Lexicon & Text Normalization

Overview of these Slides

- These slides are focused on the issue of how a search engine creates its database of documents.
- In particular how the search engine determines what words go into its inverted index, the **lexicon**
- we begin with a discussion of word frequency and determining the number of unique words (the **vocabulary**)
- then we look at some specific operations
  - tokenization
  - stop words
  - capitalization
  - case folding
  - Synonyms (thesaurus)
  - Similar sounding words
  - Stemming

Power Laws and Zipf's Law

- **An equation of the form $y = kx^c$ is called a power law**
  - **k and c are constants**
- **Zipf's law is a power law with $c = -1$**
  - George Zipf noticed the rule while examining word frequencies in text
  - His rule states that the frequency of any word is inversely proportional to its rank in the frequency table
  - Thus the most frequent word will occur approximately twice as often as the second most frequent word, three times as often as the third most frequent word, etc
- **On a log-log plot, power laws give a straight line with slope c.**

$$\log(y) = \log(kx^c) = \log k + c\log(x)$$

Some Power Law Examples

- ***Population of U.S. states*:** if we order states in the US by population and let y be the population of the x-th most populous state, then x and y obey Zipf's law approximately.
- ***Book sales at Amazon.com***; let x represent the rank of books by sales and let y be the number of sales of the x^{th} best selling book over some period of time. According to Amazon the best seller sold 1,000,000 copies and the 10^{th} best sold 10,000 copies, and the 100^{th} best selling book sold 100 copies
- ***Node degrees in the web graph***: order all pages by the number of in-links to that page. Let x be the position of a page in this ordering and let y be the number of in-links to the x^{th} page. Then y is a power law function of x

Other Power Law Examples

- **Frequency of accesses to web pages**
  - example1: the access counts on Wikipedia pages, with *s* approximately equal to 0.3
- **Words in the English language**
  - for instance, in Shakespeare's play Hamlet with *s* approximately 0.5
- **Sizes of settlements**
- **Income distributions among individuals**
- **Size of earthquakes**
- **Notes in musical performances**

http://en.wikipedia.org/wiki/Zipf's_law
http://www.nslij-genetics.org/wli/zipf/
http://www.cut-the-knot.org/do_you_know/zipfLaw.shtml

Statistical Properties of Text

- **We can ask two questions about words in a set of documents**

1. How is the frequency of different words distributed?
2. How fast does vocabulary size grow with the size of a set of documents?

- **Such factors affect the performance of information retrieval and can be used to select appropriate term weights and other aspects of an IR system.**

Word Frequency

- **A few words are very common.**
  - Two very frequent words (e.g. "the", "of") can account for about 10% of word occurrences.
- **Most words are very rare.**
  - Half the words in a corpus appear only once, called *hapax legomena* (Greek for "read only once")
- **The above phenomenon is called a "heavy tailed" distribution, since most of the probability mass is in the "tail"**

Sample Word Frequency Data (from B. Croft, UMass)

Frequent Word	Number of Occurrences	Percentage of Total
the	7,398,934	5.9
of	3,893,790	3.1
to	3,364,653	2.7
and	3,320,687	2.6
in	2,311,785	1.8
is	1,559,147	1.2
for	1,313,561	1.0
The	1,144,860	0.9
that	1,066,503	0.8
said	1,027,713	0.8

Frequencies from 336,310 documents in the 1GB TREC Volume 3 Corpus
125,720,891 total word occurrences; 508,209 unique words

Zipf's Law Impact on IR

- **Good News: Stopwords will account for a large fraction of text so eliminating them greatly reduces inverted-index storage costs.**
- **Bad News: For most words, gathering sufficient data for meaningful statistical analysis (e.g. for correlation analysis for query expansion) is difficult since they are extremely rare.**

Vocabulary Growth

- **Our second question:** How does the size of the overall vocabulary (number of unique words) grow with the size of the corpus?
- **This determines how the size of the inverted index will scale with the size of the corpus.**
- **In practice a vocabulary is not really upper-bounded due to proper names, typos, etc.**

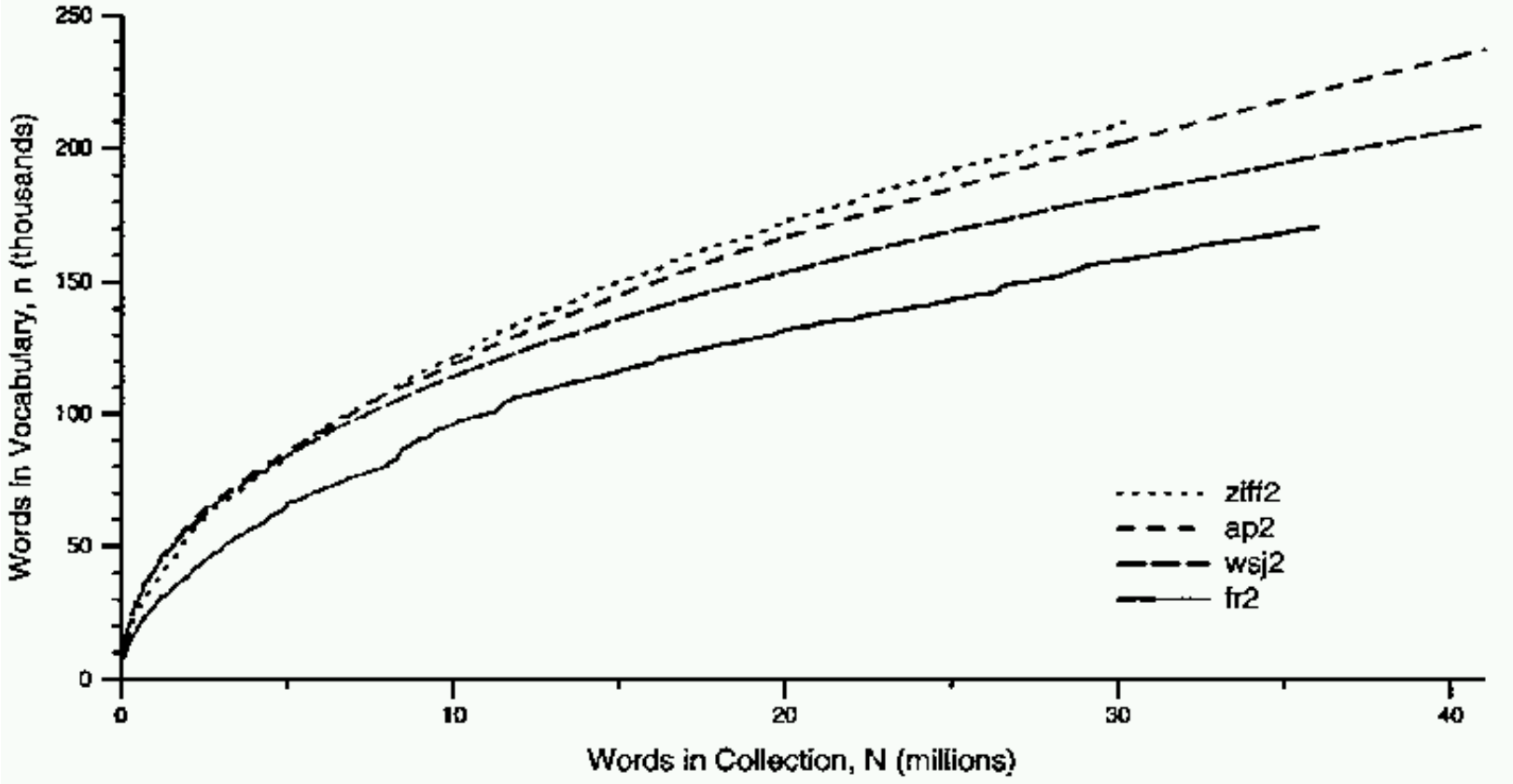
Heaps' Law

- **If *V* is the size of the vocabulary and *n* is the number of words:**

$$V = Kn^{\beta} \quad \text{with constants } K,\ 0 < \beta < 1$$

- **Typical constants:**
  - ***K*** **≈ 10 – 100**
  - **β ≈ 0.4 – 0.6 (approx. square-root)**
- So for $n = 100{,}000{,}000$ and $\beta = 0.5$, $K=1$ the vocabulary size is $\approx 10{,}000$
- Heap's law describes the number of distinct words (V) in a set of documents as a function of the document length (n)

Heaps' Law Data: An Example



Here 40,000,000 words implies a 250,000 vocabulary further implying k = 33 and b=0.5

Heap's Law Conclusions

1. the dictionary size continues to increase with more documents in the collection, rather than a maximum vocabulary size being reached
2. the size of the dictionary is quite large for large collections

- 1. and 2 imply that dictionary compression is important to do

Lexicon Construction

What is a Lexicon?

- **A database of the vocabulary of a particular domain (or a language)**
- **It is more than a list of words/phrases**
- **Usually it includes some linguistic information**
  - *Morphology* (manag- e/es/ing/ed → manage), (description of the structure of a language's units such as root words, parts of speech, intonations, stresses)
  - *Syntactic patterns* (such as Verb-Object or Subject-Verb-Object)
- **Often some semantic information is included:**
  - *Is-a hierarchy*, e.g.
    - lion is-a primate is-a mammal; corvette is-a Chevrolet is-a General Motors car
  - *Synonyms* (e.g. restaurant, café)
  - Numbers convert to normal form: Four → 4
  - Dates convert to normal form (month/day/year)
  - Alternative names converted to explicit form
    - Mr. Carr, Tyler, Presenter → Tyler Carr

One Challenge: Determining the Characters

- **Digital documents are typically bytes in a file. The first step of processing is to convert this byte sequence into a linear sequence of characters.**
  - Plain English text in ASCII encoding poses no problem
  - However, multi-byte encoding schemes, such as Unicode, UTF-8, or various national or vendor-specific standards are more difficult to identify and process.
  - Determining the correct encoding is a machine learning classification problem but is often handled by heuristic methods or by using provided document metadata such as:
  - `<html lang="en-US">, <html lang="fr">, <Q lang="he" dir="rtl">`
- **Alternately, the characters may have to be decoded out of some binary representation like Microsoft Word DOC files and/or a compressed format such as zip files**
- **Even for plain text documents, additional decoding may need to be done. In *XML* document's character entities, such as &, need to be decoded to give the correct character**
- **Finally, the textual part of the document may need to be extracted out of other material, e.g. when handling postscript (PS) or PDF files**
- **Solution:** *This problem is usually solved by licensing a software library that handles decoding document formats and character encodings.*

More Complications: Format/Language

- Documents being indexed can include items from **many different** languages
  - A single index may contain terms from many languages.
- **Sometimes a single document or its components can contain multiple language formats, e.g.**
  - French email with a German pdf attachment.
  - French email with quoted clauses from an English-language contract
  - French email with Arabic or Hebrew phrases (right-to-left)

Tokenization

- **Definition**: *tokenization*: The task of chopping a document unit into pieces, called tokens, and possibly throwing away certain characters
- A *token* is an instance of a sequence of characters in some particular document that are grouped together as a useful semantic unit
- A *term* is a (possibly normalized) type that is included in the lexicon
- Simple tokenization algorithm: "chop on whitespace and throw away punctuation characters", but consider these examples:
  - "Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing"
  - O'Neill: is the proper token, but according to our rule above it might be treated as one of these five possibilities: neill, oneill, o'neill, o' neill, o neill?
  - aren't: is the proper token, but according to our rule above it might be treated as one of these four possibilities: aren't, arent, are n't, aren t?
- So the simple strategy of splitting on all non-alphanumeric characters can and does fail and one needs a more powerful rule

Token Normalization

- *Token normalization* is the process of canonicalizing tokens so that matches occur despite superficial differences in the character sequences of the tokens.
- The **standard** way to normalize is to implicitly create *equivalence classes* , which are normally named after one member of the set; this is often called stemming;
  - *stemming* is the process for reducing inflected (or sometimes derived) words to their stem, base or root form – generally a written word form
  - A stemmer reduces the words "fishing", "fished", "fish", and "fisher" to the root word, "fish".
  - Later slides discuss stemming in more detail
- An **alternative** to creating equivalence classes is to maintain relations between un-normalized tokens. This method can be extended to hand-constructed lists of synonyms such as *car* and *automobile*.
- One way is to perform the expansion is during index construction. When the document contains automobile, we index it under car as well (and, usually, also vice-versa)

Handling Unusual Specific Tokens

- E.g. aircraft names like B-52, or a TV show name such as M*A*S*H
- A tokenizer should recognize as a single token: email addresses (jblack@mail.yahoo.com), web URLs (http://stuff.big.com/new/specials.html), numeric IP addresses (142.32.48.231), package tracking numbers (1Z9999W99845399981), and more.
- In English, *hyphenation* is used for various purposes ranging from splitting up vowels in words (*co-education*) to joining nouns as names (*Hewlett-Packard*) to a copyediting device to show word grouping (*the hold-him-back-and-drag-him-away maneuver*).
- Splitting on white space can create errors, e.g.
  - *San Francisco, Los Angeles*
  - foreign phrases (*au fait*) and compounds that are sometimes written as a single word and sometimes space separated (such as *white space* vs. *whitespace*).
  - Other cases with internal spaces that we might wish to regard as a single token include phone numbers ((800) 234-2333) and dates (Mar 11, 1983).
- See *Introduction to Information Retrieval* for more details, Section 2.2

Dropping Common Terms: Stop Words

- We have already seen that some extremely common words appear to be of little value in helping select documents matching a user need. These words are called *stop words* .
- The general strategy for determining a stop list:
  - is to sort the terms by *collection frequency* (the total number of times each term appears in the document collection), and then to take the most frequent terms as a *stop list* , the members of which are then discarded during indexing.
  - Using a stop list significantly reduces the number of postings that a system has to store;
- **Exception**: phrase searches are an important exception, e.g.
  - The phrase query "President of the United States", contains two stop words
  - The meaning of "flights ***to*** London" is likely to be lost if the word ***to*** is stopped out.
  - A search for Vannevar Bush's article "As we may think" will be difficult if the first three words are stopped out, and the system searches simply for documents containing the word think.
  - Some song titles and well known pieces of verse consist entirely of words that are commonly on stop lists (To be or not to be, Let It Be, I don't want to be, ...).

A stop list of 25 semantically non-selective words

a	an	and	are	as	at	be	by	for	from
Has	he	in	is	it	its	of	on	that	the
to	was	were	will	with					

- For efficiency, stopwords are stored in a hashtable to recognize them in constant time.
- There is some storage overhead for a hash table, but it should fit in main memory

- The MOST COMMONLY used list of stop words
- http://www.lextek.com/manuals/onix/stopwords1.html

Capitalization/Case-Folding

- *case-folding* is reducing all letters to lower case.
- Often this is a good idea:
  - *automobile* will match *Automobile*
  - *ferrari* will match *Ferrari*

- On the other hand, such case folding can equate words that might better be kept apart. Here are some Proper Nouns that when converted to lower case become something different:
  - *General Motors*,
  - *The Associated Press*
  - *the Fed* vs. *fed*
  - person names such as *Bush*, *Black*

Stemming and Lemmatization

- **Goal: to reduce multiple forms of a word to a common base form, e.g.**
  - am, are, is => be
  - car, cars, car's, car' => car
- **So for example**
  - the boy's cars are different colors =>
  - the boy car be differ color
- *Stemming* usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving a common base correctly most of the time
- *Lemmatization* usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the *lemma*

JavaScript Porter Stemmer Online

- **A stemmer written by Martin Porter has become the de-facto standard algorithm used for English stemming**

- The website to the left implements Porter's stemmer algorithm in JavaScript

- Example: "to be or not to be" is not changed at all

Example: Since the department's founding in 1968, our faculty have made pioneering contributions to fundamental and interdisciplinary fields of computing."

Since -> Sinc
department -> depart
founding -> found
pioneering -> pioneer
contributions -> contribut
fundamental -> fundament

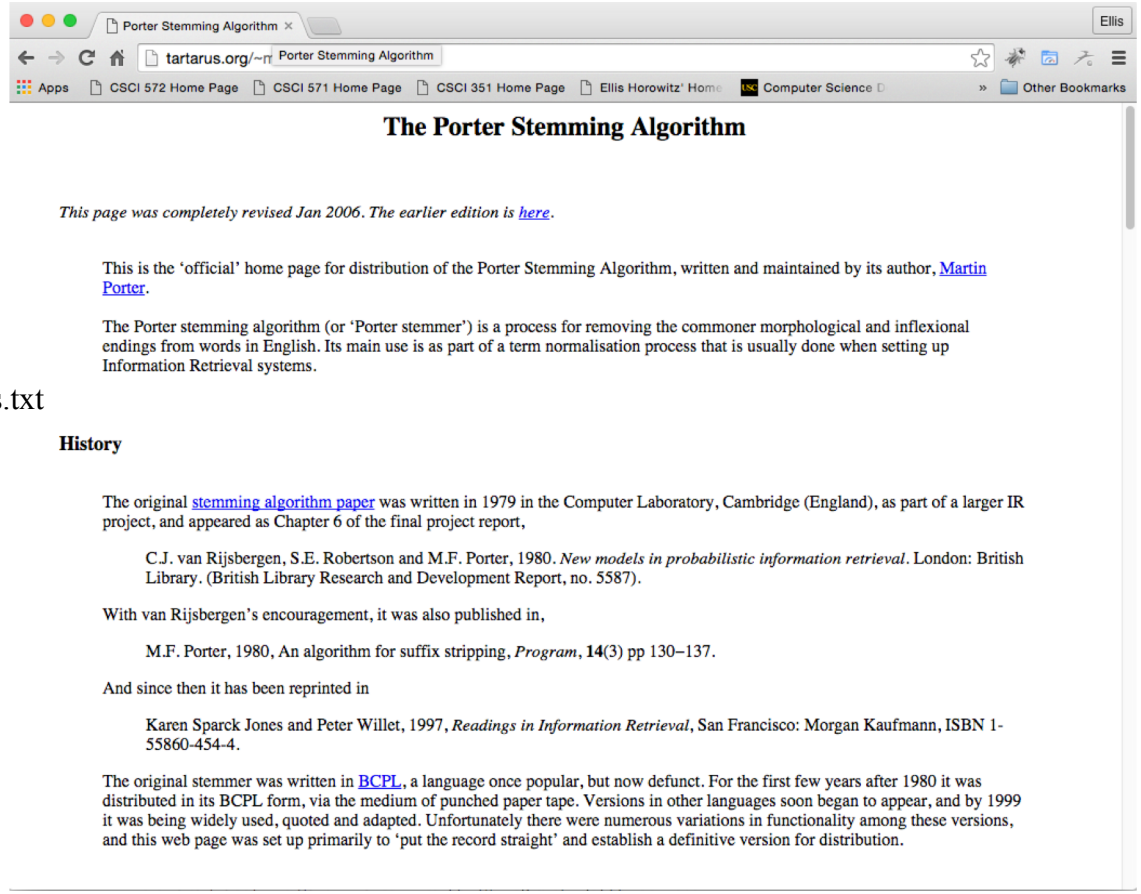
The Porter Stemming Algorithm

- The official site is located at http://tartarus.org/~martin/PorterStemmer/
- The site offers versions of the algorithm in multiple languages including C, Perl, Java, JavaScript, PHP

Original paper written in 1979;

JavaScript version can be found at http://tartarus.org/~martin/PorterStemmer/js.txt

The JavaScript is very easy to follow



Typical Rules in Porter's algorithm

- **Porter's algorithm has five phases of word reductions, applied sequentially**
- **Each phase has a set of conventions for how to select the transformation**
- **For a given set of rules only the rule with longest suffix match is applied; for example**
  - *sses* → *ss*
  - *ies* → *i*
  - *ational* → *ate*
  - *tional* → *tion*

  For Ex. CARESSES maps to CARESS since SSES is the longest match.
- **Other possibilities are suffix stripping algorithms**
  - *If the word ends in 'ed' remove the 'ed'*
  - *If the word ends in 'ing', remove the 'ing'*
  - *If the word ends in 'ly', remove the 'ly'*

Error Metrics

- **Two error mesasurements**
  - **Over-stemming:** two separate inflected words are stemmed to the same root, but should not have been, a *false positive*
  - **Under-stemming:** where two separate inflected words should be stemmed to the same root, but are not, a *false negative*
- **In the Porter stemmer**
  - Universal, university and universe are stemmed to univers, a case of over-stemming
  - alumnus goes to alumnu, alumni goes to alumni, alumna goes to alumna, but all should stem to the same root, under-stemming

One Last Algorithm Soundex Algorithm

- **Soundex is a phonetic algorithm for indexing names by their sound when pronounced in English.**
- **The basic aim is for names with the same pronunciation to be encoded to the same string so that matching can occur despite minor differences in spelling, e.g.**
  - **SMITH and SMYTH** (the second option is pronounced like the first)
- **Soundex is the most widely known of all phonetic algorithms, as it is a standard feature of MySQL, Microsoft SQL Server and Oracle**
- **E.g., *chebyshev* → *tchebycheff***
- **It was developed by Robert Russell and Margaret Oldell in 1918**

Soundex – Typical Algorithm

- Turn every token to be indexed into a 4-character reduced form
- Do the same with query terms
- Build and search an index on the reduced forms
  - (when the query calls for a soundex match)

See Understanding Classic SoundEx Algorithms
http://www.creativyst.com/Doc/Articles/SoundEx1/SoundEx1.htm#Top

Soundex – Algorithm

- The algorithm mainly encodes consonants; a vowel will not be encoded unless it is the first letter.
- Every Soundex code consists of a letter and three numbers, e.g. W-252;
  - the letter is the first letter of the surname
  - The numbers are assigned to the remaining letters of the surname according to the Soundex guide
- Washington is coded W-252 (W, 2 for the S, 5 for the N, 2 for the G, and remaining letters disregarded
- Lee is coded L-000 (L the leading letter and since the ee are dropped the zeros are used as padding)

A web page containing versions of Soundex in 36 different programming languages is here
http://rosettacode.org/wiki/Soundex
http://www.archives.gov/research/census/soundex.html

Soundex – Typical Algorithm

1. Retain the first letter of the word.
2. Change all occurrences of the following letters to '0' (zero):
   'A', E', 'I', 'O', 'U', 'H', 'W', 'Y'.
3. Change letters to digits as follows:

- B, F, P, V → 1
- C, G, J, K, Q, S, X, Z → 2
- D,T → 3
- L → 4
- M, N → 5
- R → 6

Soundex Continued

4. Remove all pairs of consecutive digits.
5. Remove all zeros from the resulting string.
6. Pad the resulting string with trailing zeros and return the first four positions, which will be of the form <uppercase letter> <digit> <digit> <digit>.

E.g., *Herman* becomes H655

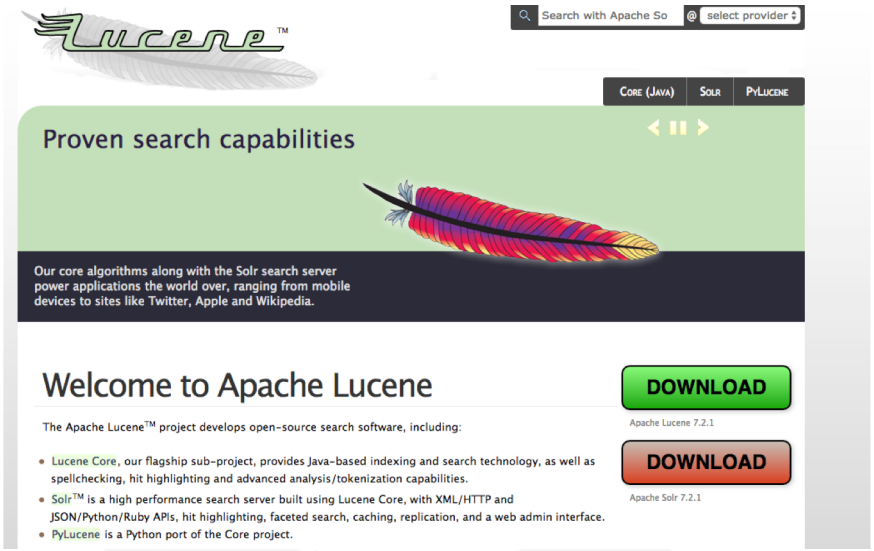
- The algorithm is straight forward to code and requires no backtracking or multiple passes over the input word

Soundex Example

- **Example 1: Zbygniewski has code Z125**
  - starts with Z, consonant in group 1 (the b), one in group 2 (the g), one in group 5 (the n), remainder is ignored
  - if Zbignyefsky is entered, the same code is produced
- **For some words using Soundex does not provide value, e.g.**
- **Example 2: disapont has code D215**
  - a spelling corrector in its database will retrieve all words with code D215 which includes the following 31 words: disband, disbands, disbanded, disbanding, disbandment, disbandments, dispense, dispenses, dispensed, dispensing, dispenser, dispensers, dispensary, dispensaries, dispensable, dispensation, dispensations, deceiving, deceivingly, despondent, despondently, disobeying, disappoint, disappoints, disappointed, disappointing, disappointedly, disappointingly, disappointment, disappointments, disavowing

Postscript: Lucene and Solr

- **Lucene**
  - a high-performance, full-featured text search engine library written entirely in Java.
  - suitable for any application that requires full-text search
  - an open source project available for free download
- **Solr**
  - HTTP-based Search Server
  - Uses XML for configuration
  - Many, many nice features that Lucene users need
    - Faceting, spell checking, highlighting
    - Caching, Replication, Distributed
- Download both from http://lucene.apache.org



Main Lucene Modules

- **Lucene is the underlying software that imports documents and creates an inverted index**
  - **Document Parsing and Analysis**
    - Tokenization
    - Where tokens are indexed
  - **Document Identification**
    - Where the Document ID is created
    - Date of Document is extracted
    - Title of document is extracted
  - **Indexing**
    - Provides access to indexes
    - Maintains indexes

Lucene Tokenizers Specify How the Text in a Field is to be Indexed

- Tokenizers in Lucene
  - WhitespaceTokenizer
    - divides text at whitespace
  - SimpleTokenizer
    - divides text at non-letters
    - convert to lower case
  - StopTokenizer
    - SimpleAnalyzer
    - removes stop words
  - StandardTokenizer
    - good for most European Languages
    - removes stop words
    - convert to lower case
    - https://lucene.apache.org/core/3_0_3/api/core/org/apache/lucene/analysis/standard/StandardTokenizer.html
- Create you own Tokenizers

TokenFilters Refine the Results of Tokenizers

- LowerCaseFilter
- StopFilter
- ISOLatin1AccentFilter
- SnowballFilter
  - stemming: reducing words to root form
  - rides, ride, riding => ride
  - country, countries => countri
- contrib/analyzers for other languages
- SynonymFilter (from Solr)
- WordDelimiterFilter (from Solr)