COSC431
Information Retrieval

Term Conflation
Outline

• Recap
• This week’s problem
• Soundex
• Metaphone
• Spelling Errors (and Americanisms)
• Edit distance
• N-grams
• Stemming
• Thesaurus
• Content Bearing Terms
Recap

• IR is important because:
  – IR is between information and knowledge
  – To control information is to control knowledge
    • “Nam et ipsa scientia potestas est” - Sir Francis Bacon (1597)

• Information needs are not queries
  – Users translate needs into queries, the IR system answers queries

• Analysis
  – Determine performance with respect to need
    • Recall = FoundRelevant/Relevant
    • Precision = FoundRelevant/Found
  – Determine effectiveness of ranking
    • Average precision, Mean Average Precision (MAP), DCG, etc.

• Better ranking puts better results at the top
  – Reduces foraging, increases productivity, increases knowledge
Queries

• IR systems only find documents containing given words

• Boolean
  – Find documents that satisfy an expression (e.g.: a AND b OR c)

• Ranking
  – Find documents containing any of the words (disjunctive ranking)

• Alternatives:
  – Conjunctive ranking
  – Boolean ranking hybrids
The Problem

• Document:
  <doc>Smythe discovered the brains were
diseased with the infectious prion
bovine spongiform encephalopathy,
recently discovered in America.</doc>

• Information need:
  – What disease did Smith find in cow populations in America?

• Query:
  – Smith disease cow America

• Problem:
  – As there are no terms in common between the query and the
document, the IR systems (we have described so far) won’t find
this document

• Consequence:
  – Information is missed, recall is low, precision could be low
The Differences

• In the example:
  – Smith ≠ Smythe
    • This is a difference in the spelling of two phonetically similar names
  – America ≠ Amrica
    • This is a spelling mistake
  – disease ≠ diseased
    • This is a morphological difference
      – These two words are different tenses of the same base word
  – cow ≠ bovine
    • These two words are synonyms
Similar Sounding Names

• Smith ≠ Smythe
  – This has become known as the Soundex problem
• Similar names (e.g. smith, smythe and schmidt) are algorithmically converted into a common code so they may be found more easily

• History
  – 1918 Robert C. Russell patents a system for creating card indexes based on the sound of peoples names
  – This system was used (1930s) for indexing (on 3x5 cards) the 1880 US census
  – This is all before the invention of the computer!

• Useful today
  – Directory enquiries (411), Wisconsin Driver's License Numbers
  – Search engines, databases
Soundex Algorithm

- Convert name into uppercase and remove punctuation
- Take first letter unchanged
- Convert remaining letters to digits using the table
- Discard adjacent identical coded letters
- Truncate or pad to length 4 using 0s

<table>
<thead>
<tr>
<th>Letters</th>
<th>Digit</th>
</tr>
</thead>
<tbody>
<tr>
<td>AEIOUHWY</td>
<td>skip</td>
</tr>
<tr>
<td>BFPV</td>
<td>1</td>
</tr>
<tr>
<td>CGJKQSZX</td>
<td>2</td>
</tr>
<tr>
<td>DT</td>
<td>3</td>
</tr>
<tr>
<td>L</td>
<td>4</td>
</tr>
<tr>
<td>MN</td>
<td>5</td>
</tr>
<tr>
<td>R</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>S530</td>
</tr>
<tr>
<td>Smythe</td>
<td>S530</td>
</tr>
<tr>
<td>Schmidt</td>
<td>S253</td>
</tr>
<tr>
<td>de Ward</td>
<td>D630</td>
</tr>
</tbody>
</table>
Soundex Problems

• Poor Precision - many false positives
  – Knuth = K530
  – Kant = K530

• Poor Recall - many false negatives
  – Leigh = L200
  – Lee = L000

• Dependence of initial letter
  – Phifer = P160
  – Fifer = F160

• No rules for Prefixes
  – O’Keefe = O210
  – Keefe = K100
Soundex Problems

• Doesn’t handle “standard” abbreviations (familiars)
  – Bill = B400
  – William = W450

• Designed for English names
  – No accents or non-Romans (Æ, Þ, ð)
  – Appalling on words that are not names

• Several (incompatible) variants exist

• How many unique Soundex codes exist?
Soundex Alternatives

• Daitch-Mokotoff
  – Designed for Slavic and German spellings of Jewish names

• Fonem
  – Designed for French names

• Guth name matching algorithm
  – Ethnicity independent but is a name pair matcher
Metaphone

• Designed for names and words
• 4 character code max
• 16 consonant sounds (or phonems):
  
  B X(sh) S K J T F H L M N P R Ø(th) W Y

• Rules:
  – Double letters→single (except c)
  – Initial letter
    • kn, gn, pn, ac, wr→drop initial
    • x→s
    • wh→w
    • vowel→keep
    • Otherwise follow the rules
  – Drop remaining vowels
# Metaphone Rules

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>When</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>B</td>
<td>unless at the end of a word after &quot;m&quot; as in &quot;dumb&quot;</td>
</tr>
<tr>
<td>C</td>
<td>X</td>
<td>(sh) if -cia- or -ch-</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>if -ci-, -ce- or -cy-</td>
</tr>
<tr>
<td></td>
<td>K</td>
<td>otherwise, including -sch-</td>
</tr>
<tr>
<td>D</td>
<td>J</td>
<td>if in -dge-, -dgy- or -dgi-</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>otherwise</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>F</td>
<td>silent if in -gh- and not at end or before a vowel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>in -gn- or -gned- (also see dge etc. above)</td>
</tr>
<tr>
<td></td>
<td>J</td>
<td>if before i or e or y if not double gg</td>
</tr>
<tr>
<td></td>
<td>K</td>
<td>otherwise</td>
</tr>
<tr>
<td>H</td>
<td>J</td>
<td>silent if after vowel and no vowel follows</td>
</tr>
<tr>
<td></td>
<td>H</td>
<td>otherwise</td>
</tr>
<tr>
<td>J</td>
<td>J</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>K</td>
<td>silent if after &quot;c&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>otherwise</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>F</td>
<td>if before &quot;h&quot;</td>
</tr>
<tr>
<td></td>
<td>P</td>
<td>otherwise</td>
</tr>
<tr>
<td>Q</td>
<td>K</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>X</td>
<td>(sh) if before &quot;h&quot; or in -sio- or -sia-</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>otherwise</td>
</tr>
<tr>
<td>T</td>
<td>X</td>
<td>(sh) if -tia- or -tio-</td>
</tr>
<tr>
<td></td>
<td>Ø</td>
<td>(th) if before &quot;h&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>silent if in -tch-</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>otherwise</td>
</tr>
<tr>
<td>V</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>F</td>
<td>silent if not followed by a vowel</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>if followed by a vowel</td>
</tr>
<tr>
<td>X</td>
<td>KS</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>KS</td>
<td>silent if not followed by a vowel</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>if followed by a vowel</td>
</tr>
</tbody>
</table>
Metaphone Examples

- Shubert = XBRT
- School = SKL
- Color = KLR
- Colour = KLR
- Their = ØR
- There = ØR
- Smith = SMØ
- Smythe = SMØ
- Schmidt = SKMT

- Have gone from names spelled differently (Smith, Smythe) to words that sound the same (their and there), but Smith and Smythe are still equivalent!

- It’s often possible to pronounce a metaphone encoding and guess the original word!
Spelling Correction

• America ≠ Amrica
  – Spelling mistakes

• 80-95% of spelling errors are:
  – One letter wrong
    • America ≠ Amereca
  – Two adjacent letters transposed
    • Accuracies ≠ Accuraceis
  – One letter inserted
    • Coiled ≠ Coilled
  – One letter omitted
    • Correlation ≠ Corelation

• Additionally, the first letter is usually correct
Spelling Correction

• Early (1964) solution by Damerau
  – Order dictionary of words by length
  – One letter wrong
    • Words must be same length
    • Must differ in only one place

  – Two letters transposed
    • Words must be same length
    • Transpose where different

  – One letter inserted
    • Dictionary word must be 1 shorter

  – One letter omitted
    • Dictionary word must be 1 longer
Spelling Correction

• Reduce those terms that need to be matched:
  – There are 26 letters in the Roman alphabet
  – The letters of a word can be encoded in a 26 bit long bit-string
    • E.g. debate

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
<th>i</th>
<th>j</th>
<th>k</th>
<th>l</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

• Compute this for every word in the dictionary
• Compute this for term
• If the two strings differ by at most one bit then string compare
  • That is, the Hamming Distance is 1
Tries

• Letter wrong
  – alphibet

• Match against trie as far as possible
  – alph
  – i ≠ a
  – exchange i for a and keep matching
  – bet

• Letter deleted
  – Skip a node and continue

• Letter inserted
  – Skip a letter and continue

• Letters transposed
  – Switch letters and continue
Spelling Correction

• Unix spelling checker
  – Build a sparse hash table of correctly spelled words
    • 1 million bits for 10,000 words, is to use 1 bit in 100
    • Probability of a random sequence hashing to a correct word is 1/100

• Spelling correcting extension
  – Runtime
    
    ```
    hash the given word
    if collision
      correctly spelled
    else
      try all possible alternatives
    ```
  
  • For a word of length 8 letters
    – 8 deletions
    – 26 * 9 insertions
    – 25 * 8 changes
    – 7 exchanges
  
  • Requires 449 hashes of 8 characters to verify matches against the dictionary
Edit Distance

- Is it possible to measure how different two strings are?
- Edit distance
  - Minimum number of operations to convert one string into another
  - Operations allowed:
    - Insert character, delete character, replace character
      
      \[
      \begin{align*}
      d(\text{""}, \text{""}) &= 0 \quad // \text{both empty} \\
      d(s, \text{""}) &= \text{len}(s) \quad // \text{one is empty} \\
      d(\text{""}, s) &= \text{len}(s) \quad // \text{other is empty} \\
      d(s1 + c1, s2 + c2) &= \\
      \min(
      \begin{array}{l}
      c1 = c2 \
      ? d(s1, s2) \quad // \text{same char at end} \\
      : d(s1, s2) + 1, \quad // \text{replace c1 with c2} \\
      d(s1 + c1, s2) + 1, \quad // \text{c2 was inserted} \\
      d(s1, s2 + c2) + 1 \quad // \text{c1 was inserted}
      \end{array}
      )
      \end{align*}
      \]
    - Insertion and deletion are reciprocal
N-grams

• Category of algorithms including bi-grams and tri-grams
  – A bi-gram is two adjacent letters in a word
  – A tri-gram is three adjacent letters

• For each word
  – Compute the set of unique bi-grams
  – Compute the similarity using the Dice coefficient
    \[ d = \frac{2c}{a + b} \]
    – where
      \[ a = \text{unique in string A} \]
      \[ b = \text{unique in string B} \]
      \[ c = \text{shared in A and B} \]

If \( d = 1 \), words are the same
If \( d = 0 \), words share no n-grams
N-grams

• Statistics
  Bi-grams → statistics
  7 Unique → at statistics

• Statistical
  bi-grams → statistical
  8 unique → at statistical

• Common to statistics and statistical
  6 bi-grams → at is statistics

\[ d = \frac{2 \times 6}{7 + 8} = 0.8 \]

• Similarity = 0.8
• Also used in clustering to find similar word groups
Stemming

- disease ≠ diseased
  - Disease and diseased are different morphological forms

- Morphological forms are common
  - disease, diseased, diseasing, diseases
  - treatment, treats, treated, treating, treat

- Documents containing these variants (but not the original search term) are likely to be additional relevant documents that match the information need, even if they don’t match the query

- Stemming algorithms typically try to reduce related words to the same base (or stem)
Suffix Stripping

• The English language uses a small number of suffixes to alter the tense of a word
  

• Given a list of suffixes repeatedly strip the longest suffix from the end of a word

• Example suffix list
  
  [ase], [ed], [i], [ion], [ise], [ment], [ness], [s]
Suffix Stripping

• Example transformations
  treatment → treat[ment]
  detection → detect[ion]
  mistiness → mist[i][ness]

• However it does go wrong
  diseases → d[ise][ase][s]

• From:
  advertise → advert[ise]
  lactase → lact[ase]
  bananas → banana[s]
Suffix Replacement

• Simple suffix replacement
• Always perform the longest rule

• Example rules:
  ~ies → ~y (except ~eies and ~aies)
  ~es → ~e (except ~aes, ~ees or ~oes)
  ~s → ~ (except ~us or ~ss)

• Examples
  cries → cry [ rule 1]
  inductees → inductee [rule 2 fails (ees), so rule 3]
  horses → horse [rule 2]
The Porter Algorithm

• All words of the form: [C](VC)^m[V]
  – C: some number of consonants
  – V: some number of vowels
  – []: optional
  – ()^m: m combination of ()

• 5 step transformation rules including:
  (m>0) ? ~ational → ~ate // relational → relate
  (m>1) ? ~ance → ~ // allowance → allow

• The algorithm is not perfect despite widespread use
Krovetz Stemmer

• A Hybrid algorithmic / dictionary method
• Results in words not stems
• There is a reference implementation for Java / C/C++
  – https://sourceforge.net/p/lemur/wiki/KrovetzStemmer/

• The Krovetz Stemmer
  – Has an exception list
  – Has a hard-coded word replacement list
  – Follows a set of rules
  – Patches up the results to be a “proper” word
Successor Stemming

• Is it possible to determine algorithmically how and where to break into stems from just the words?
• For a given word, against a dictionary, determine how many successors there are to each possible stem of a word
• Successor number decreases as the string gets longer until a segment boundary
• These boundaries are possible stem location
Successor Stemming

• Example:

Test word: READABLE

Dictionary:
ABLE, APE, BEATABEL, FIXABLE, READ, READABLE READING, READS, RED, ROPE, RIPE

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Successors</th>
<th>Letters</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R$</td>
<td>3</td>
<td>EIO</td>
</tr>
<tr>
<td>RE</td>
<td>2</td>
<td>AD</td>
</tr>
<tr>
<td>REA</td>
<td>1</td>
<td>D</td>
</tr>
<tr>
<td>READ</td>
<td>3</td>
<td>AIS</td>
</tr>
<tr>
<td>READA</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>READAB</td>
<td>1</td>
<td>L</td>
</tr>
<tr>
<td>READABL</td>
<td>1</td>
<td>E</td>
</tr>
<tr>
<td>READABLE</td>
<td>1</td>
<td>\0</td>
</tr>
</tbody>
</table>
Successor Stemming

• Methods of choosing the stem
  – Cutoff
    • Break if the character has a successor greater than some constant
  – Peak and plateau
    • Break if a character has a successor greater than the character preceding
      and greater than the character following
  – Complete word
    • Break at a segment if the segment is a complete word also in the dictionary
  – Statistical methods

• But, it might break at a prefix
  – Antibody, antibiotics, antibacterial
  – If segment occurs $\leq 12$ times, use it
  – Else use second segment
Strong and Weak Stemmers

• Weak Stemmer
  – Merges only a small number of morphologically related terms together (e.g. s-striping)

• Strong Stemmer
  – Merges many more variants together

• In IR weak stemmers generally perform better than strong stemmers – why?
Thesaurus

• cow ≠ bovine
  – These words are synonyms

• Obvious solution:
  – Use Roget’s thesaurus or WordNet
  – Build a B-tree from the head words pointing to the synonyms
  – For each word in the query
    • Add synonyms from the thesaurus

• But
  – Roget’s and WorldNet are general purpose
  – Most queries are special purpose

• Can we manually construct a thesaurus?
  – Prohibitively expensive, but some do exist (UMLS, CAS, etc.)
  – Considerable time to build and verify
Automatic Thesaurus Construction

• There are many methods, here’s one:
  – For a collection of documents, \( c \), build a list of all unique terms \( U \)
  – For each unique term, \( u \in U \)
    • Construct subset, \( s \), all documents containing the term \( u \)
    • Construct a subset, \( r \), all documents not containing the term \( u \)
    • Construct \( T \), the list of unique terms in \( s \)
    • For each unique term \( t \in T \)
      – Compute \( df_{ts} \), the document frequency of term \( t \) in subset \( s \)
      – Compute \( dt_{tr} \), the document frequency of term \( t \) in the remaining documents \( r \)

• Now, if \( df_{ts} >> dt_{tr} \), then \( t \) co-occurs more frequently with \( u \) than without term \( u \). On other words, \( t \) and \( u \) correlate
  – There are other (better) methods of computing the correlation
Automatic Thesaurus Construction

• Problem:
  – Some terms only occur once!
  – Some terms occur in every document!

• Solution:
  – Determining which terms in $U$ are good (discriminating) terms

• Problem:
  – Multi-word synonyms: “world health organization” and “who”

• Solution:
  – Determining which phrases from $U$ should be used

• Does *correlation* equate to *synonymy*?
  – “Thames”, “tube” and “underground” are likely to correlate to “London”, but are they synonyms?
  – If recall and precision are increased, does this matter?
Content Bearing Terms

• A content bearing term is a term that has a meaning in the context of the document collection. Terms that are good discriminators in the collection. Terms that part of the collection is about.

• Content bearing terms:
  – Likely to have an ordered distribution in the collection
  – Likely to co-occur with other content bearing terms

• Non-content bearing terms:
  – Likely to be randomly distributed in the collection (noise)
  – Likely to co-occur with many other terms

• So, if a content bearing term correlates with other content bearing terms, compute the correlation to all other terms that co-occur with that term
Content Bearing Terms

• Very briefly:
  – For each term, \( t \), in the collection
    • Divide the collection into two parts those documents containing \( t \) and those not containing \( t \)
    • For each document containing \( t \)
      – Sum the correlation of each unique term to term \( t \)
    • Average these over the whole subset containing \( t \)

• For strong content bearing terms, the score will be high. Other terms will show a strong correlation to the term. Other terms will co-occur with that term. That term is a content bearing term
Content Bearing Phrases

• For a given document, if a phrase only occurs once, it is not likely to be content bearing

• Construct a candidate list of all possible phrases from a document thus:
  – Replace every stop word with a break
  – Replace all punctuation with a break
  – Compute the list of all phrases along with a count of the number of occurrences
  – If a phrase occurs more than once, it and all sub-phrases of it are candidates

• From the list of candidates, determine if or not it is content bearing

• Does this miss phrases like “cancer of the lung” as “of” and “the” are stop-words?
Index Time or Run Time

• Once the stems and thesaurus words are computed, should all terms encountered during indexing be converted into this reduced vocabulary?

• Advantages
  – Reduced vocabulary file
  – Increased postings density
    • so the index is smaller overall (due to compression)

• Disadvantages
  – Can’t search for the original word
  – Can’t change the algorithm after indexing
Summary

• Looked at term conflation
  – Soundex and metaphone
  – Spelling checkers
  – Stemming
  – Thesaurus

• String similarity measures
  – Edit distance
  – N-grams

• Content bearing terms

• For next lecture: