Wild Query & Spelling Correction - IR weekly report 3

hello underworld

8th May 2021

1 Some initial concepts

1.1 Dictionary

The dictionary is a kind of data structure and has two broad classes of solutions: hashing and search tree.The choice of solution (hashing, or search trees) is governed by a number of questions:
(1) How many keys are we likely to have?
(2) Is the number likely to remain static, or change a lot and in the case of changes, are we likely to only have new keys inserted, or to also have some keys in the dictionary be deleted?
(3) What are the relative frequencies with which various keys will be accessed? Actually, each of these solutions takes their advantages and disadvantages.

1.2 Wildcard Query

Wildcard queries are used in any of the following situations:
(1) the user is uncertain of the spelling of a query term (e.g., Sydney vs. Sidney, which leads to the wildcard query S*dney);
(2) the user is aware of multiple vari- ants of spelling a term and (consciously) seeks documents containing any of the variants (e.g., color vs. colour);
(3) the user seeks documents containing variants of a term that would be caught by stemming, but is unsure whether the search engine performs stemming (e.g., judicial vs. judiciary, leading to the wildcard query judicia*);
(4) the user is uncertain of the correct rendition of a foreign word or phrase (e.g., the query Universit*Stuttgart).
Actually, the wildcard is not limited to the end sometimes and appears not only once in the query. So we need a more common solution to this problem.

1.3 Permuterm Index

This is not a bad solution to the Wildcard Query. Permuterm index can be generated in the following way:
(1)Add a $ to the end of each term
(2)§ Rotate the resulting term and index them in a B-tree
Here is an example for the Permuterm Index in Figure 1:
Empirically, dictionary quadruples in size,including as it does all rotations of each term.

1.4 K-gram Index

This is also a solution for Wildcard query. K-gram index can be generated in the following way:
(1)Add a $ to the end and start of each term
(2)Enumerate all k-grams (sequence of k chars) occurring in any term
Here is an example for k-gram in Figure 2:

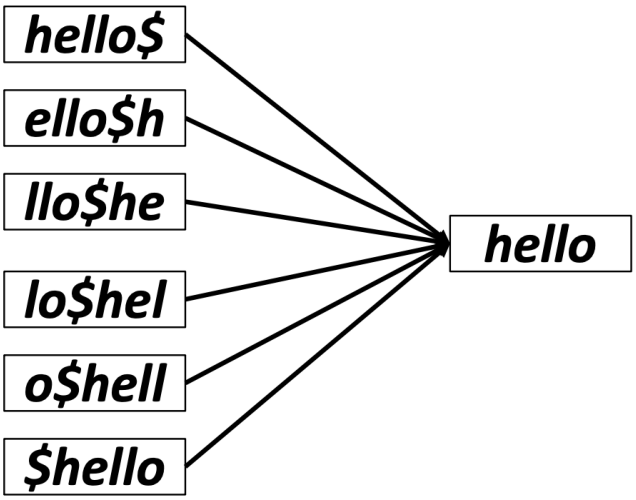

Figure 1: example of "hello" in permuterm index

e.g., from text "***April is the cruelest month***" we get the 2-grams (*bigrams*)

a,ap,pr,ri,il,l,i,is,s,t,th,he,e,$c,cr,ru,
ue,el,le,es,st,t$, m,mo,on,nt,h

Figure 2: example of k-gram (k = 2)

1.5 Edit Distance

Given two character strings s1 and s2, the edit distance between them is the minimum number of edit operations required to transform s1 into s2. Most commonly, the edit operations allowed for this purpose are:
(1) insert a character into a string;
(2) delete a character from a string and
(3) replace a character of a string by another character;
For these operations, edit distance is sometimes known as Levenshtein distance.

2 Some Processes

2.1 Process of the Wild Query

Here are two solutions for the Wild Query, including the permuterm index and k-gram index.

2.1.1 Using the permuterm index

Step1: Introduce a special symbol $ into our character set, to mark the end of a term.
Step2: Construct a permuterm index, in which the various rotations of each term (augmented with $) all link to the original vocabulary term.
Step3: Rotate targeted wildcard query so that the * symbol appears at the end of the string. Here are some formats for the rotation.
Step4: Look up the string(s) in the permuterm index(via the dictionary, usually the search tree). Do some boolean operations with the result.
Step5: Post filter the result against the query.
Step6: Surviving enumerated terms are then looked up in the term-document inverted index.

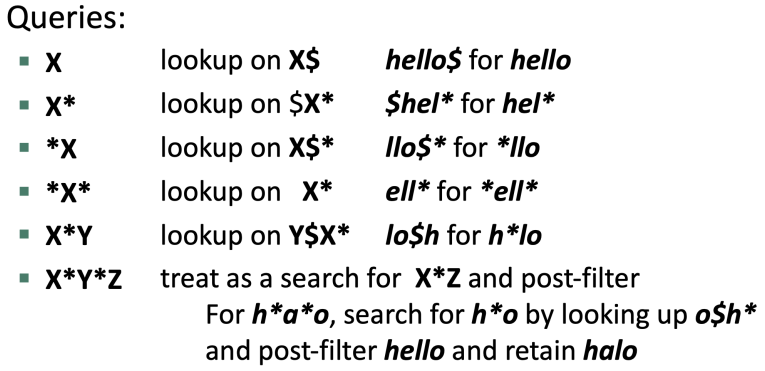

Figure 3: some rotations of Wild Query

2.1.2 Using the k-gram index

Whereas the permuterm index is simple, it can lead to a considerable blowup from the number of rotations per term; for a dictionary of English terms, this can represent an almost ten-fold space increase. We now present a second technique, known as the k-gram index, for processing wildcard queries. Here are the steps:
Step1: Get the K-gram index due to the terms in vocabulary. Here is an example of K-gram index.(k = 2)

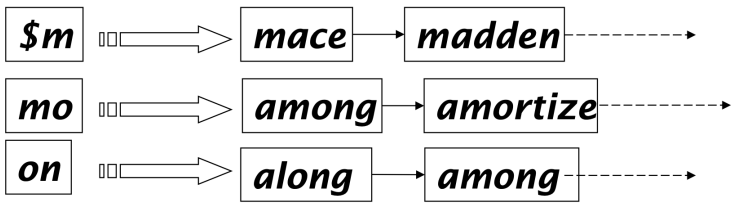

Figure 4: example of k-gram index

Step2: Get the terms matched the K-gram of our Wild Query.
Step3: Gets terms that match AND version of our wildcard query.
Step4: Post-filter these terms against query.
Step5: Surviving enumerated terms are then looked up in the term-document inverted index.

2.2 Spelling Error Detection

For non-word spelling, any word not in a dictionary is an error.
For real-word spelling, the word making a phrase or a sentence no sense is an error.

2.3 Spelling Correction

2.3.1 Generate the candidates

Step1: Run throughdictionary,check edit distance with each word.
Step2: Generate all words within edit distancek (e.g.,k=1 or 2) and then intersect them with dictionary.
Step3: Use a character k-gram index and find dictionary words that share "most" k-grams with word (e.g., by Jaccard coefficient).
ps:Jaccard coefficient for measuring the overlap between two sets A and B, defined to be $(A\&B)/(A|B)$.
Step4: Compute them fast with a Levenshtein finite state transducer
Step5: Have a precomputed map of words to possible corrections

2.3.2 Choose the best one

This process is based on the Noisy Channel Model. Here is the abstract model:

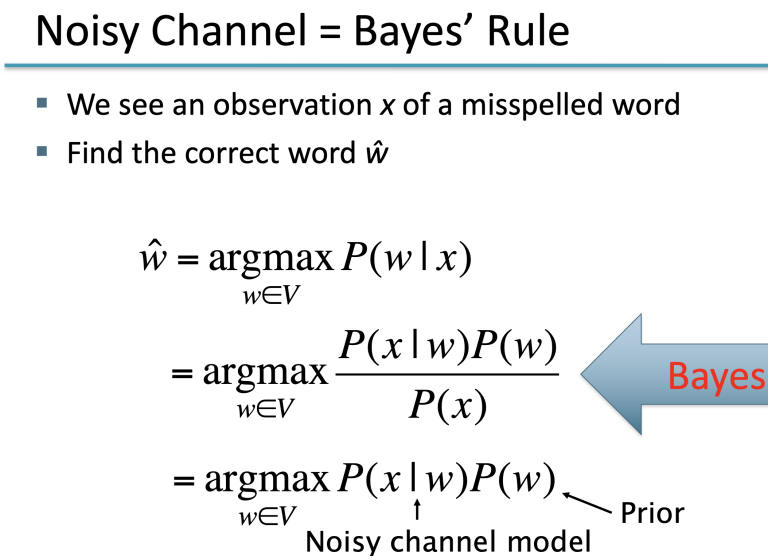

Figure 5: Noisy Channel Model

Step1: Generate the confusion matrix by computing error probability(P(x,w))
Step2: Get the probability of each word(P(w)).
Step3: Choose the best x, which maximize the P(x,w)*P(w).

3 Some Algorithms

3.1 Edit Distance

It is well-known how to compute the (weighted) edit distance between two strings in time O(—s1— × —s2—), where —si— denotes the length of a string si. The idea is to use the dynamic programming algorithm in Figure 6, where the characters in s1 and s2 are given in array form.

```
EDITDISTANCE(s1, s2)
 1  int m[i, j] = 0
 2  for i ← 1 to |s1|
 3  do m[i, 0] = i
 4  for j ← 1 to |s2|
 5  do m[0, j] = j
 6  for i ← 1 to |s1|
 7  do for j ← 1 to |s2|
 8     do m[i, j] = min{m[i − 1, j − 1] + if (s1[i] = s2[j]) then 0 else 1fi,
 9                      m[i − 1, j] + 1,
10                      m[i, j − 1] + 1}
11  return m[|s1|, |s2|]
```

Figure 6: algorithm of Edit Distance