**Distributed Data Analytics**

**Exercise 06 – Group 2**

**Nicholas Yegon**

**Matrikel No: 1748461**

1. **Setup**

Download the plain text version of Moby Dick from here: https://www.gutenberg.org/files/2701/2701-0.txt and store it as input file for your MapReduce program.

A screenshot of a computer program

Description automatically generated with medium confidence

Prepare a list of punctuations and a list of stop words for the English language (you can use one from online, or create one yourself)

To capture all punctuation marks and separate them from words and numbers, I used regular expressions ‘**pattern='\W'**’. this will match all characters that are not letters or digits.

I found stop words online and here is my stopwords:

stopwords = [

"a", "an", "and", "as", "at", "be", "by", "for", "from", "has", "he", "in", "is",

"it", "its", "of", "on", "that", "the", "to", "was", "were", "with", "I", "you",

"your", "we", "they", "his", "her", "him", "she", "me", "myself", "ourselves",

"them", "themselves", "ours", "our", "who", "what", "where", "when", "why", "how",

"which", "there", "here"

]

1. **Word importance with text rank**

Write a MapReduce program that archives the following things:

1. Remove punctuation and stopwords from the text.

**Mapper.py:**

import re

import sys

stopwords = [

"a", "an", "and", "as", "at", "be", "by", "for", "from", "has", "he", "in", "is",

"it", "its", "of", "on", "that", "the", "to", "was", "were", "with", "I", "you",

"your", "we", "they", "his", "her", "him", "she", "me", "myself", "ourselves",

"them", "themselves", "ours", "our", "who", "what", "where", "when", "why", "how",

"which", "there", "here"

]

for line in sys.stdin:

line = line.strip()

line = line.lower()

line = re.sub(*pattern*='\W', *repl*=' ', *string*=line)

line = re.sub(*pattern*='\s+', *repl*=' ', *string*=line)

words = line.split()

filteredWords = [word for word in words if word not in stopwords]

line = ' '.join(filteredWords)

print(line)

This code converts input text to lowercase, removes punctuation marks and white spaces from input.txt file using regular expressions, filter out stop words from the text and joins the filtered text.

**Reducer.py:**

import sys

for line in sys.stdin:

line = line.strip()

print(line)

The reducer code reads the output from the mapper, remove any trailing white space and prints the line.

1. Count the number of ”links” for each target word as defined by the Text Rank algorithm.

**Mapper.py**

import re

import sys

from collections import defaultdict

stopwords = [

"a", "an", "and", "as", "at", "be", "by", "for", "from", "has", "he", "in", "is",

"it", "its", "of", "on", "that", "the", "to", "was", "were", "with", "I", "you",

"your", "we", "they", "his", "her", "him", "she", "me", "myself", "ourselves",

"them", "themselves", "ours", "our", "who", "what", "where", "when", "why", "how",

"which", "there", "here"

]

linksCount = defaultdict(*set*)

for line in sys.stdin:

line = line.strip()

line = line.lower()

line = re.sub(*pattern*='\W', *repl*=' ', *string*=line)

line = re.sub(*pattern*='\s+', *repl*=' ', *string*=line)

words = line.split()

filteredWords = [word for word in words if word not in stopwords]

line= ' '.join(filteredWords)

if not line:

continue

tokens = line.split()

numTokens = len(tokens)

for i in range(numTokens):

Word = tokens[i]

linksCount[Word].update(tokens[j] for j in range(i+1, numTokens))

for Word in linksCount:

numLinks = len(linksCount[Word])

print(*f*'{Word}\t{numLinks}')

**Reducer.py**

import sys

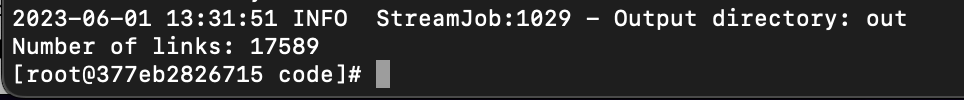
linkCounter = 0

for line in sys.stdin:

linkCounter += 1

print(*f*'Number of links: {linkCounter}')

**output**

****

1. Display every word with a number of links above a threshold of 100.

**Reducer.py:**

import sys

treshold = 100

linkCounter = 0

for line in sys.stdin:

word, count = line.strip().split('\t')

count = *int*(count)

if count > treshold:

print(*f*'{word}\t{count}')

we will define threshold and initialize it to 100 and then do a check if the count is greater than threshold, we display the word with its count.

**output**

A screenshot of a computer

Description automatically generated

Examine the output and iterate a couple of times by extending your list of stopwords and adjusting the threshold until you see meaningful words that seem important for this story.

1. **Extending text rank**

Extend your Text Rank algorithm by allowing bigger tuples of words.

**Mapper.py**

import re

import sys

from collections import defaultdict

stopwords = [

"a", "an", "and", "as", "at", "be", "by", "for", "from", "has", "he", "in", "is",

"it", "its", "of", "on", "that", "the", "to", "was", "were", "with", "I", "you",

"your", "we", "they", "his", "her", "him", "she", "me", "myself", "ourselves",

"them", "themselves", "ours", "our", "who", "what", "where", "when", "why", "how",

"which", "there", "here"

]

linksCount = defaultdict(*set*)

for line in sys.stdin:

line = line.strip()

line = line.lower()

line = re.sub(*pattern*='\W', *repl*=' ', *string*=line)

line = re.sub(*pattern*='\s+', *repl*=' ', *string*=line)

words = line.split()

filteredWords = [word for word in words if word not in stopwords]

line= ' '.join(filteredWords)

if not line:

continue

tokens = line.split()

for n in range(1, len(tokens) + 1):

numTokens = len(tokens)

for i in range(numTokens - n + 1):

wordTuple = *tuple*(tokens[i:i+n])

Word = wordTuple[0]

linksCount[Word].update(wordTuple[1:])

for Word in linksCount:

numLinks = len(linksCount[Word])

print(*f*'{Word}\t{numLinks}')

Provide a qualitative analysis of different sizes of tuples and how that affects the important words.

Different sizes of tuples will impact the importance of word in the output. For example, having a tuple of one word, the output will be independent words without relation to other words. Having a tuple of 2 words means we will be looking at the relations of the words and we can get another broader meaning of the phrase.

1. **Engineering task\***

Write a second MapReduce program that uses the output of Text Rank and sorts the generated tuples by number of links.  
The mapper and reducer will be very simple. The main task is to engineer a bash script that executes the two MapReduce jobs sequentially and use that script inside the Docker Image.

**Secondmapper.py**

import sys

BLUE = '\033[34m'

RESET\_COLOR = '\033[0m'

for line in sys.stdin:

line = line.strip()

words, numLinks = line.split('\t')

numLinks = *int*(numLinks)

colored\_words = *f*'{BLUE}{words}{RESET\_COLOR}'

print(*f*'{numLinks}\t{colored\_words}')

**secondreducer.py**

import sys

tuples = []

for line in sys.stdin:

line = line.strip()

numLinks, words = line.split('\t')

try:

numLinks = *int*(numLinks)

tuples.append((numLinks, words))

except *ValueError*:

continue

*#sort tuples by numLinks*

sortedTuples = sorted(tuples)

for numLinks, words in sortedTuples:

print(*f*'{words}\t{numLinks}')

**modified run.sh file**

rm -r out

rm -r sorted\_out

rm -r hdfs

mkdir hdfs

hdfs dfs -put inpt.txt hdfs/inpt.txt

hadoop jar /opt/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.5.jar \

-files mapper.py \

-mapper "python3 mapper.py" \

-file reducer.py \

-reducer "python3 reducer.py" \

-input hdfs/inpt.txt \

-output out

cat out/part-00000

hadoop jar /opt/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.5.jar \

-files secondmapper.py \

-mapper "python3 secondmapper.py" \

-file secondreducer.py \

-reducer "python3 secondreducer.py" \

-input out/part-00000 \

-output sorted\_out | awk '{print "\033[0;32m" $0 "\033[0m"}'

*#cat sorted\_out/part-00000*

echo "First 10 lines:"

head -n 10 sorted\_out/part-00000

echo "Last 10 lines:"

tail -n 10 sorted\_out/part-00000

**output**

**A screenshot of a computer

Description automatically generated with medium confidence**

I have come up with a second MapReduce program that takes the output of the first program which is performing text rank as input and sorts the generated number of links in an ascending order. It then gives an output of the first 10 and the last 10 number of links when sorted. This is just to illustrate the achievement of sorting in ascending order.

I also modified the run file (run.sh) to enable the 2 MapReduce programs to run concurrently.