List Comprehensions and Dictionaries

Methods in Computational Linguistics I
October 8, 2010

Last Time

- File IO with Matt.
- Any Questions?

This time

- List Comprehensions
- Generating Expressions
- Dictionaries

How to split a string into a list of words

- sent = “That isn’t the problem, Bob.”
- sent.split()
- vs.
- nltk.word_tokenize(sent)

List Comprehensions

- Compact way to process every item in a list.

- [x for x in array]

Methods

- Using the iterating variable, x, methods can be applied.
- Their value is stored in the resulting list.
- [len(x) for x in array]

Conditionals

- Elements from the original list can be omitted from the resulting list, using conditional statements
- [x for x in array if len(x) == 3]

Building up

- These can be combined to build up complicated lists
- [x.upper() for x in array if len(x) > 3 and x.startswith('t')]

Lists containing lists

- Lists can contain lists
- [[a, 1], [b, 2], [d, 4]]
- ...or tuples
- [(a, 1), (b, 2), (d, 4)]
- [[d, d*d] for d in array if d < 4]

Using multiple lists

- Multiple lists can be processed simultaneously
- [x*y for x in array1 for y in array2]

Dictionaries

- Dictionaries map one object to another.
- For instance:
  - Student to grade
  - Word to pronunciation
  - Filename to corpus

Dictionaries

- Dictionaries allow you to “look up” one object by referencing the other.
- These two objects are called the ‘key’ and the ‘value’.
- Dictionaries are indexed by their ‘key’ and the associated ‘value’ is retrieved.
- (In other languages, dictionaries are sometimes called hash tables, maps, or associative arrays.)

Dictionaries in python

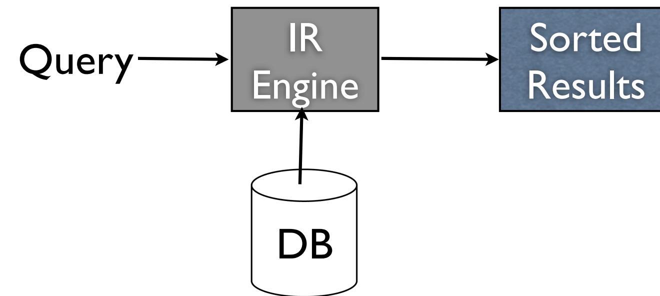
- Demo time.
- Chapter 5 in the NLTK book (pp. 189-198)

Information Retrieval

- Searching for documents, or information within documents.
- Search Engines -- Google, Yahoo, Bing, etc.
- Library search products
- Travel sites -- Orbitz, Expedia, Kayak, etc.
- Job search -- Monster, Careers.com

IR framework

- The user requests information via a **query**
- The system delivers a set of documents, typically ordered by **relevance**.



Inverted Index

- How can we determine which documents contain the words that we are interested in?
- A - "Cars need gasoline."
- B - "Gasoline prices rising."
- C - "Click for prices."

- cars - {A}
- need - {A}
- gasoline - {A,B}
- prices - {B,C}
- rising - {B}
- click - {C}
- for - {C}

Near misses

- “cars” doesn’t match “car” in an inverted index.
- “running” doesn’t match “run”.
- “running” doesn’t match “marathon”.

Morphological Analysis

- Morphological analysis that converts “cars” to “car +plural”, and “running” to “run +gerund”
- Only store the stem (or lemma) of every word in the index.
- At query time, stem the query.
- CON: This eliminates valuable information from the query.

Query Expansion

- Augment the query with related words, including stems, synonyms, etc.
- similar("running") = {"run", "runs", "ran", "marathon", "race", ...}
- Identifying similar words is an open research question.

Calculating Relevance

- How is relevance calculated?
- A - “Cars need gasoline.”
- B - “Gasoline prices rising.”
- C - “Click for prices.”
- *query - “price of gasoline”*

- Count the number of hits.
- Count the number of close hits.
- Scale the value of matching a word based on the rarity of the word. Matching “the” is less important than matching “centennial”.

Next Time

- Assignment 3 is assigned.
- Object-oriented design in Python