Anagram generation using sentence probabilities

Sven Gestegård Robertz
Department of Computer Science, Lund University
email: sven@cs.lth.se

June 3, 2002

Abstract

An anagram is a word or phrase made by transposing the letters of another word or phrase, and is a popular kind of puzzle. Finding anagrams for a word or a phrase is a difficult task due to the huge number of possible permutations of the letters, and using computers for anagram generation has been increasingly popular. This paper presents an attempt on constructing an automatic anagram generator based on sentence probabilities. Using a relatively small corpus of one million words, a simple implementation using bigram probabilities proved to yield quite satisfactory results but is too inclusive due to the limited context recorded in bigrams. A trigram based version produces much less incorrect results but would need a larger corpus since it is much more exclusive.

1 Introduction

An anagram[1] is a word or phrase made by transposing the letters of another word or phrase and are popular as puzzles. It is also common to make humorous and/or abusive anagrams of names of people and places. Anagrams have also been believed to have mystical or prophetic meaning, especially when applied to religious texts.

Finding anagrams on long words or sentences is very difficult because of the huge number of possible permutations of the letters. Therefore, it is desirable to use computers to search for anagrams. Currently, there exists a large number of anagram generation programs[7]. Many are available on the world wide web both for download and as online services[5, 2]. The common approach is to use a dictionary to limit the amount of possible anagrams to those consisting of actual words. This method yields a huge number of

anagrams, and the automatically generated result must be filtered manually in order to find the ones that actually mean something.

The main problem with automatic anagram finders is to select the anagrams that actually have a meaning (in a certain language) from the huge number of possible permutations of the letters. If we limit ourselves to single-word anagrams the problem is easily solved by dictionary lookup, but if we also want to find multi-word anagrams, the problem not only to find proper words but also to select the series of words that actually have a meaning.

The idea in this work is to reduce the number of false hits by using a statistical method to select the only the most probable sentences.

2 Statistical methods

A *corpus*[9, 4, 10, 8] is a large collection of texts or speech. Corpora are used for statistical analysis of natural language. Two application areas are lexicography[3] and language recognition. In lexicography, corpora are used to find the most common uses of a word in order to select proper examples and citations. In speech and optical character recognition, statistics are used to select the most probable among a set of similarly sounding or looking words given the previous words.

While grammar based methods require detailed knowledge and careful hand-coding, statistical methods are much more flexible and are easy to adapt to different styles of writing (e.g., a scientific corpus and a newspaper corpus in the same language would be quite different) or to different languages, especially with the huge amounts of electronically available text, on the world wide web and electronic libraries.

2.1 Sentence probabilities

For a sentence $S = w_1 w_2 w_3 \dots w_n$ the sentence probability is

$$P(w_1 w_2 \dots w_n) = P(w_1) \times P(w_2|w_1) \times P(w_3|w_1 w_2) \times P(w_n|w_1 \dots w_{n-1})$$

It is, however, not practically possible to use this definition of sentence probability for actual calculations, since it would require an infinitely large corpus in order to accept all correct sentences in a language. Instead, the sentence probability has to be approximated in some way.

2.2 n-gram probabilities

The sentence probability can be approximated by using n-grams, i.e., the conditional probability of a word given the $n-1$ previous words. In this work, we use bigrams and trigrams. The bigram probability of a sentence is

$$P_{bigram}(S) \approx P(w_1) \times P(w_2|w_1) \times \ldots \times P(w_n|w_{n-1})$$

where the probability of a bigram $w_k w_{k+1}$ is calculated as

$$P(w_k w_{k+1}) = P(w_k|w_{k+1}) = \frac{freq(w_k w_{k+1})}{freq(w_k)}$$

The trigram probability is

$$P_{trigram}(S) \approx P(w_1) \times P(w_2|w_1) \times P(w_3|w_1 w_2) \ldots \times P(w_n|w_{n-2} w_{n-1})$$

Using unigram probabilities is equal to dictionary lookup, i.e., all permutations of the words are displayed. The threshold probability effectively sets the size of the dictionary.

2.3 Drawbacks of relying on statistics

The drawbacks of relying entirely on statistics is that it does not take any (formal) semantical knowledge about the language into account. For instance, a certain compound noun may not be very probable in a given corpus, but is a valid anagram anyhow. And a series of words, that are very probable in the corpus, may not be a full sentence and should not be presented as an anagram since they, in isolation, have no meaning.

Thus, relying solely on statistics, without any formal semantic analysis, will yield a too limited system. Such a system will be too inclusive in some respects and too exclusive in others.

3 Implementation

The basic algorithm for generating anagrams is as follows

- The algorithm keeps three strings:

  prev is the start of the anagram that has been accepted as probable,

  head is the start of the word(s) currently being built, and

  rem is the remaining, not yet used, letters.

- for each letter in *rem*, add it to *head*.
  - If it produces a valid word (and *prev* + *head* is a probable series of words), append *head* to *prev* and find recursively find the set of words that can be built from the remaining letters of *rem*.
  - Continue recursively until all letters have been used.

A common way to reduce the number of branches in the recursion tree is to use signatures (the set of characters in a string, rather than the string itself) in the recursive search and then look up all words that match (i.e., can be built from) a certain signature on the way up. I also try to cut of branches that are dead ends early by stopping the search if the `prev` string doesn't contain a probable series of words.

As an example, consider the phrase "think different". In some stage of execution, the algorithm will be in the following state: a word ("fifteenth") has been found, and the search continues, recursively on the remaining letters.

```
(prev,head,rem) = ("fifteenth","", "dikrn")
```

Then we select one letter from *rem* and continue the search recursively. (Since there is only one vowel in *rem*, there is effectively only one path and it isn't interesting to try and build more than one word from the letters of *rem*.)

```
("fifteenth","d", "ikrn")
-->
("fifteenth","di", "krn")
-->
...
-->
("fifteenth","dikrn", "")
```

Now, we look in our dictionary to find any words that can be built from the letters "dikrn". We find the word "drink" and add that to *prev*.

```
(prev,head,rem) = ("fifteenth drink","", "")
```

All the letters have been used, so the recursion stops and we return the result.

Figure 1 shows the algorithm i a little more detail using Java-like pseudo code. The search is done recursively in the method `searchRecursive(prev, head, rem)`.

```
Collection searchRecursive(String prev, String head, String rem){
/* returns collection of strings */

    Collection result;

    if(rem.length() == 1) {
        String candidate = head+rem;

        if(sp.sigExists(candidate)) {
            result = sp.getWordsForSig(candidate);
         } else {
            result = null;
        }
    } else {
        if(hasProbableSentences(prev) ) {
            result = doActualSearchExpand(prev, head, rem, depth);
            result = getProbableSentences(prev,result);
        } else {
          result = null;
        }
   }
    return result;
}

Collection doActualSearchExpand(String prev, String head, String rem){

    Collection result;
    for( each character c in rem ) {
        String newRem = rem - c;
        String newHead = head + c;
        if(hasVowels(newHead)) {
            heads = sp.getWordsForSig(newHead);
        }
        if(heads > 0 && hasVowels(newRem)) {
            String newPrev = concat(prev, newHead);
            theRest = searchRecursive(newPrev, "", newRem);

            result.addAll( allCombinations(heads, theRest));
        }
        tmp = searchRecursive(prev, head+c, s.toString(), depth+1);
            result.addAll(tmp);
        }
   }
   return result;
}

Collection allCombinations(Collection first, collection second){
    return all combinations (x,y) where x in first and y in second
}

boolean sigExists(String signature) {
   return true if any word in the corpus matches the signature, else false
}

Collection getProbableSentences(String head, collection tails){
    Collection result;
    for( each String t in tails) {
        candidate = head + " " +  t;
        if(getSentenceProbability(candidate) > THRESHOLD) {
            result.add(candidate);
        }
    }
    return result;
}
```


Figure 1: The anagram generation algorithm.

4 Experimental results

This section presents the results of sample executions of the program using a Swedish and an English corpus, respectively. The results illustrate the difference between using bigram and trigram probabilities; using trigram probabilities is much more exclusive .

The Swedish corpus was the complete works of Selma Lagerlf (962497 words), and the word to be anagrammed was "universitet". In this case, using bigrams gave 88 anagrams whereas using trigrams only produced one.

The English corpus was randomly selected from the Project Gutenberg electronic library[6] and consisted of Rudyard Kipling, *The Works*; Julian Hawthorne, *The Lock and Key Library/Real Life*; Bennett, *How to Live on 24 Hours a Day*; and O Henry, *Waifs and Strays, etc.* The size was 723910 words (including the Project Gutenberg copyright notice). The phrase to be anagrammed was "anagram generator".

For each anagram, the sentence probability is given (the limit was 10^{-10}).

4.1 Results from using bigram probabilities

```
Anagrams for "universitet":
en ser ut i vit [p=1.9458480510721317E-10]   ser ut i vinet  [p=1.015513774753907E-6]
envis i ett ur  [p=5.142381068564074E-7]     ser ut inte vi  [p=2.9621187688537784E-8]
envis i tu tre  [p=4.440915397008955E-7]     ser ut vi inte  [p=3.0729917387234457E-6]
envist i er ut  [p=1.7160445913500644E-7]    ser vi i ett nu [p=1.8823062145991216E-10]
er nu vi sett i [p=1.937940900390817E-10]    ser vi inte ut  [p=3.7808359616827964E-7]
er vi nu sett i [p=7.308805681473939E-10]    ser vi nu i ett [p=1.1218191968677428E-8]
i ett ur sven i [p=1.5516368737656366E-10]   ser vi ut inte  [p=2.7759636302831487E-9]
ii ett ur sven  [p=3.0035429642629304E-8]    sett i er nu vi [p=2.452672726387112E-10]
in reste ut vi  [p=5.891550112985203E-9]     sitter i nu ve  [p=1.7904648728424079E-9]
in vi reste ut  [p=1.58599764968365E-8]      sven i tu tre i [p=5.816617555780169E-10]
inte ser ut vi  [p=9.478780060332091E-8]     tu tre i en vis [p=7.664376858367229E-10]
inte ut vi ser  [p=5.902352680503676E-9]     tu tre i sven i [p=5.816617555780167E-10]
ner se ut i vit [p=3.96659435500385E-10]     tvi ni se er ut [p=1.4087963583645376E-9]
ni reste ut vi  [p=2.4219565474030946E-8]    tvi ni se ut er [p=1.1592270654237034E-8]
ni ut vi reste  [p=8.126785388846779E-10]    ur sven i ett i [p=2.327455310648455E-10]
ni vi reste ut  [p=1.0866459358890525E-8]    ut i sven i tre [p=1.1661532621133962E-10]
nu i er vi sett [p=1.3060234777380274E-10]   ut inte ser vi  [p=3.4678463635361306E-9]
nu reste i vit  [p=2.0237228371499173E-9]    ut inte vi ser  [p=9.222426063286992E-10]
nu ser vi i ett [p=1.3830647632616007E-9]    ut se er tvi ni [p=1.267722806993553E-10]
nu ve er i sitt [p=9.040149103118745E-10]    ut se i vinter  [p=3.998242714219306E-10]
nu vet i riset  [p=5.242222049351395E-10]    ut vi in reste  [p=5.930663383767719E-10]
nu vi ser i ett [p=1.8627134858742094E-10]   ut vi inte ers  [p=3.735082555631232E-9]
rent ut vi se i [p=1.2379361686304611E-9]    ut vi inte res  [p=3.735082555631232E-9]
res inte ut vi  [p=6.826703269932555E-7]     ut vi inte ser  [p=5.976132089009971E-8]
res nu vi ett i [p=2.004998394252703E-10]    ut vi ni reste  [p=1.2190178083270168E-9]
res nu vi i ett [p=1.8907373450143707E-8]    ut vi reste in  [p=3.911327078251717E-9]
reste nu i vit  [p=3.6932941777986E-8]       ut vi ser inte  [p=7.03244119671731E-8]
reste ut i vin  [p=5.234842067974278E-8]     ve er i sitt nu [p=3.5051123568001315E-9]
reste ut vi in  [p=1.173398123475515E-8]     ve er nu i sitt [p=2.8871976198085495E-8]
```

```
reste ut vi ni  [p=5.866990617377575E-9]     vers i ett nu i [p=5.585713767774757E-9]
riset en ut vi  [p=3.956204617078708E-8]     vers i inte ut  [p=5.218269547053441E-9]
rut inte se vi  [p=1.0177793832489041E-6]    vers i nu i ett [p=1.1401286909778828E-9]
rut inte vi se  [p=2.1318476247657276E-7]    vet nu i riset  [p=1.3154700955091158E-8]
se i tur ni vet [p=1.2782933719464848E-10]   vi in reste ut  [p=3.964994124209125E-9]
se ut er tvi ni [p=7.099247719163898E-9]     vi ni reste ut  [p=8.149844519167894E-9]
se ut i vinter  [p=9.020035563278755E-7]     vi nu ser ett i [p=1.0755405029256881E-10]
sen vi i ett ur [p=3.742883064527547E-10]    vi nu ser i ett [p=3.688172702030936E-9]
sen vi i tu tre [p=3.232321138562728E-10]    vi se ut i rent [p=4.874189357955171E-10]
sen vi ut i ert [p=2.737452226214421E-10]    vi ser inte ut  [p=1.182483675258817E-7]
sen vi ut i tre [p=7.527993622089657E-10]    vi ser nu i ett [p=2.7195616893763467E-9]
ser inte ut vi  [p=8.785210787624866E-8]     vi ser ut inte  [p=7.58763392277394E-8]
ser nu vi i ett [p=5.122462086154077E-10]    vi sitter nu e  [p=3.744814625403478E-9]
ser ut en i vit [p=2.4840613417942104E-10]   vi tre i tusen  [p=2.135511044353802E-9]
ser ut i en vit [p=3.8286992714908027E-7]    vi ut inte ser  [p=7.377940850629595E-10]

Anagrams for "anagram generator":
german or a great an    [p=1.9334758773318616E-7]
german or are a gnat    [p=1.978049467675584E-9]
german or are a tang    [p=6.593498225585281E-10]
german or art an age    [p=6.422454091511403E-9]
manner o a great rag    [p=3.353617069110306E-10]
great a man or anger    [p=1.8626279303618577E-9]
marre no great an ag    [p=2.019265167212665E-9]
```

4.2 Result from using trigram probabilities

```
Anagrams for "universitet":
ser ut i en vit [p=1.186829867641766E-6]

Anagrams for "anagram generator":
*** no anagrams found ***
```

5 Conclusions

The success of any statistical method hinges on having a large enough number of samples. In the experiments done with the implementation presented in this paper, the size of the corpus was limited to about one million words due to memory issues.

Using bigram probabilities produces a reasonably good selection of anagrams. However, since it only requires each pair of words to be probable, it is often too dull an instrument and too inclusive.

Trigram probabilities take more context into account and, hence, should produce better results. However, this also increases the requirements on the corpus. Unfortunately, a corpus size of one million words proved to be too small for getting useful results using trigram probabilities since it is much more exclusive than using bigram probabilities.

A big advantage of statistical methods over formal based ones are that statistical methods are much easier to adapt to new languages or to changes in a language. In this work, both an English and a Swedish corpus were used and both of them worked to an equal degree of satisfaction without any modification to the program itself.

Statistical methods have proven to be very successful in natural language recognition, but it seems that relying solely on statistics and not taking formal semantic knowledge into consideration may be too limited a method for generating sentences. In our experiments, performance was further hampered by using a rather small corpus. This proved to be too inclusive in some respects and too exclusive in others.

Using the semantics captured in sentence probabilities for anagram generation is an improvement over merely using dictionary lookup, and drastically reduce the number of generated anagrams. However, it still produces a large number of results that has to be manually filtered. It also rejects valid but improbable anagrams like, for instance, compound nouns.

References

[1] The anagrammy awards web page. http://www.anagrammy.com/.

[2] Arrak anagram server. http://ag.arrak.fi/index_en.html.

[3] Cobuild home page. http://www.cobuild.collins.co.uk/.

[4] Corpus linguistics web page. http://www.ruf.rice.edu/ barlow/corpus.html.

[5] Internet anagram server. http://www.wordsmith.org/anagram/.

[6] Project gutenberg web site. http://promo.net/pg/.

[7] Word games source code archive. anagram source code. http://www.gtoal.com/wordgames/anagrams.html.

[8] Survey of the state of the art in human language technology. http://cslu.cse.ogi.edu/HLTsurvey/HLTsurvey.html, 1996.

[9] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 2000.

[10] Pierre Nugues. *An Introduction to Language Processing with Prolog*. to be published.