

AI Programming
CS662-2008F-11
Information Retrieval

David Galles

Department of Computer Science
University of San Francisco

11-0: Processing text

- Now that we know a little bit about how to consider and compare different states, we'll think about a problem with a harder representation.
- English-language text
 - Specifically, webpages
- We'll look at several different approaches to automatically summarizing and understanding documents.

11-1: Difficulties

- What makes working with natural language hard?

11-2: Difficulties

- What makes working with natural language hard?
- Nonunique parses
- Synonyms and multiple meanings
- Slang and technical terms
- Analogy and metaphor
- Misspelling and incorrect grammar

11-3: Information Retrieval

- Information retrieval deals with the storage, retrieval, organization of, and access to information items
- Overlaps with:
 - Databases (more of a focus on content)
 - AI
 - Search engines

11-4: Needs and queries

- A user typically has an *information need*.
- The job of an IR system is to translate that need into a query language and then find documents that satisfy that need.
- What are some sorts of query languages?

11-5: Query Languages

- What are some sorts of query languages?
- Keyword - Google, Yahoo!, etc.
- Natural language - Ask.com
- SQL-style
- Similar item - Netflix, Amazon
- Multimedia - Flickr

11-6: User tasks

- We'll also distinguish between different types of user tasks.
- The most common are *searching* and *browsing*.
 - Searching - the user has a specific information need, and wants a document that meets that need.
 - “Find me an explanation of the re module in Python”
 - Browsing - the user has a broadly defined set of interests, and wants information that satisfies his/her interests.
 - “Find me interesting pages about Python”
- Different modes have different models of success.

11-7: User tasks

- Searching and browsing are both *pull* tasks.
 - User is actively fetching information from a repository.
- We can also think about *push* tasks, where selected data is delivered to a client as it is made available.
- This is called *filtering*
 - RSS readers are an example of this, as is Google News.

11-8: Modeling a Document

- In order to match a query to a document, an IR system must have a *model* of the document.
- This might be:
 - A category or description (as in a library)
 - A set of extracted phrases or keywords
 - The full text of the document
 - Full text with filtering

11-9: “Bag of words” model

- The techniques we’ll look at today treat a document as a *bag of words*.
- Order is discarded; we just count how often each word appears.
- No semantics involved
- Intuition: Frequently-appearing words give an indication of subject matter.
- Advantage: No need to parse, computationally tractable for large collections.
- Disadvantage: Contextual information and meaning is lost.

11-10: Data cleaning

- When preparing a document such as a webpage for an IR system, the data must typically be *cleaned* first.
 - HTML, Javascript removed.
 - (Links and structural information might be kept separately)
 - Non-words removed.
 - Converted to lower case
 - *stopwords* removed. These are words that have little or no semantic content. (a, an, the, he, she, their, among, etc)

11-11: Data Cleaning

- *Stemming* might also be performed.
- Word suffixes, such as pluralization, past tense, -ing are removed.
- run, runs, running, runner all become run.
- Advantages: If we're just counting words, this lets us correctly count different forms of a word.
- Disadvantages: dealing with abnormal forms (person/people, run/ran), potential misgrouping (university, universal)
- The stemmer can be tuned to minimize either *false positives* (accidentally stemming a word it shouldn't) or *false negatives* (not stemming a word it should.)

11-12: “Bag of Words”

- Once a document has been cleaned, the simplest model just counts how many times each word occurs in the document.
- This is typically represented as a dictionary.
- You built this in assignment 1.

11-13: Evaluating an IR System

- Our prototypical use case is this:
 - The user submits a query to the system
 - Some documents are returned
- How can we evaluate performance?

11-14: Precision and Recall

- Precision measures how well returned documents match a query.
 - $\text{precision} = \frac{\text{matchingDocs}}{\text{totalDocsReturned}}$
- Recall measures the fraction of relevant documents returned.
 - $\text{recall} = \frac{\text{relevantDocsReturned}}{\text{totalRelevantDocs}}$
- When might we want high precision? High recall?
- Often, we can trade precision against recall.

11-15: Boolean Queries

- Boolean queries are simple, but not very practical.
- User provides a set of keywords.
 - Possibly also OR terms
- All documents containing all keywords are returned.
- This is the sort of query model that databases use

11-16: Boolean Queries

- Weaknesses:
 - No concept of partial match, or ability to rank results
 - Low recall
 - Boolean queries are awkward for users

11-17: Probabilistic Queries

- A simple extension is to allow partial matches on queries
- Score documents according to the fraction of query terms matched
- Return documents according to score
 - Example: Document contains “cat cat dog bunny fish”
 - Query is “cat dog (bunny OR snake) bird”
 - Score is $3/4$.

11-18: Probabilistic Queries

- Weaknesses:
 - Still requires logical queries
 - Doesn't deal with word frequency
 - Dependent on query length - short queries will have a hard time getting differentiated scores.
 - The average Google query is only three words long!

11-19: Dealing with Word Frequency

- Intuitively, some words in a document should matter more than others.
 - The word “aardvark” occurring 10 times in a document is probably more meaningful than the word “date” occurring 10 times.
- We want to weight words such that words which are rare in general, but common in a document, are more highly considered.

11-20: Building a corpus

- To measure how frequently words occur in general, we must construct a corpus.
 - This is a large collection of documents
- Must be careful to ensure that we select documents of the appropriate style
- Different types of documents have different word frequencies
 - New York Times vs Livejournal
- The statistical distribution of words in a corpus is called a *language model*.

11-21: Building a corpus

- We begin by cleaning the data as before
- Construct a dictionary that maps words to the number of pages they occur in.
 - Don't worry about multiple occurrences within a document
- The result is referred to as *document frequency*

11-22: TFIDF

- We can now weight each word to indicate its importance in the language model.
- The most common weighting scheme is TF-IDF: term frequency - inverse document frequency.
- $TFIDF(word) = TF(word) * \log\left(\frac{|corpus|}{DF(word)}\right)$
- $TF(word)$ is how frequently the word occurs in the search query (or a specific document)
- $DF(word)$ is the number of pages in the corpus that contain the word.

11-23: TFIDF

- Think about extrema:
 - What happens if a word occurs in exactly one document in the corpus?
 - What happens if a word occurs in every document in the corpus?
- We want to favor words that discriminate interesting pages from non-interesting pages

11-24: Word Weighting

- We can now process each document and assign a weight to each word.
- We could use this to improve the performance of the probabilistic scorer.
- More interestingly, we can use it to determine how similar two documents are.
- This gives us another way for users to search
 - “Find more documents like this”

11-25: Documents as vectors

- At this point, each document can be represented as a dictionary of words and TFIDF scores

{cat: 4.33; dog: 2.1 ; bunny: 8.2; fish: 0.3}

- Conceptually, these documents can be thought of as an n -dimensional vector, where n is the number of words in the lexicon (all words in all documents) and the value of $v[n]$ is the TFIDF score for that word.
- Many elements of the vector are zero, since those words don't appear in that specific document.

11-26: Comparing vectors

- We can now use well-known techniques from geometry to compare these vectors.
- We could measure the angle between the vectors.
 - The scale is not convenient, and the calculation is complicated.
- Easier is to measure the cosine of this angle.
- Identical documents have a cosine of 1, and completely dissimilar documents have a cosine of zero.

11-27: Computing cosine similarity

- The formula for the cosine of the angle between two vectors is: $\frac{a \cdot b}{\|a\| \|b\|}$
- This is the dot product of the two vectors, divided by the product of their magnitudes.
- The dot product is computed by summing the product of the respective elements of each vector:
- $\sum_i v1[i] * v2[i]$
- The magnitudes are computed by calculating the square root of the sum of the squares of each component. (this is Pythagoras' rule)
- $\sqrt{\sum_i v[i]^2}$

11-28: Computing cosine similarity

- The entire formula, in terms of words in documents, looks like this:

$$\cos(d_1, d_2) = \frac{\sum_{word \in d_1 \cap d_2} d_1[word] * d_2[word]}{\sqrt{\sum_{word \in d_1} d_1[word]^2} * \sqrt{\sum_{word \in d_2} d_2[word]^2}}$$

- This is a very powerful and useful technique for comparing documents.
- It can also be used to compare a query to a document.
- We'll return to it when we study clustering.

11-29: Putting it together

- To use a vector model:
 - Collect and clean a corpus and compute document frequencies.
 - For each document in the collection, clean and compute document frequencies.
 - For a query or sample document, compute TFIDF scores.
 - Compute cosine similarity for each document in the collection and return results from highest to lowest.

11-30: Summary

- Searching vs browsing
- “bag of words” model
- Precision and Recall
- Boolean and Probabilistic Queries
- Term Weighting
- Vector Models and cosine similarity