Spell Checking: Edit Distance

VSM, session 8

Northeastern University
College of Computer and Information Science

Slides by: Jesse Anderton

Spell Checking

10-15% of all queries contain spelling errors, so spell checking can help a substantial fraction of users.

A straightforward approach is to replace words not found in a spelling dictionary.

We typically try to find the word from the dictionary with the shortest *edit distance* to the word the user typed.

Example Errors
poiner sisters
brimingham news
catamarn sailing
hair extenssions
marshmellow world
miniture golf courses
psyhics
home doceration

Damerau-Levenshtein Distance

```python
def editDistance(s1, s2):

    # Initialize the table
    d = [[0] * (len(s2) + 1) for _ in range(len(s1) + 1)]
    for i in range(len(d)):
        d[i][0] = i
    for j in range(len(d[0])):
        d[0][j] = j

    # Populate table
    for i in range(1, len(s1) + 1):
        for j in range(1, len(s2) + 1):
            cost = 0 if s1[i - 1] == s2[j - 1] else 1
            d[i][j] = min(
                d[i - 1][j    ] + 1,    # deletion
                d[i    ][j - 1] + 1,    # insertion
                d[i - 1][j - 1] + cost, # substitution
                )
            if (i > 1 and j > 1 and s1[i - 1] == s2[j - 2]
                and s1[i - 2] == s2[j - 1]):
                d[i][j] = min(
                    d[i][j],
                    d[i-2][j-2] + cost, # transposition
                    )
    return d[len(s1)][len(s2)]
```

Damerau-Levenshtein Distance counts the minimum number of insertions, deletions, substitutions, or transpositions to transform one string into another.

- Insertion: extens**s**ions → extensions
- Deletion: poiner → poin**t**er
- Substitution: marshm**e**llow → marshm**a**llow
- Transposition: b**ri**mingham → b**ir**mingham

A dynamic programming algorithm is used to calculate this efficiently.

Example: Edit Distance

```python
def editDistance(s1, s2):

    # Initialize the table
    d = [[0] * (len(s2) + 1) for _ in range(len(s1) + 1)]
    for i in range(len(d)):
        d[i][0] = i
    for j in range(len(d[0])):
        d[0][j] = j

    # Populate table
    for i in range(1, len(s1) + 1):
        for j in range(1, len(s2) + 1):
            cost = 0 if s1[i - 1] == s2[j - 1] else 1
            d[i][j] = min(
                d[i - 1][j    ] + 1,    # deletion
                d[i    ][j - 1] + 1,    # insertion
                d[i - 1][j - 1] + cost, # substitution
                )
            if (i > 1 and j > 1 and s1[i - 1] == s2[j - 2]
                and s1[i - 2] == s2[j - 1]):
                d[i][j] = min(
                    d[i][j],
                    d[i-2][j-2] + cost, # transposition
                    )
    return d[len(s1)][len(s2)]
```

		b	**l**	**a**	**s**	**t**
	0	1	2	3	4	5
b	1					
a	2					
l	3					
k	4					
s	5					

Optimizations

It's not efficient to calculate edit distance between a query term and each word in the spelling dictionary.

- People usually get the first letter of the word right, so we can restrict our search to words starting with the same letter.
- We can restrict our search to words with the same or similar length.
- We can restrict our search to words that *sound the same*, using a phonetic code to group words (such as Soundex).

Soundex

1. Keep the first letter (in upper case).
2. Replace these letters with hyphens: a,e,i,o,u,y,h,w.
3. Replace the other letters by numbers as follows:

    1: b,f,p,v
    2: c,g,j,k,q,s,x,z
    3: d,t
    4: l
    5: m,n
    6: r

4. Delete adjacent repeats of a number.
5. Delete the hyphens.
6. Keep the first three numbers or pad out with zeros.

extenssions → E235; extensions → E235
marshmellow → M625; marshmallow → M625
brimingham → B655; birmingham → B655
poiner → P560; pointer → P536

Developed in the early 20th century, and first patented in 1918.

The idea is to generate a code based how how words sound, so similar-sounding words get the same code.

Many improved algorithms have been developed, but Soundex is still the most common variant in American English.

Commonly supported by database systems, such as Oracle, DB2, MySQL, etc. and used, e.g., for name comparison.

Wrapping Up

It's very common for users to misspell words, so spelling correction has a noticeable impact on query performance.

Given a spelling dictionary, we can employ a quick dynamic programming algorithm on similar-sounding words to find the one that's closest in spelling to what the user typed.

- What if there are multiple candidates with equal minimal edit distance?
- What if the word the user intended is not in the spelling dictionary (e.g. a name)?
- What if the word the user typed *is* in the dictionary, but it's not the word they intended?

Next, we'll look at a probabilistic approach that helps resolve some of these problems.