Outline

- Natural Language Processing
  - Preprocessing
  - Statistics
  - Language models

Preprocess

- Tokenization or text normalization: turn data into sequence(s) of tokens

1. Remove unwanted stuff: HTML tags, encoding tags
2. Determine word boundaries: usually white space and punctuations
   - Sometimes can be tricky, like Ph.D.
3. Remove stopwords: the, of, a, with, …
4. Case folding: lower-case all characters.
   - Sometimes can be tricky, like US and us
5. Stemming/Lemmatization (optional): looks, looked, looking → look

Vocabulary

Given the preprocessed text

- Word token: occurrences of a word
- Word type: unique word as a dictionary entry (i.e., unique tokens)

- Vocabulary: the set of word types
  - Often 10k to 1 million on different corpora
  - Often remove too rare words

Zipf's Law

- Word count f, word rank r
- Zipf's law: $f * r \approx \text{constant}$

Word	Count f	rank r	fr
the	3332	1	3332
and	2972	2	5944
a	1775	3	5235
he	877	10	8770
but	410	20	8400
be	294	30	8820
there	222	40	8880
one	172	50	8600
two	104	100	10400
turned	51	200	10200
comes	16	500	8000
family	8	1000	8000
brushed	4	2000	8000
Could	2	4000	8000
Applausive	1	8000	8000

Zipf's law on the corpus *Tom Sawyer*

Bag-of-Words

How to represent a piece of text (sentence/document) as numbers?

- Let m denote the size of the vocabulary
- Given a document d, let $c(w, d)$ denote the #occurrence of w in d
- Bag-of-Words representation of the document

$$v_d = [c(w_1, d), c(w_2, d), \ldots, c(w_m, d)]/Z_d$$

- Often $Z_d = \sum_w c(w, d)$

tf-idf

- tf: normalized term frequency

$$tf_w = \frac{c(w,d)}{\max_v c(v,d)}$$

- idf: inverse document frequency

$$idf_w = \log \frac{\text{total \#doucments}}{\text{\#documents containing } w}$$

- tf-idf: $tf\text{-}idf_w = tf_w * idf_w$
- Representation of the document

$$v_d = [tf{-}idf_{w_1}, tf{-}idf_{w_2}, \dots, tf{-}idf_{w_m}]$$

Cosine Similarity

How to measure similarities between pieces of text?

- Given the document vectors, can use any similarity notion on vectors
- Commonly used in NLP: cosine of the angle between the two vectors

$$sim(x, y) = \frac{x^\top y}{\sqrt{x^\top x}\sqrt{y^\top y}}$$

Statistical language model

- Language model: probability distribution over sequences of tokens
- Typically, tokens are words, and distribution is discrete

- Tokens can also be characters or even bytes

- Sentence: "*the quick brown fox jumps over the lazy dog*"

Tokens: x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9

- Probabilistic model:

$$\mathrm{P}\left[x_1, x_2, x_3, \ldots, x_{\tau-1}, x_\tau\right]$$

Unigram model

- Unigram model: define the probability of the sequence as the product of the probabilities of the tokens in the sequence

Independence!

$$\mathrm{P}[x_1, x_2, \ldots, x_\tau] = \prod_{t=1}^{\tau} \mathrm{P}[x_t]$$

- Sentence: "*the dog ran away*"

$$\hat{\mathrm{P}}[the\ dog\ ran\ away] = \hat{\mathrm{P}}[the]\,\hat{\mathrm{P}}[dog]\,\hat{\mathrm{P}}[ran]\,\hat{\mathrm{P}}[away]$$

- How to estimate $\hat{\mathrm{P}}[the]$ on the training corpus?

Word	Count f
the	3332
and	2972
a	1775
he	877
but	410
be	294
there	222
one	172

n-gram model

- n-gram: sequence of n tokens

- n-gram model: define the conditional probability of the n-th token given the preceding $n-1$ tokens

$$\mathrm{P}[x_1, x_2, \dots, x_\tau] = \mathrm{P}[x_1, \dots, x_{n-1}] \prod_{t=n}^{\tau} \mathrm{P}[x_t | x_{t-n+1}, \dots, x_{t-1}]$$

Markovian assumptions

- $n = 1$: unigram
- $n = 2$: bigram
- $n = 3$: trigram

Training n-gram model

- Straightforward counting: counting the co-occurrence of the grams

For all grams $(x_{t-n+1}, \dots, x_{t-1}, x_t)$

1. count and estimate $\widehat{\mathrm{P}}[x_{t-n+1}, \dots, x_{t-1}, x_t]$
2. count and estimate $\widehat{\mathrm{P}}[x_{t-n+1}, \dots, x_{t-1}]$
3. compute

$$\widehat{\mathrm{P}}[x_t | x_{t-n+1}, \dots, x_{t-1}] = \frac{\widehat{\mathrm{P}}[x_{t-n+1}, \dots, x_{t-1}, x_t]}{\widehat{\mathrm{P}}[x_{t-n+1}, \dots, x_{t-1}]}$$

- Sentence: "*the dog ran away*" by trigram (n=3)

$$\widehat{\mathrm{P}}[the\ dog\ ran\ away] = \widehat{\mathrm{P}}[the\ dog\ ran]\ \widehat{\mathrm{P}}[away | dog\ ran]$$

$$\widehat{\mathrm{P}}[the\ dog\ ran\ away] = \widehat{\mathrm{P}}[the\ dog\ ran] \frac{\widehat{\mathrm{P}}[dog\ ran\ away]}{\widehat{\mathrm{P}}[dog\ ran]}$$

Rectify: smoothing

- Sparsity issue: $\widehat{\mathrm{P}}[\ldots]$ most likely to be 0

- Basic method: adding non-zero probability mass to zero entries

- Example: Laplace smoothing that adds one count to all n-grams

pseudocount$[dog\ ran\ away] = \text{actualcount}\ [dog\ ran\ away] + 1$ ⇦ trigram

pseudocount$[dog\ ran] \approx \text{actualcount}[dog\ ran] + |V|$

$$\hat{\mathrm{P}}[away|dog\ ran] = \frac{\text{pseudocount}[dog\ ran\ away]}{\text{pseudocount}\ [dog\ ran]}$$

Since number of bigrams "dog ran" ≈ Number of trigrams containing "dog ran"