Introduction to Natural Language Processing

Document Classification

Problem Motivation

- You’re Google News, and you want to group news articles by topic.
- You’re a legal tech firm, and you need to sift through 100,000 pages of legal documents to find the relevant ones.
- You’re building a search engine for your Harry Potter fan fiction repository

Overview of Approach

- Compile documents
- Featurize them
- Compare their features

Simple Sample Problem

- 2 documents: “blue house” and “red house”
- Could featurize based on word counts
  - “blue house” → (red, blue, house) = (0, 1, 1)
  - “red house” → (red, blue, house) = (1, 0, 1)

Bag of Words

- A document represented as a vector of word counts is called a “bag of words”

- “blue house” → (red, blue, house) = (0, 1, 1)
- “red house” → (red, blue, house) = (1, 0, 1)
- “red red house” → (red, blue, house) = (2, 0, 1)

Terminology

Corpus: a dataset of text. e.g. Newspaper Articles, tweets

Document: A single entry from our corpus. e.g. Article, Tweet, Sentence, etc.

Vocabulary: All the words that appear in our corpus.

Bag of Words: A vector representation of a document based on word counts.

Stop Words: Words we ignore in our analysis that are too common to be useful.

Token: A single word.

Comparing the Features: Cosine Similarity

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{||A|| ||B||}$$

“red house” vs “blue house” →
(1,0,1) vs (0, 1, 1) →
similarity = .5

Is there a better way to featurize?

- Bag of words is naive--just word counts!
  - Longer docs => higher counts.
- Every word has equal weighting
  - “the” and “data” have different predictive power

- Can adjust word counts based on their frequency in the corpus...

Term Frequency

- Raw counts emphasize results from longer documents.
- Can normalize counts within a document to transform from term counts to term frequency.
    - Typically use a Euclidean/L2-norm.

TF-IDF

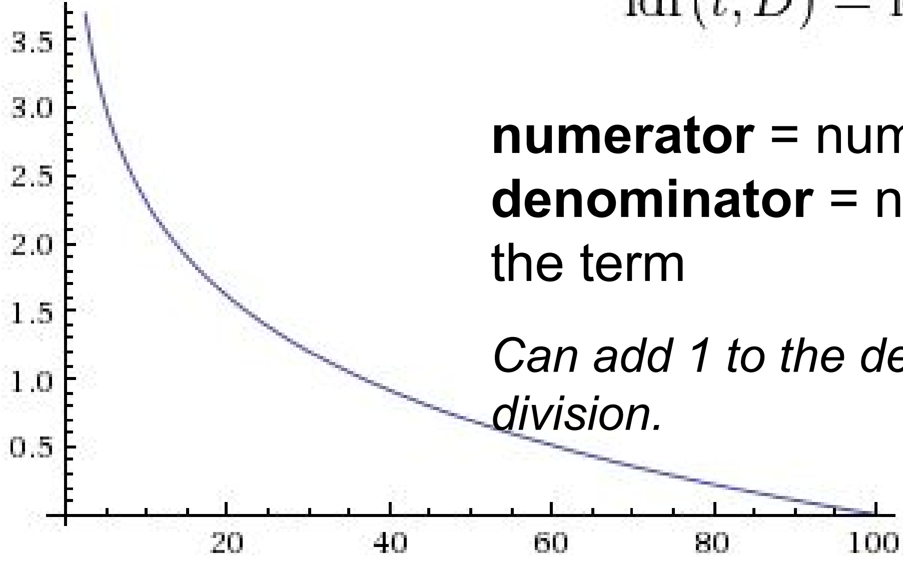
$$TF * (?)$$

- How to adjust the frequency?
- Words found in only one document should have highest weighting.
- Words found in every document should have lowest weighting.

IDF - Inverse Document Frequency

$$TF * IDF$$

$$\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$



numerator = number of documents in corpus
denominator = number of documents in corpus containing the term

Can add 1 to the denominator (and numerator) to avoid zero division.

Feature Engineering for Text

- Tokenize
- Remove stop words
- Stemming
  - e.g. “walked” → “walk”
- Lemmatization (often better than stemming)
  - e.g. “people” → “person”
- N-grams / skip grams

N-Grams

- Word order matters.
  - Lose this info with a bag of words.
- N-Grams attempt to retain this:

“Hello, my name is Brad.”:

2-Grams:

(Hello my), (my name), (name is), (is Brad)

3-Grams:

(Hello my name), (my name is), (name is Brad)

Be Aware: Word2Vec

Google project using neural nets to encode semantic meaning of words as a vector.

The vector for “San Francisco” has highest cosine similarity with: “Los Angeles”, “Golden Gate”, “Oakland”, “California”, “San Diego”, ...

Be Aware: Feature Hashing

Recall hash tables (i.e. dicts) use a function to assign keys to a “location” for fast lookup.

Can use hashing to take care of vectorizing data.

+ Improves Speed/Memory Usage.

- Loss of Interpretability

More Advanced NLP Problem Types

- Sentiment analysis
  http://nlp.stanford.edu/sentiment/index.html
- Machine translation
  https://medium.com/s-c-a-l-e/how-baidu-mastered-mandarin-with-deep-learning-and-lots-of-data-1d94032564a5
- Probabilistic parsing