Week 8

The automatic recognition of verb patterns

for checking grammar and providing writing suggestions

April 14, 2018

A example: V by amount

... there this relation between meaning and pattern is inevitable – that meaning and usage have a profound and systematic effect on each other. *–John Sinclair*

- They expect the production of electric cars this year to increase by nearly 20 per cent. (INCREASE/DECREASE)
- The government lost by one vote. (WIND/LOSE)
- The meeting overrun by one hour. (OVERRUN)

Verb patterns and Dictionary

- In Collins Cobuild English Dictionary, each sense entry of is annotated with patterns observed in a corpus
- Other publishers followed suit with similar patterns in different format
- Patterns of 5,000 common verbs (in the Bank of English) are available ()
  -
  -
  -
  -
- Learner's writing errors can be attributed to ...
  -
  -

- –
- –

- Learner's writing errors can be attributed to ...
  - –
  - –
  - –
  - –

Source: Oliver Mason and Susan Hunston (University of Birmingham). The automatic recognition of verb patterns– A feasibility study.

International Journal of Corpus Linguistics 9:2 (2004), 253-270 www.academia.edu/400537/The_Automatic_Recognition_of_Verb_Patterns_A_Feasibility_Study?auto=download

What makes a sentence a good dictionary example?

- In Collins Cobuild English Dictionary, each sense entry of is annotated with patterns observed in a corpus

- Patterns of 5,000 common verbs (in the Bank of English) are available ()

Source: Oliver Mason and Susan Hunston (University of Birmingham). The automatic recognition of verb patterns– A feasibility study.

International Journal of Corpus Linguistics 9:2 (2004), 253-270 www.academia.edu/400537/The_Automatic_Recognition_of_Verb_Patterns_A_Feasibility_Study?auto=download

Preparation

- Use MapReduce framework to do the following:
  - word count
  - ngram count
  - collocations (with distance between a head-collocate pair).
- **MapReduce** (`https://en.wikipedia.org/wiki/MapReduce`) is a programming model for processing **big data** with a **parallel**, **distributed** algorithm on a **cluster** of commodity personal computers.
- A MapReduce program is composed of

  - mapper: performs filtering data and generate (key, value) pair (e.g., key = word, value = count = 1)
  - reducer: performs the summary operation (e.g., counting instances of a word)

- The MapReduce framework (implementation) manages the processing by running mapper and reducer tasks in parallel, controlling all communications and data transfers, and providing for redundancy and fault tolerance.
- For simplicity and convenience, we show how to do MapReduce locally and make use of the multiple CPU cores found in today's personal computer–Local MapReduce

Local MapReduce and Examples

- See `github.com/dspp779/local-mapreduce`
- Usage

```
./lmr <chunk size> <#reducer> <mapper> <reducer> <directory>
```

  - <chunk size>: Split data into chunks with <chunk size>
  - <#reducer>: Each output line from mappers would then be hashed into <num of reducer> different reducer
  - <mapper>, <reducer>: Shell command/Python program

– <directory>: The output directory

Local MapReduce and a Word Count Example

- Mapper and Reducer

```
tr -sc "a-zA-Z" "\n"    (s = Squeeze; c = Complement)
uniq -c                 (c = add Count)
```

- Testing mapper

```
$ echo 'Colorless green ideas \n sleep furiously' | tr -sc "a-zA-Z" "\n"
Colorless
green
ideas
sleep
furiously
```

- Testing reducer

```
$ echo $'Colorless green ideas \n sleep furiously' | tr -sc "a-zA-Z" "\n"
| sort | uniq -c
   1 Colorless
   1 furiously
   1 green
   1 ideas
```

Ngram Count

- Mapper

```
import re, sys

def tokens(str1): return re.findall('[a-z]+', str1.lower())
def ngrams(sent, n):
    return [ ' '.join(x) for x in zip(*[sent[i:] for i in range(n)
        if i <= len(sent) ] ) ]

for line in sys.stdin:
    sent = tokens(line)
    for n in range(2, 6):
        for ngram in ngrams(sent, n):
            print ('%s\t%s' % (ngram, 1))
```

- Testing mapper

```
echo $'Colorless green ideas \n sleep furiously' | python nc-mapper.py

colorless green 1
green ideas 1
colorless green ideas 1
sleep furiously 1
```

- Reducer

```
import sys
from collections import Counter, defaultdict

ngm_count = defaultdict(Counter)
for line in sys.stdin:
    ngm, count = line.split('\t'); n = ngm.count(' ')+1
    ngm_count[n][ngm] += int(count)

for n in range(2, 6):
    for ngm in ngm_count[n]:
        if ngm_count[n][ngm] >= 3:
            print( '%s\t%s' % (ngm, ngm_count[n][ngm]) )
```

- Testing reducer

```
echo $'Colorless green ideas \n sleep furiously' | python nc-mapper.py
| sort | python nc-reducer.py

colorless green 1
green ideas 1
sleep furiously 1
colorless green ideas 1
```

- Running local MapReduce

```
echo $'Colorless green ideas \n sleep furiously'
 | ./lmr 5m 16  'python nc-mapper.py' 'python nc-reducer.py' out

hashing script hashing.py.BWar
 >>> Temporary output directory for mapper created: mapper_tmp.YZ4i
 >>> Mappers running...
 >>> Reducer running. Temporary input directory: mapper_tmp.YZ4i
 >>> Cleaning...
 >>> Temporary directory deleted: mapper_tmp.YZ4i
 * Output directory: out
 * Elasped time: 0:00:02

$ cat out/*
sleep furiously 1
colorless green ideas 1
colorless green 1
green ideas 1
```

- Life-size Test on British National Corpus

```
$ time cat bnc.sent.txt | python nc-mapper.py | sort | python nc-reducer.py

$ grep '^ability ' bnc.ngm.3.plus.txt | sort -k2nr -t $'\t'
ability to pay 108
ability to make 97
ability to cope 64
...
ability range 17
...
ability and willingness 9
...
ability and enthusiasm 6
ability and motivation 6
ability could 6
ability of local 6
ability of the system 6
ability tests 6
```

```
...
ability to conceive and develop 3
ability to conduct 3
ability to construct and convey 3
...
ability to make sense 3
ability to meet the challenges 3
ability to recognise words 3
...
ability to solve problems 3
ability to summon 3
ability to talk and write 3
ability to think logically 3
...
$
```

Extracting Collocations with Local MapReduce

- Mapper

```
from collections import defaultdict, Counter
import sys
from nltk.corpus import stopwords

eng_stopwords = set(stopwords.words('english'))
max_distance = 5
skipbigram = defaultdict(Counter)

for line in sys.stdin:
    ngm, count = line.strip().split('\t')
    ngm = ngm.split(); distance = len(ngm)-1
    skipbigram[ngm[0]+' '+ngm[-1]][distance] += int(count)
    skipbigram[ngm[-1]+' '+ngm[0]][-distance] += int(count)

for bigram in sorted(skipbigram.keys()):
    print('%s\t%s\t%s'%(bigram, sum(skipbigram[bigram].values()),
          sorted(list(skipbigram[bigram].items()), key=lambda x: x[1] )) )
```

- Reducer

```
from math import sqrt
from itertools import groupby
```

```
import sys
k0, U0, k1 = 1, 10, 5
def getHighCounts(list1, COUNT, k):
    if not list1:
        return []
    size = len(list1)
    totals = [ COUNT(x) for x in list1 ]
    grandtotal = sum(totals)
    avg = (0.0+grandtotal)/size
    sdv = sqrt( sum( (x-avg)**2 for x in totals )/size )
    return [ x for x in list1 if COUNT(x) >= avg+k*sdv ]

lines = [ line.strip().split('\t') for line in sys.stdin ]
lines = [ x[0].split()+x[1:] for x in lines]
for head, headgroup in groupby(lines, key=lambda x: x[0]):
    cands = [ (x[0], x[1], int(x[2]), eval(x[3])) for x in headgroup]
    cands.sort(key= lambda x: x[2] )
    goodColls = getHighCounts(cands, lambda x: x[2], k0)
    goodColls = [ (head, coll, total,
                           getHighCounts(dCounts, lambda x: x[1], k1) )
                           for head, coll, total, dCounts in goodColls ]
    for head, coll, total, dCounts in goodColls:
        if dCounts: print('%s\t%s\t%s\t%s' % (head, coll, total, dCounts))
```

Lab Work

- Purpose: Selecting good examples for collocations
- Input:
  - SENTS: a set of sentences
  - COLLS: a set of collocation with distance (e.g., ['difficulty', 'task', 3])
  - PRONS: a list of pronouns, 'i, you, your, yours, he, she, they, him, her, them, his, their, it'
- Output:
  - EXAMPLES: A set of word, col, sentence

- Mapper

  Read a sentence S in SENTS

  For each distance bigram, S[i], S[i+d], where d in [-5,5]

    If *isCollocation*(S[i], S[i+d], d) and $10 \leq |S| \leq 25$,

      Output S[i]_S[i+d] <tab> S

- Reducer

  For all S in each key group of (*Word*, *Col*, *Dist*)

  Compute *Score(S)*

  = location of *Word* − #(words $\in$ S & $\notin$ HiFre-Words)

  − #(words $\in$ S & words $\in$ PRONS)

  Find S^* with the maximum value of *Score*
  Output *Word_Col* <tab> S^*