DATASCI W261: Machine Learning at Scale

David Rose
david.rose@berkeley.edu
W261-1
Week 02
2015.09.11

HW2.0

- Race conditions occur when the execution of separate processes is dependent on the timing of execution of statements within those processes. Problems can occur, often non-deterministically, when the behavior of two or more threads interferes with the intended sequence, resulting in unanticipated behavior. Often this occurs when separate processes require the same resource. Example: updating a global variable. Process A reads the value of variable v, then process B reads the value of v. A increments v and writes the new value. B increments v and writes the new value. The result is that the update by process A has been overwritten by process B. Both processes have executed properly, yet the results are incorrect.
- MapReduce is a data processing concept with two main parts. The map component takes raw input and converts each element of the input into an output value structured as a key-value pair. The reduce component aggregates the output of the map, typically based on the key, and may perform additional calculations on the result.
- MapReduce differs from Hadoop in that MapReduce is a subset of the Hadoop system, which also includes a file system, HDFS, and other infrastructure.
- Hadoop is based on the MapReduce concept, which in turn is based on, but does not strictly adhere to, the concept of functional programming. For a simple example please see exercise HW2.1, below.

HW2.1

In [136]:
```
%%file generate-2.1.py
#!/usr/bin/python
''' generate a list of key-value pairs where the key is a random i
    and the value is an empty string
'''
from __future__ import print_function
from random import randint
minrange = 0
maxrange = 99
randcount = 10000
with open('randomnumbers.txt', 'w') as fout:
    for i in range(0, randcount):
        print(' '.join([str(randint(minrange, maxrange)), '']), fi
```

```
Overwriting generate-2.1.py
```

In [137]:
```
# generate the file of random numbers
!chmod +x *.py
!./generate-2.1.py
```

In [125]:
```
%%file map-2.1.py
#!/usr/bin/python
''' map function copies stdin to stdout '''
from __future__ import print_function
import sys
for line in sys.stdin:
    print(line.strip(), '')

```

```
Overwriting map-2.1.py
```

In [126]:
```
%%file reduce-2.1.py
#!/usr/bin/python
''' reduce function copies stdin to stdout '''
from __future__ import print_function
import sys
for line in sys.stdin:
    print(line.strip())

```

```
Overwriting reduce-2.1.py
```

In [142]:
```

```

```
# HW2.1. Sort in Hadoop MapReduce
!printf "loading random number file\n"
!hdfs dfs -put -f randomnumbers.txt /in
!printf "preparing output directory\n"
!hdfs dfs -rm -r -f -skipTrash /out/out-2.1
!printf "executing yarn task\n"
# specify comparator and option for sorting numerically
!yarn jar /usr/local/Cellar/hadoop/2.7.0/libexec/share/hadoop/tool
    -D mapred.output.key.comparator.class=org.apache.hadoop.mapred
    -D mapred.text.key.comparator.options=-n \
    -files ./map-2.1.py,./reduce-2.1.py \
    -mapper ./map-2.1.py -reducer ./reduce-2.1.py \
    -input /in/randomnumbers.txt -output /out/out-2.1
!printf "checking output directory: "
!hdfs dfs -ls /out/out-2.1
!hdfs dfs -cat /out/out-2.1/part-00000 | head -n 20
!hdfs dfs -cat /out/out-2.1/part-00000 | tail -n 20


```

```
loading random number file
preparing output directory
Deleted /out/out-2.1
executing yarn task
checking output directory: Found 2 items
-rw-r--r--   1 david supergroup          0 2015-09-14 18:52 /out/ou
t-2.1/_SUCCESS
-rw-r--r--   1 david supergroup      39032 2015-09-14 18:52 /out/ou
t-2.1/part-00000
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
cat: Unable to write to output stream.
99
99
99
```

```
99
99
99
99
99
99
99
99
99
99
99
99
99
99
99
99
99
99
```

Discussion: this implementation of a sort is trivial to the extreme, as it relies entirely on the sort capability of Hadoop. Both the map and reduce steps simply copy stdin to stdout. The actual sorting occurs after Hadoop captures the map output and before it presents it as input to the reducer.

If there were > 1 reducer, then additional work would be required. Each reducer would produce a non-overlapping set of sorted key-value pairs (where the value is null in this case). The final step would require concatenating the output of each of the reducers while paying attention to their ordering so that the final output would be a single contiguous ordered list.

Mapper script. This mapper is used for each subsequent exercise.

- in addition to emitting word counts the mapper also emits email classification counts

In [128]:

```
%%writefile map.py
#!/usr/bin/python
''' mapper reads from stdin, emits counts of words, and
    counts of email classifications
'''
from __future__ import print_function
import re
import string
import sys

# regular expression to remove all punctuation
punctuation = re.compile('[%s]' % re.escape(string.punctuation))

# words to be used in analysis
# normalize parameters same as input
findwords = punctuation.sub('', sys.argv[1]).lower().split()

for line in sys.stdin:
    # split line into three tokens: id, classification, email cont
    # both the email subject and the email body are included in th
    tokens = line.split('\t', 2)
    isspam = tokens[1]
    # emit count of email classification, using magic word '__CLAS
    if isspam == '0': # ham
        print('__CLASS__', 1, 0)
    else: # spam
        print('__CLASS__', 0, 1)
    # convert text to lower case
    text = tokens[len(tokens) - 1].lower()
    # remove punctuation from text
    text = punctuation.sub('', text)
    # split into individual words
    words = re.findall(r"[\w']+", text)
    for word in words:
        # only report on word if it is in the word list parameter
        # or report on all words if parameter equals '*'
        if word in findwords or sys.argv[1] == '*':
            # emit the word and the classification count
            if isspam == '0': # ham
                print(word, 1, 0)
            else: # spam
                print(word, 0, 1)

```

```
Overwriting map.py
```

Reducer script for HW2.2. Aggregates results from mapper, emits summed counts of all words processed by mapper.

In [129]:

```
%%writefile reduce-2.2.py
#!/usr/bin/python
'''
    reducer aggregates counts of words from stdin and emits those
'''
from __future__ import print_function
import sys
words = {}
for line in sys.stdin:
    tokens = line.split()
    word = tokens[0]
    if word not in words.keys():
        words[word] = 0
    # mapper produces counts based on email classification
    # this reducer is only interested in total counts
    words[word] += int(tokens[1]) + int(tokens[2])
# emit results
for word in sorted(words.keys()):
    # ignore counts for email classification
    if word != '__CLASS__':
        #print('\t'.join([word, str(words[word])]), file=sys.stder
        print('\t'.join([word, str(words[word])]))

```

```
Overwriting reduce-2.2.py
```

HW2.2

In [130]:

```
# HW2.2. Using the Enron data from HW1 and Hadoop MapReduce stream
# mapper/reducer pair that  will determine the number of occurrenc
# single, user-specified word.
!printf "loading enron email file\n"
!hdfs dfs -put -f enronemail_1h.txt /in
!printf "preparing output directory: "
!hdfs dfs -rm -r -f -skipTrash /out/out-2.2
!printf "executing yarn task\n"
!yarn jar /usr/local/Cellar/hadoop/2.7.0/libexec/share/hadoop/tool
    -files ./map.py,./reduce-2.2.py \
    -mapper './map.py "assistance"' \
    -reducer ./reduce-2.2.py \
    -input /in/enronemail_1h.txt -output /out/out-2.2
!printf "checking output directory: "
!hdfs dfs -ls /out/out-2.2
!printf "result: "
!hdfs dfs -cat /out/out-2.2/part-00000


```

```
loading enron email file
preparing output directory: Deleted /out/out-2.2
executing yarn task
checking output directory: Found 2 items
-rw-r--r--   1 david supergroup          0 2015-09-12 14:36 /out/ou
t-2.2/_SUCCESS
-rw-r--r--   1 david supergroup         14 2015-09-12 14:36 /out/ou
t-2.2/part-00000
result: assistance      10
```

Reducer script for HW2.3-5. Aggregates results from mapper to generate vocabulary and email classification counts.

Test the results against the same data set, classifying email records and comparing the results to known classifications.

In [131]:

```
%%writefile reduce-2.3.py
#!/usr/bin/python
''' reducer aggregates counts of words and email classifications

    reducer then applies a Naive Bayes classifier against the same
    to buld the training parameters, classifying email records and
    results to known classsifications.
'''
from __future__ import print_function
import math
import re
import string
import sys
```

```
14 # store statistics on the original list of words of interest
15 keywords = {}
16 # counts of each email classification
17 hamcount = 0
18 spamcount = 0
19 # counts of words in each email classification
20 spamwordcount = 0
21 hamwordcount = 0
22
23 # minimum word frequency for inclusion
24 minfrequency = sys.maxint
25 try:
26     minfrequency = int(sys.argv[1])
27 except:
28     pass
29
30 for line in sys.stdin:
31     tokens = line.split()
32     word = tokens[0]
33     # special case for count of email classification
34     if word == '__CLASS__':
35         hamcount += int(tokens[1])
36         spamcount += int(tokens[2])
37     # regular case of count of word
38     else:
39         if word not in keywords.keys():
40             keywords[word] = [0, 0]
41         keywords[word][0] += int(tokens[1])
42         keywords[word][1] += int(tokens[2])
43         hamwordcount += int(tokens[1])
44         spamwordcount += int(tokens[2])
45
46 # prune low frequency words from vocabulary
47 if minfrequency < sys.maxint:
48     tmpwords = {}
49     for word in keywords:
50         if sum(keywords[word]) < minfrequency:
51             # decrement word counts accordingly
52             hamwordcount -= keywords[word][0]
53             spamwordcount -= keywords[word][1]
54         else:
55             # keep high frequency words
56             tmpwords[word] = keywords[word]
57     print('prune: removed {} of {} total words due to frequency le
58           .format(len(keywords) - len(tmpwords), len(keywords), mi
59           file=sys.stderr)
60     keywords = tmpwords
61
62 # total number of unique words
63 vocabcount = len(keywords)
64 # total number of email records
65 doccount = spamcount + hamcount
66
67 # counters for determining error rate
```

```
68 correct = 0
69 incorrect = 0
70
71 # regular expression for removing punctuation
72 punctuation = re.compile('[%s]' % re.escape(string.punctuation))
73
74 with open('enronemail_1h.txt', 'r') as cfile:
75     for line in cfile:
76         # words to be used in Naive Bayes classification
77         nbwords = {}
78         tokens = line.split('\t', 2)
79         eid = tokens[0]
80         isspam = tokens[1]
81         # build bag of words for email record
82         text = tokens[len(tokens) - 1].lower()
83         text = punctuation.sub('', text)
84         docwords = re.findall(r"\w+", text)
85         for word in docwords:
86             if word in keywords.keys():
87                 if word not in nbwords:
88                     nbwords[word] = 1
89                 else:
90                     nbwords[word] += 1
91
92         # calculate the probability of the email record being spa
93         # natural log conversion is used to avoid floating point u
94
95         # start with the prior probability of a spam record
96         logpspam = math.log(spamcount / float(doccount))
97         for word in nbwords:
98             # add the probability of the word being present in thi
99             # multiplied by the number of times the word appears i
100            logpspam += (nbwords[word] *
101                (math.log(keywords[word][1] + 1 / float(spamwordco
102
103        # start with the prior probability of a ham record
104        logpham = math.log(hamcount / float(doccount))
105        for word in nbwords:
106            # add the probability of the word being present in thi
107            # multiplied by the number of times the word appears i
108            logpham += (nbwords[word] * (math.log(keywords[word][(
109
110        # determine the classification, based on comparison of log
111        nbclass = '0'
112        if logpspam > logpham:
113            nbclass = '1'
114
115        # add some statistics
116        if isspam == nbclass:
117            correct += 1
118        else:
119            incorrect += 1
120
121        # emit the results
```

```
122         #print('\t'.join([eid, isspam, nbclass, str(isspam == nbc
123         print('\t'.join([eid, isspam, nbclass]))
124 # print some statistics
125 print('correct: {}, incorrect: {}, training error: {}'.format(cor
126     str(float(incorrect) / (correct + incorrect))), file=sys.stde
127
128
```

```
Overwriting reduce-2.3.py
```

HW2.3

In [132]:

```
  1 # HW2.3. Using the Enron data from HW1 and Hadoop MapReduce, write
  2 # mapper/reducer pair that will classify the email messages by a s
  3 # user-specified word.
  4 !printf "loading enron email file\n"
  5 !hdfs dfs -put -f enronemail_1h.txt /in
  6 !printf "preparing output directory: "
  7 !hdfs dfs -rm -r -f -skipTrash /out/out-2.3
  8 !printf "executing yarn task\n"
  9 !yarn jar /usr/local/Cellar/hadoop/2.7.0/libexec/share/hadoop/tool
 10     -files ./map.py,./reduce-2.3.py \
 11     -mapper './map.py "assistance"' \
 12     -reducer ./reduce-2.3.py \
 13     -input /in/enronemail_1h.txt -output /out/out-2.3
 14 !printf "checking output directory: "
 15 !hdfs dfs -ls /out/out-2.3
 16
 17
```

```
loading enron email file
preparing output directory: Deleted /out/out-2.3
executing yarn task
correct: 60, incorrect: 40, training error: 0.4
checking output directory: Found 2 items
-rw-r--r--   1 david supergroup          0 2015-09-12 14:37 /out/ou
t-2.3/_SUCCESS
-rw-r--r--   1 david supergroup       2672 2015-09-12 14:37 /out/ou
t-2.3/part-00000
```

HW2.4

In [133]:

```
# HW2.4. Using the Enron data from HW1 and in the Hadoop MapReduce
# write  a mapper/reducer pair that will classify the email messag
# multinomial Naive Bayes Classifier using a list of one or
# more user-specified words.
!printf "loading enron email file\n"
!hdfs dfs -put -f enronemail_1h.txt /in
!printf "preparing output directory: "
!hdfs dfs -rm -r -f -skipTrash /out/out-2.4
!printf "executing yarn task\n"
!yarn jar /usr/local/Cellar/hadoop/2.7.0/libexec/share/hadoop/tool
    -files ./map.py,./reduce-2.3.py \
    -mapper './map.py "assistance valium enlargementWithATypo"' \
    -reducer ./reduce-2.3.py \
    -input /in/enronemail_1h.txt -output /out/out-2.4
!printf "checking output directory: "
!hdfs dfs -ls /out/out-2.4
```

```
loading enron email file
preparing output directory: Deleted /out/out-2.4
executing yarn task
correct: 63, incorrect: 37, training error: 0.37
checking output directory: Found 2 items
-rw-r--r--   1 david supergroup          0 2015-09-12 14:37 /out/ou
t-2.4/_SUCCESS
-rw-r--r--   1 david supergroup       2672 2015-09-12 14:37 /out/ou
t-2.4/part-00000
```

HW2.5

In [134]:

```
# HW2.5. Using the Enron data from HW1 an in the  Hadoop MapReduce
# write  a mapper/reducer for a multinomial Naive Bayes Classifier
# classify the email messages using  words present. Also drop word
# frequency of less than three (3). How does it affect the misclas
# error of learnt naive multinomial Bayesian Classifiers on the tr
!printf "loading enron email file\n"
!hdfs dfs -put -f enronemail_1h.txt /in
!printf "preparing output directory: "
!hdfs dfs -rm -r -f -skipTrash /out/out-2.5
!printf "executing yarn task\n"
!printf "no minimum frequency for word inclusion in vocabulary\n"
!yarn jar /usr/local/Cellar/hadoop/2.7.0/libexec/share/hadoop/tool
    -files ./map.py,./reduce-2.3.py \
    -mapper './map.py "*"' \
    -reducer './reduce-2.3.py' \
    -input /in/enronemail_1h.txt -output /out/out-2.5
!printf "checking output directory: "
!hdfs dfs -ls /out/out-2.5

```

```
loading enron email file
preparing output directory: Deleted /out/out-2.5
executing yarn task
no minimum frequency for word inclusion in vocabulary
correct: 100, incorrect: 0, training error: 0.0
checking output directory: Found 2 items
-rw-r--r--   1 david supergroup          0 2015-09-12 14:37 /out/ou
t-2.5/_SUCCESS
-rw-r--r--   1 david supergroup       2672 2015-09-12 14:37 /out/ou
t-2.5/part-00000
```

In [135]:
```
# HW2.5. ... Also drop words with a
# frequency of less than three (3). How does it affect the misclas
# error of learnt naive multinomial Bayesian Classifiers on the tr
!printf "preparing output directory: "
!hdfs dfs -rm -r -f -skipTrash /out/out-2.5
!printf "executing yarn task\n"
!printf "setting minimum frequency for word inclusion in vocabular
!yarn jar /usr/local/Cellar/hadoop/2.7.0/libexec/share/hadoop/tool
    -files ./map.py,./reduce-2.3.py \
    -mapper './map.py "*"' \
    -reducer './reduce-2.3.py 3' \
    -input /in/enronemail_1h.txt -output /out/out-2.5
!printf "checking output directory: "
!hdfs dfs -ls /out/out-2.5
```

```
preparing output directory: Deleted /out/out-2.5
executing yarn task
setting minimum frequency for word inclusion in vocabulary to 3
prune: removed 3910 of 5739 total words due to frequency less than 3
correct: 97, incorrect: 3, training error: 0.03
checking output directory: Found 2 items
-rw-r--r--   1 david supergroup          0 2015-09-12 14:37 /out/ou
t-2.5/_SUCCESS
-rw-r--r--   1 david supergroup       2672 2015-09-12 14:37 /out/ou
t-2.5/part-00000
```

Discussion: removing words with a frequency less than 3 results in an increase in training error from 0.0 to 0.03. This would indicate that some of those words, though uncommon, are strongly associated with one classification or the other, but not both, resulting in a more biased model when they are excluded.

In [ ]:
```
1
```