npfl067 assignment 2

Jaromir Salamon

Saturday, April 30, 2016

1 Source data

Source data are available here:

- TEXTEN1.TXT
- TEXTCZ1.TXT
- TEXTEN1.ptg
- TEXTCZ1.ptg

2 Tasks to do

2.1 Requirements

For all parts of this homework, work alone. On top of the results/requirements specific to a certain part of the homework, turn in all of your code, commented in such a way that it is possible to determine what, how and why you did what you did solely from the comments, and a discussion/comments of/on the results (in a plain text/html) file. Technically, follow the usual pattern (see the Syllabus):

2.2 Best Friends

In this task you will do a simple exercise to find out the best word association pairs using the pointwise mutual information method.

First, you will have to prepare the data: take the same texts as in the previous assignment, i.e.

TEXTEN1.txt and TEXTCZ1.txt

(For this part of Assignment 2, there is no need to split the data in any way.)

Compute the **pointwise mutual information** for all the possible **word pairs appearing consecutively** in the data, **disregarding pairs** in which one or both words appear **less than 10 times in the corpus**, and **sort** the results from the **best to the worst** (did you get any negative values? Why?) Tabulate the results, and show the **best 20 pairs for both data sets**.

Do the same now but for distant words, i.e. words which are at least 1 word apart, but not farther than **50 words** (both directions). Again, tabulate the results, and show the **best 20 pairs for both data sets.**

2.3 Word Classes

- TEXTEN1.ptg
- TEXTCZ1.ptg

These are your data. They are almost the same as the .txt data you have used so far, except they now contain the part of speech tags in the following form:

- rady/NNFS2-----A----
- ,/Z:----------

where the tag is separated from the word by a slash ('/'). Be careful: the tags might contain everything (including slashes, dollar signs and other weird characters). It is guaranteed however that there is no slash-word. Similarly for the English texts (except the tags are shorter of course).

2.3.1 The Task

Compute a **full class hierarchy of words** using the **first 8,000 words** of those data, and **only for words occurring 10 times or more** (use the same setting for both languages). Ignore the other words for building the classes, but keep them in the data for the bigram counts. For details on the algorithm, use the Brown et al. paper distributed in the class; some formulas are wrong, however, so please see the corrections on the web (Class 12, formulas for Trick #4). **Note the history of the merges, and attach it to your homework**. Now run the same algorithm again, but stop when reaching 15 classes. **Print out all the members of your 15 classes and attach them too**.

2.3.2 Hints:

The initial mutual information is (English, words, limit 8000):

- 4.99726326162518 (if you add one extra word at the beginning of the data)
- 4.99633675507535 (if you use the data as they are and are carefull at the beginning and end).

NB: the above numbers are finally confirmed from an independent source :-).

The first 5 merges you get on the English data should be:

- case subject
- cannot may
- individuals structure
- It there
- even less

The loss of Mutual Information when merging the words "case" and "subject":

- Minimal loss: 0.00219656653357569 for case+subject

2.4 Tag Classes

Use the same original data as above, but this time, you will compute the classes for tags (the strings after slashes). Compute tag classes for all tags appearing 5 times or more in the data. Use as much data as time allows. You will be graded relative to the other student's results. Again, note the full history of merges, and attach it to your homework. Pick three interesting classes as the algorithm goes (English data only; Czech optional), and comment on them (why you think you see those tags there together (or not), etc.).

3 Common

Methods implementation of uni-gram, bi-gram and computing count of those elements are part of library.

```
def generateUnigrams(tokens):
    grams = []
    for idx, item in enumerate(tokens[:]):
        grams.append((item))
    return grams

def generateBigrams(tokens, distance):
    grams = []
    for idx, item in enumerate(tokens[:]):
        for i in range(if distance == 1 else 2, distance + 1):
            if(idx+i) > len(tokens)-1: break
            grams.append((item, tokens[idx+i]))
    return grams

def countNgram(ngrams):
    cgrams = {}
    for item in ngrams:
        cgrams[item] = (cgrams[item]+1) if item in cgrams else 1
    return cgrams
```

4 Best Friends

4.1 Requirements:

- Compute the pointwise mutual information for all the possible word pairs appearing consecutively in the data
- Disregarding pairs in which one or both words appear less than 10 times in the corpus
- Sort the results from the best to the worst (did you get any negative values? Why?)
- Tabulate the results, and show the best 20 pairs for both data sets.
- Do the same now but for distant words, i.e. words which are at least 1 word apart, but not farther than 50 words (both directions)

4.2 Output

- Best 20 pairs for CZ language consecutively (best-friends-CZ-close.txt)
- Best 20 pairs for CZ language distant (best-friends-CZ-far.txt)
- Best 20 pairs for EN language consecutively (best-friends-EN-close.txt)
- Best 20 pairs for EN language distant (best-friends-EN-far.txt)

4.3 Computing PMI

Computing PMI of best friends is done by PMI method in common library. It simply calculates PMI according following formula:

$$pmi(x,y) = \log_2\left(\frac{p(x,y)}{p(x)\,p(y)}\right) = log_2\left(\frac{\frac{c(x,y)}{N}}{\frac{c(x)}{N}\frac{c(y)}{N}}\right) = log_2\left(\frac{c(x,y)\,N}{c(x)\,c(y)}\right)$$

PMI method follows above defined rule: “disregarding pairs in which one or both words appears less than 10 times in the corpus” with descendent sort

```
def pmi(unigram, bigram, treshold):
    Nbi = sum(bigram.values()) * 1.0
    Nu0 = sum(unigram.values()) * 1.0
    Nu1 = sum(unigram.values()) * 1.0

    pmi = {}
    for key in bigram.keys():
        if (unigram[key[0]] >= treshold and unigram[key[1]] >= treshold):
            pmi[key] = math.log(1.0 * (bigram[key] / Nbi) / ((unigram[key[0]] / Nu0) *
(unigram[key[1]] / Nu1)),2)

    sorted_pmi = sorted(pmi.items(), key=operator.itemgetter(1), reverse = True)
    return sorted_pmi
```

4.4 Calculation

Calculation of final results is done by following method (w/o prints to screen):

```
def calculate (fileIn, fileOut, distance, enc):
    lines = []
    for line in codecs.open(fileIn, "r", encoding=enc).readlines():
        lines.append(line.strip())

    limit = 20 #limit of output to file

    cUnigram = countNgram(generateUnigrams(lines))
    cBigram = countNgram(generateBigrams5(lines,distance))

    pmi = pmi(cUnigram, cBigram, 10.0) #10 is limit of occurrences

    count = 0
    file = codecs.open(fileOut, "w", encoding ="utf-8")
    for item in pmi:
        count += 1
        if count > limit: break
        file.write(", ".join(item[0]) + "\t" + "{:.9f}".format(item[1])  + '\n')
    file.close()
```

And then 4 calls of this function

```
fileIn  = 'input/TEXTCZ1.txt'
fileOut = 'output/best-friends-CZ-close.txt'
calculate(fileIn, fileOut, 1, "iso8859_2")

fileOut = 'output/best-friends-CZ-far.txt'
calculate(fileIn, fileOut, 50, "iso8859_2")


fileIn  = 'input/TEXTEN1.txt'
fileOut = 'output/best-friends-EN-close.txt'
calculate(fileIn, fileOut, 1, "ascii")

fileOut = 'output/best-friends-EN-far.txt'
calculate(fileIn, fileOut, 50, "ascii")
```

4.5 Results

For bi-gram setup of distance 1, this means close pairs and for distance 50, this means far pairs we have next results of 20 top pairs for English and Czech language corpuses.

4.5.1 PMI for close pairs in English text

Pair	PMI
La, Plata	14.16937
Asa, Gray	14.03187
Fritz, Muller	13.36202
worth, while	13.33287
faced, tumbler	13.26248
lowly, organised	13.2169
Malay, Archipelago	13.11048
shoulder, stripe	13.05389
Great, Britain	12.91456
United, States	12.84744
English, carrier	12.52551
specially, endowed	12.40182
branched, off	12.37736
Sir, J	12.37736
de, Candolle	12.36202
mental, qualities	12.36202
Galapagos, Archipelago	12.34494
red, clover	12.32388
self, fertilisation	12.31693
systematic, affinity	12.25183

4.5.2 PMI for close pairs in Czech text

Pair	PMI

Hamburger, SV	14.28895
Los, Angeles	14.06244
Johna, Newcomba	13.76288
Č., Budějovice	13.6336
série, ATP	13.46897
turnajové, série	13.43441
Tomáš, Ježek	13.42898
Lidové, noviny	13.32992
Lidových, novin	13.27103
veřejného, mínění	13.06244
teplota, minus	12.98152
jaderné, zbraně	12.95553
Ján, Čarnogurský	12.95553
Milan, Máčala	12.89781
lidských, práv	12.86288
společném, státě	12.70843
akciových, společností	12.69249
Pohár, UEFA	12.62538
privatizačních, projektů	12.61568
George, Bushe	12.60301

4.5.3 PMI for distant pairs in English text

Pair	PMI
dried, floated	8.735396
floated, dried	8.647933
dried, germinated	8.358427
avicularia, vibracula	8.313816
dried, dried	8.303285
eastern, Pacific	8.301843
stripe, shoulder	8.216954
floated, germinated	8.210869
floated, floated	8.192254
layer, hexagonal	8.165781
survival, fittest	8.139786
dried, days	8.097966
heath, heath	8.060835
dimorphic, trimorphic	8.060835
geese, webbed	8.028278
clover, clover	8.026445
floated, days	8.020487

germinated, dried	7.943389
CHAPTER, THE	7.817858
carrier, faced	7.817858

4.5.4 PMI for distant pairs in Czech text

Pair	PMI
výher, výher	9.855552
žel, žel	9.136361
Bělehrad, Benfica	8.951937
h, teplota	8.900406
13h, 13h	8.855552
ODÚ, VPN	8.826406
Sandžaku, Sandžaku	8.826406
Petrof, Petrof	8.785764
Atény, Benfica	8.581987
13h, zataženo	8.575444
13h, skoro	8.54743
CIA, CIA	8.504478
vychází, h	8.485369
výher, IV	8.478483
13h, st	8.456456
pořadí, výher	8.456456
Bělehrad, Kyjev	8.439383
IFS, IFS	8.437841
km, žel	8.435146
silniční, doprava	8.366974

4.6 Conclusion

About PMI calculation are valid following rules according to PMI definition:

$$pmi(x,y) = \log_2\left(\frac{p(x,y)}{p(x)\,p(y)}\right)$$

- When two words are **independent** to each other, then joint probability p(x,y) is equal its multiplication of probability of each p(x) and p(y), so then **PMI = $\log_2(1)$ = 0**

- When two words are shown with **more** probability as pair then each separate appearances, then joint probability p(x, y) > p(x) and p(y) multiplication, so then **PMI = $\log_2$ (>1) = >0**
- When two words are shown with **less** probability as pair then each separate appearances, then joint probability p(x, y) < p(x) and p(y) multiplication, so then **PMI = $\log_2$ (<1) = <0**

4.6.1 PMI for close word pairs

From results of PMI for close pairs we can see in top 20 the names of people and cities or collocations.

4.6.2 PMI for distant word pairs

For distant pairs PMI we can see in top 20 in some cases the same words in pair in rest some lower probable collocations.

5 Word Classes and Tag Classes

5.1 Requirements

5.1.1 Word classes

- Compute a full class hierarchy of words using the first 8,000 words of data
- Only for words occurring 10 times or more (use the same setting for both languages)
- Note the history of the merges, and attach it to your homework.
- Now run the same algorithm again, but stop when reaching 15 classes. Print out all the members of your 15 classes and attach them too.

5.1.2 Tag classes

- Compute the classes for tags
- Compute tag classes for all tags appearing 5 times or more in the data.
- Use as much data as time allows.
- Note the full history of merges, and attach it to your homework.
- Pick three interesting classes as the algorithm goes (English data only; Czech optional), and comment on them (why you think you see those tags there together (or not), etc.).

5.2 Output

5.2.1 Word classes

- History of the merged word classes for CZ language (merged-classes-CZ-words.txt)
- Last 15 word classes for CZ language (merged-classes-CZ-words-15.txt)

- History of the merged word classes for EN language (merged-classes-EN-words.txt)

- Last 15 word classes for EN language (merged-classes-EN-words-15.txt)

5.2.2 Tag classes

- History of the merged tag classes for CZ language (merged-classes-CZ-tags.txt)
- Last 15 tag classes for CZ language (merged-classes-CZ-tags-15.txt)

- History of the merged tag classes for EN language (merged-classes-EN-tags.txt)
- Last 15 tag classes for EN language (merged-classes-EN-tags-15.txt)

5.3 Computing MI

Computing MI for word and tag classes is done by cMI method in library. It simply calculates MI according following formula (calculates MI for one words or tags pair):

$$I(x,y) = p(x,y)\ \log_2\left(\frac{p(x,y)}{p(x)\,p(y)}\right) = \frac{c(x,y)}{N} log_2\left(\frac{\frac{c(x,y)}{N}}{\frac{c(x)}{N}\frac{c(y)}{N}}\right) = \frac{c(x,y)}{N} log_2\left(\frac{c(x,y)\,N}{c(x)\,c(y)}\right)$$

```
def cmi(N, cx, cy, cxy):
    return ((cxy / N) * math.log(1.0 * (cxy * N) / (cx * cy), 2)) if (1.0 * (cxy * N)
/ (cx * cy) > 0) else 0
```

Total MI over all words or tags is then done in calcQ method.

5.4 Computing Subterms, Substractions and Minimal loss

According to slide 134 are calculated following methods for support Word and Tag Classes task.

Computing subterms and total MI:

```
def calcQ(cUnigram, cBigram, N, q):
    mi = 0
    for key in cBigram.keys():
        q[key] = cmi(N, cUnigram[key[0]], cUnigram[key[1]], cBigram[key])
        mi += q[key]
    return mi, q
```

Computing substractions:

```
def calcS(uniqueTerms, cUnigramL, cUnigramR, q):
    s = {}
    for word in uniqueTerms:
        # Note: q[word,word] doesn't exist in this implementation! Anyway it is
unigram with MI which is log2(1) = 0
        s[word] = 0
        s[word] = sum(t((wordL, word), q) for wordL in cUnigramL.keys()) + \
                  sum(t((word, wordR), q) for wordR in cUnigramR.keys())
    return s
```

And computing minimal loss:

```
def calcL(uniqueTerms, cUnigramL, cUnigramR, cBigram, N, q, s):
    minl = 1000.0
    l = {}

    for a in range(0, len(uniqueTerms)):
        wordA = uniqueTerms[a]
```

```
        for b in range(a + 1, len(uniqueTerms)):
            wordB = uniqueTerms[b]

            sumL = 0
            sumR = 0

            cUnigramRAB = cUnigramR[wordA] + cUnigramR[wordB]

            for wordL in cUnigramL.keys():
                if not (wordL == wordA or wordL == wordB):
                    cBigramLAB = t((wordL,wordA),cBigram) + t((wordL,wordB),cBigram)
                    sumL = sumL + cmi(N, cUnigramL[wordL], cUnigramRAB, cBigramLAB)

            cUnigramLAB = cUnigramL[wordA] + cUnigramL[wordB]

            for wordR in cUnigramR.keys():
                if not (wordR == wordA or wordR == wordB):
                    cBigramRAB = t((wordA,wordR),cBigram) + t((wordB,wordR),cBigram)
                    sumR = sumR + cmi(N, cUnigramR[wordR], cUnigramLAB, cBigramRAB)


            cBigramAB = t((wordA, wordA), cBigram) + t((wordA, wordB), cBigram) + \
                        t((wordB, wordA), cBigram) + t((wordB, wordB), cBigram)
            l[(wordA, wordB)] = s[wordA] + s[wordB] - \
                                t((wordA, wordB), q) - \
                                t((wordB, wordA), q) - \
                                cmi(N,cUnigramLAB,cUnigramRAB,cBigramAB) - sumL - sumR


            if (l[(wordA, wordB)] < minl):
                minl = l[(wordA, wordB)]
                minWordA = wordA
                minWordB = wordB

    return minl, minWordA, minWordB
```

5.5 Calculation of classes merge

Then also method for classes merge is estabilished:

```
def doClassesMerge(uniqueTerms, cBigram, cUnigramL, cUnigramR, classes, wordA, wordB):
    # merge counts of unigram wordA and unigram wordB
    # into unigram wordA and delete unigram wordB
    cUnigramL[wordA] = cUnigramL[wordA] + cUnigramL[wordB]
    del cUnigramL[wordB]
    cUnigramR[wordA] = cUnigramR[wordA] + cUnigramR[wordB]
    del cUnigramR[wordB]

    # merge classes
    classes[wordA] = classes[wordA] + "+" + classes[wordB]
    del classes[wordB]

    # update uniqueTerms
    uniqueTerms.remove(wordB)

    # merge counts of bigram
    cBigramAux = cBigram.copy()
    for key in cBigram.keys():
        if key[0] == wordB:
            if key[1] == wordB:
                '''key[0],key[1] = minWordB,minWordB'''
                if(t((wordB,wordB),cBigramAux)>0)and(t((wordA,wordA),cBigramAux)>0):
```

```
                    cBigramAux[(wordA, wordA)] += cBigramAux[(wordB, wordB)]
                    del cBigramAux[(wordB, wordB)]
                else:
                    cBigramAux[(wordA, wordA)] = 0
            else:
                '''key[0],key[1] = minWordB,key[1]'''
                if(t((wordB,key[1]),cBigramAux)>0)and(t((wordA,key[1]),cBigramAux)>0):
                    cBigramAux[(wordA, key[1])] += cBigramAux[(wordB, key[1])]
                    del cBigramAux[(wordB, key[1])]
                else:
                    cBigramAux[(wordA, key[1])] = 0
        else:
            if key[1] == wordB:
                '''key[0],key[1] = key[0],minWordB'''
                if(t((key[0],wordB),cBigramAux)>0)and (t((wordA,wordB),cBigramAux)>0):
                    cBigramAux[(wordA, wordB)] += cBigramAux[(key[0], wordB)]
                    del cBigramAux[(key[0], wordB)]
                else:
                    cBigramAux[(wordA, wordB)] = 0
                # else:
                '''key[0],key[1] = key[0],key[1] - not necessary to care :-)'''

    return uniqueTerms, cBigramAux, cUnigramL, cUnigramR, classes
```

5.6 Calculation

Main calculation is then done according to process described in slides 133 – 136:

```
def calculate(fileIn, fileOut, limit, variant, enc):
    # data structures
    mi = 0                # mutual information
    uniqueTermsLen = 0    # unique(i)
    cUnigram = {}         # ck(i)   unigram counts
    cBigram = {}          # ck(i,j) bigram counts
    cUnigramL = {}        # ckl(i)  unigram counts left
    cUnigramR = {}        # ckr(j)  unigram counts right
    q = {}                # qk(i,j) subterms
    s = {}                # sk(a)   substractions
    l = {}                # Lk(a,b) table of losses
    classes = {}          # classes
    occurringLimit = 10 # for words 10 occurencies limit, for tags 5

    lines_temp = []
    lines = []
    fileName, fileExt = os.path.splitext(fileOut)

    for line in codecs.open(fileIn, "r", encoding=enc).readlines():
        if variant == "words":
            lines_temp.append(line.strip().split("/")[0])
        else:
            lines_temp.append(line.strip().split("/")[1])
    lines = lines_temp[0:limit]

    if variant == "words":
        occurringLimit = 10
    else:
        occurringLimit = 5

    N = len(lines) * 1.0

    cUnigram = countNgram(generateUnigrams(lines))
    cBigram = countNgram(generateBigrams(lines,1))
```

```
    cUnigramL = {}
    cUnigramR = {}
    for key in cBigram.keys():
            cUnigramL[key[0]] = cUnigram[key[0]]
            cUnigramR[key[1]] = cUnigram[key[1]]

    for item, count in cUnigramR.items():
        if count >= occurringLimit:
            classes[item] = item

    uniqueTerms = list(set(classes.keys()))

    file = codecs.open(fileName + '-' + variant + fileExt, "w", encoding = "utf-8")
    while True:
        mi, q = calcQ(cUnigram, cBigram, N, q)
        s = calcS(uniqueTerms, cUnigramL, cUnigramR, q)
        minl, wordA, wordB = calcL(uniqueTerms,cUnigramL, cUnigramR, cBigram, N, q, s)

        uniqueTerms, cBigram, cUnigramL, cUnigramR, classes =
        doClassesMerge(uniqueTerms,cBigram,cUnigramL,cUnigramR,classes,wordA,wordB)

        file.write('for ' + str(len(classes)) + ' classes is mutual information: ' +
         str(mi) + ' minimal loss: ' + str(minl) + ' for ' + t(wordA, classes) + '\n')
        N -= 1
        if len(classes) == 1: break
        if len(classes) == 15: classes15 = classes.copy()
    file.close()

    file = codecs.open(fileName + '-' +variant + '-15' + fileExt,"w",encoding="utf-8")
    i = len(classes15)
    for key, val in classes15.items():
        file.write('class #' + str(i) + ' is: ' + val + '\n')
        i -= 1
    file.close()
```

And then this calculation method is called several times to get particular results are required:

```
fileIn  = 'input/TEXTEN1.ptg'
fileOut = 'output/merged-classes-EN.txt'
calculate(fileIn, fileOut,8000,'words','ascii')
calculate(fileIn, fileOut,-1,'tags','ascii')

fileIn  = 'input/TEXTCZ1.ptg'
fileOut = 'output/merged-classes-CZ.txt'
calculate(fileIn, fileOut,8000,'words','iso8859_2')
calculate(fileIn, fileOut,800,'tags','iso8859_2')
```

5.7 Results

According to hints in task definition I checked following:

1. The initial mutual information is (English, words, limit 8000):
    - Expected: 4.99633675507535 (if you use the data as they are and are careful at the beginning and end).
    - Calculated: 4.99561189953
    - Difference: 0.00072485554535
2. The first 5 merges you get on the English data:
    - Expected:
        - case subject

        - cannot may
        - individuals structure
        - It there
        - even less
    - Calculated:
        - subject+case
        - cannot+may
        - individuals+structure
        - there+It
        - less+even
    - Difference: just order of word pairs in merges
3. The loss of Mutual Information when merging the words "case" and "subject":
    - Expected: 0.00219656653357569
    - Calculated: 0.00219656653358
    - Difference: -0.00000000000000431

Full results are recorder into particular files mentioned in chapter 5.2 Output

5.8 Conclusion

5.8.1 Results tests against hints

The differences investigated in Result chapter are in 1st test small and most likely given by not carefully care about beginning and end of data. In 2nd test only the order of merged words which doesn't play the role because difference in 3rd test is given by round error (precision of calculated number is smaller than expected number).

Based on those I am sure that implementation is correct.

5.8.2 Three interesting English tag classes

- RBS (Adverb, comparative)+JJS (Adjective, superlative) - 25 classes: For example words "most" and "best" can be superlative of adjective, also superlative of adverb.
- PRP (Personal pronoun)+EX (Existential there) – 31 classes: I would expect PRP which could be represent by "I" or "me" is most likely merged with VBx (Verb,…) and also EX which is simple "there" is most likely with VBx.
- VBG (Verb, gerund or present participle) +CD (Cardinal number) – 21 classes: I would expect CD merged first with IN (Preposition or subordinating conjunction) for example "one of", "in two", than with VBG which is mostly connected with another part-of-speech or also IN.