303
thy     305
are     318
our     324
have    326
will    374
what    420
thou    449
your    501
but     566
his     577
for     642
not     696
this    699
with    774
that    946
you     1024
and     2397
the     2907
```

```
# get the stopwords output to local file system and name it to stopwords.txt
hadoop fs -get /tmp/stopwords .
cp stopwords/part-00000 stopwords.txt
```

2. Building the Inverted Index

The mapper_index and reducer_index files are generated to get the inverted index. The output of inverted index results includes word, wordcount, and location (document id and line number).

mapper_index.py

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


# tokenize function
def tokenize(text):
    min_length = 3   # typically, words with length smaller than 3 is not meaningful
    words = map(lambda word: word.lower(), text.strip().split(' '))  # case insensitive
    p = re.compile('^[a-zA-Z]+$')  # remove the mess up
    filtered_words  = list(filter(lambda token: p.match(token) and len(token)>=min_length, words))
    return filtered_words

# get the nonstopwords
word_list =[]
with open("stopwords.txt", 'r') as f:
    for line in f:
        word, count = line.strip().split('\t')
    word_list.append(word)

# get the word+line
fname = None
```

reducer_index.py

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

previous = None
wc = 0
count = 1
doclist = []
# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    key, doc, lc = line.split('\t')
    # by key (here: word) before it is passed to the reducer
    if key == previous:
        count+=1
        doclist.append((doc,lc))
    else:
        if previous:
            print(previous + '\t' + str(count) + "\t" + str(doclist))
            doclist = []
        count=1
        previous = key
        doclist.append((doc,lc))
if previous == key:
    print(previous + '\t' + str(count) + "\t" + str(doclist))

```

When we run the code, the final inverted index can be obtained, which won't contain the stop words (stopwords.txt) identified in Step 1. Then we transfer the output to local file system so that we can query the results locally.

```
# get the nonstopwords
word_list =[]
with open("stopwords.txt", 'r') as f:
    for line in f:
        word, count = line.strip().split('\t')
    word_list.append(word)
```

```
# get the index output to local file system
hadoop fs -get /tmp/index .
```

```
zhihua@namenode:~$ clear
zhihua@namenode:~$ hadoop fs -cat /tmp/index/part-00000 | head -10
abate   2       [('Halmet', '3701'), ('The_Tragedy_of_Romeo_and_Juliet', '3450')]
abatements      1       [('Halmet', '3704')]
abbey   1       [('The_Tragedy_of_Romeo_and_Juliet', '1964')]
abbreviations   1       [('The_Tragedie_of_Macbeth', '355')]
abed    1       [('The_Tragedy_of_Romeo_and_Juliet', '2907')]
abhorred        3       [('The_Tragedy_of_Romeo_and_Juliet', '4196'), ('The_Tragedie_of_Macbeth', '3495'), (
abhors  1       [('The_Tragedy_of_Romeo_and_Juliet', '3100')]
abide   4       [('The_Tragedy_of_Romeo_and_Juliet', '4544'), ('Halmet', '4758'), ('The_Tragedie_of_Macbeth'
abiure  1       [('The_Tragedie_of_Macbeth', '2893')]
able    3       [('The_Tragedy_of_Romeo_and_Juliet', '4382'), ('The_Tragedy_of_Romeo_and_Juliet', '206'), ('
cat: Unable to write to output stream.
zhihua@namenode:~$ hadoop fs -cat /tmp/index/part-00000 | head -20
abate   2       [('Halmet', '3701'), ('The_Tragedy_of_Romeo_and_Juliet', '3450')]
abatements      1       [('Halmet', '3704')]
abbey   1       [('The_Tragedy_of_Romeo_and_Juliet', '1964')]
abbreviations   1       [('The_Tragedie_of_Macbeth', '355')]
abed    1       [('The_Tragedy_of_Romeo_and_Juliet', '2907')]
abhorred        3       [('The_Tragedy_of_Romeo_and_Juliet', '4196'), ('The_Tragedie_of_Macbeth', '3495'), (
abhors  1       [('The_Tragedy_of_Romeo_and_Juliet', '3100')]
abide   4       [('The_Tragedy_of_Romeo_and_Juliet', '4544'), ('Halmet', '4758'), ('The_Tragedie_of_Macbeth'
abiure  1       [('The_Tragedie_of_Macbeth', '2893')]
able    3       [('The_Tragedy_of_Romeo_and_Juliet', '4382'), ('The_Tragedy_of_Romeo_and_Juliet', '206'), ('
aboue   2       [('The_Tragedie_of_Macbeth', '3062'), ('The_Tragedie_of_Macbeth', '2890')]
abound  1       [('The_Tragedie_of_Macbeth', '2859')]
about   53      [('The_Tragedie_of_Macbeth', '1204'), ('The_Tragedie_of_Macbeth', '3214'), ('The_Tragedie_of
57'), ('The_Tragedie_of_Macbeth', '110'), ('The_Tragedie_of_Macbeth', '1201'), ('The_Tragedie_of_Macbeth', '
_Tragedie_of_Macbeth', '2936'), ('The_Tragedie_of_Macbeth', '2402'), ('The_Tragedie_of_Macbeth', '59'), ('Th
Macbeth', '1337'), ('The_Tragedie_of_Macbeth', '1973'), ('Halmet', '4186'), ('Halmet', '4966'), ('Halmet', '
'Halmet', '1659'), ('Halmet', '280'), ('Halmet', '1512'), ('Halmet', '4877'), ('Halmet', '2711'), ('Halmet',
 ('Halmet', '1812'), ('Halmet', '3860'), ('Halmet', '3901'), ('Halmet', '1151'), ('Halmet', '1150'), ('Halme
'), ('Halmet', '1934'), ('Halmet', '5043'), ('The_Tragedy_of_Romeo_and_Juliet', '2292'), ('The_Tragedy_of_Ro
and_Juliet', '4850'), ('The_Tragedy_of_Romeo_and_Juliet', '4831'), ('The_Tragedy_of_Romeo_and_Juliet', '4853
('The_Tragedy_of_Romeo_and_Juliet', '4751'), ('The_Tragedy_of_Romeo_and_Juliet', '574'), ('The_Tragedy_of_Ro
and_Juliet', '4796'), ('The_Tragedy_of_Romeo_and_Juliet', '986'), ('The_Tragedy_of_Romeo_and_Juliet', '1933'
The_Tragedy_of_Romeo_and_Juliet', '2078'), ('The_Tragedy_of_Romeo_and_Juliet', '3955'), ('The_Tragedy_of_Rom
above   11      [('The_Tragedy_of_Romeo_and_Juliet', '861'), ('The_Tragedy_of_Romeo_and_Juliet', '2980'), ('
Tragedy_of_Romeo_and_Juliet', '3283'), ('The_Tragedy_of_Romeo_and_Juliet', '1308'), ('The_Tragedy_of_Romeo_a
uliet', '2367'), ('The_Tragedy_of_Romeo_and_Juliet', '865'), ('The_Tragedie_of_Macbeth', '211'), ('Halmet',
abridgement     1       [('Halmet', '1738')]
absence 2       [('The_Tragedie_of_Macbeth', '2079'), ('The_Tragedie_of_Macbeth', '1849')]
absent  2       [('The_Tragedie_of_Macbeth', '3356'), ('Halmet', '4637')]
absolute        5       [('Halmet', '3958'), ('Halmet', '4291'), ('The_Tragedie_of_Macbeth', '2330'), ('The_
```

3. Query the Inverted Index

A query program on top of your full inverted file index that accepts a user-specified query (one or more words) and returns not only the document IDs but also the locations in the form of line numbers is created.

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

path = r'index/part-00000'

user_input = raw_input("Please input the words you want to query, seperate by space: ")

words = user_input.split(" ")
flag = 0 # check if the word can be found
with open(path, 'r') as f:
    for line in f:
        word, count, location = line.strip().split('\t')
        if word in words:
            print("The number of occurence of %s: %s" % (word, count))
            print("The location of occurence of %s: %s" % (word, location))
            flag = 1
if flag == 0:
    print("Sorry, the words you are querying cannot be found!")
```

```
zhihua@namenode:~$ ./query.py
Please input the words you want to query, seperate by space: hello
Sorry, the words you are querying cannot be found!
zhihua@namenode:~$ ./query.py
Please input the words you want to query, seperate by space: juliet
The number of occurence of juliet: 27
The location of occurence of juliet: [('The_Tragedy_of_Romeo_and_Juliet', '2748'
), ('The_Tragedy_of_Romeo_and_Juliet', '46'), ('The_Tragedy_of_Romeo_and_Juliet'
, '3529'), ('The_Tragedy_of_Romeo_and_Juliet', '2476'), ('The_Tragedy_of_Romeo_a
nd_Juliet', '66'), ('The_Tragedy_of_Romeo_and_Juliet', '1520'), ('The_Tragedy_of
_Romeo_and_Juliet', '4301'), ('The_Tragedy_of_Romeo_and_Juliet', '4495'), ('The_
Tragedy_of_Romeo_and_Juliet', '2939'), ('The_Tragedy_of_Romeo_and_Juliet', '1170
'), ('The_Tragedy_of_Romeo_and_Juliet', '1215'), ('The_Tragedy_of_Romeo_and_Juli
et', '4481'), ('The_Tragedy_of_Romeo_and_Juliet', '1311'), ('The_Tragedy_of_Rome
o_and_Juliet', '3733'), ('The_Tragedy_of_Romeo_and_Juliet', '4260'), ('The_Trage
dy_of_Romeo_and_Juliet', '2982'), ('The_Tragedy_of_Romeo_and_Juliet', '3551'), (
'The_Tragedy_of_Romeo_and_Juliet', '4171'), ('The_Tragedy_of_Romeo_and_Juliet',
'4054'), ('The_Tragedy_of_Romeo_and_Juliet', '19'), ('The_Tragedy_of_Romeo_and_J
uliet', '9'), ('The_Tragedy_of_Romeo_and_Juliet', '2705'), ('The_Tragedy_of_Rome
o_and_Juliet', '2953'), ('The_Tragedy_of_Romeo_and_Juliet', '1490'), ('The_Trage
dy_of_Romeo_and_Juliet', '4397'), ('The_Tragedy_of_Romeo_and_Juliet', '1308'), (
'The_Tragedy_of_Romeo_and_Juliet', '2843')]
```

Extra credit: Integrate queries with Spark

Several codes in the Spark framework are worte to query the inverted index. The results are showed below.

In [1]:
```
# read the data
path = r'HW2/index/part-00000'  #the part of index output file
data = sc.textFile(path)
```

In [14]:
```
# query words you want

query_words = ['abroad', 'able', 'juliet']

filter_rdd = data.filter(lambda x: x.split("\t")[0] in query_words)
results = filter_rdd.collect()
if not results:
    print("Sorry, the words you are querying cannot be found!")
else:
    word_list = filter_rdd.map(lambda x:x.split("\t")[0]).collect()
    for word in query_words:
        if word not in word_list:
            print("The result of word {%s} not found!" %word)
    for res in results:
        word, count, location = res.split("\t")
        print("The number of occurence of {%s}: %s" % (word, count))
        print("The location of occurence of {%s}: %s" % (word, location))
```

```
The result of word {abroad} not found!
The number of occurence of {able}: 3
The location of occurence of {able}: [('The_Tragedy_of_Romeo_and_Juliet', '4382'), ('The_Tragedy_of_Romeo_and_Juliet', '206'),
('Halmet', '4405')]
The number of occurence of {juliet}: 27
The location of occurence of {juliet}: [('The_Tragedy_of_Romeo_and_Juliet', '2748'), ('The_Tragedy_of_Romeo_and_Juliet', '46'),
('The_Tragedy_of_Romeo_and_Juliet', '3529'), ('The_Tragedy_of_Romeo_and_Juliet', '2476'), ('The_Tragedy_of_Romeo_and_Juliet',
'66'), ('The_Tragedy_of_Romeo_and_Juliet', '1520'), ('The_Tragedy_of_Romeo_and_Juliet', '4301'), ('The_Tragedy_of_Romeo_and_Jul
iet', '4495'), ('The_Tragedy_of_Romeo_and_Juliet', '2939'), ('The_Tragedy_of_Romeo_and_Juliet', '1170'), ('The_Tragedy_of_Romeo
_and_Juliet', '1215'), ('The_Tragedy_of_Romeo_and_Juliet', '4481'), ('The_Tragedy_of_Romeo_and_Juliet', '1311'), ('The_Tragedy_
of_Romeo_and_Juliet', '3733'), ('The_Tragedy_of_Romeo_and_Juliet', '4260'), ('The_Tragedy_of_Romeo_and_Juliet', '2982'), ('The_
Tragedy_of_Romeo_and_Juliet', '3551'), ('The_Tragedy_of_Romeo_and_Juliet', '4171'), ('The_Tragedy_of_Romeo_and_Juliet', '405
4'), ('The_Tragedy_of_Romeo_and_Juliet', '19'), ('The_Tragedy_of_Romeo_and_Juliet', '9'), ('The_Tragedy_of_Romeo_and_Juliet',
'2705'), ('The_Tragedy_of_Romeo_and_Juliet', '2953'), ('The_Tragedy_of_Romeo_and_Juliet', '1490'), ('The_Tragedy_of_Romeo_and_J
uliet', '4397'), ('The_Tragedy_of_Romeo_and_Juliet', '1308'), ('The_Tragedy_of_Romeo_and_Juliet', '2843')]
```