Natural Language Processing

(NLP)

Objectives for the morning

- Explain the steps of turning raw text data into useful features
- Explain the difference between stemming and lemmatization
- Compute the TF-IDF of documents

Motivation

- You run Google News and you want to group news articles by topic

- You run a legal tech firm and you need to sift through 1,000,000 pages of legal documents to find the relevant ones

Text Featurization Pipeline

- Tokenization
- Lower case conversion
- Punctuation removal
- Stop words removal
- Stemming / Lemmatization
- Bag of words / N-grams

Terminology

- Corpus:
    - A dataset of text
    - E.g. newspaper articles, tweets
- Document:
    - A single entry from our corpus
    - E.g. article, sentence, tweet
- Vocabulary:
    - All the words that appear in the corpus
- Token:
    - An entity
    - E.g. a word

Sample Sentence

“Students are learning from other students”

Tokenization

Take the document and split it into a list of tokens

“Students are learning from other students”

[“Students”, “are”, “learning”, “from”, “other”, “students”]

Lower case conversion

[“Students”, “are”, “learning”, “from”, “other”, “students”]

[“students”, “are”, “learning”, “from”, “other”, “students”]

Stop word removal

- Remove the words we ignore in our analysis that are too common to be useful (since they are too common, they do not provide predictive power)

- Sklearn and nltk have standard list of stop words

[“students”, “are”, “learning”, “from”, “other”, “students”]

[“students”, “learning”, “other”, “students”]

Stemming/Lemmatization

- Stemming:
  - Removes morphical suffixes
    - speaking -> speak
    - actually -> actual
  - Does it without context
  - Sometimes weird
  - Pretty fast

- Lemmatization:
  - Replace words with canonical form
    - better -> good
    - speaking -> speak
  - Uses hash tables
  - Slower

Stemming/Lemmatization

[“students”, “learning”, “other”, “students”]

⇩

[“student”, “learn”, “other”, “student”]

Corpus with 3 documents

Doc 1: “Students are learning from other students”

- [“student”, “learn”, “other”, “student”]

Doc 2: “I am teaching at Galvanize”

- [“teach”, “galvanize”]

Doc 3: “There are students learning at Galvanize”

- [“student”, “learn”, “galvanize”]

Bag of words

A document represented as a vector of word counts is called "bag of words"

Vector for our corpus: (galvanize, learn, other, student, teach)

document	galvanize	learn	other	student	teach
Doc 1					
Doc 2					
Doc 3					

Bag of words

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

Vector for our corpus: (galvanize, learn, other, student, teach)

document	galvanize	learn	other	student	teach
Doc 1	0	1	1	2	0
Doc 2	1	0	0	0	1
Doc 3	1	1	0	1	0

Bag of words

- Issues with bag of words:
  - Word counts
    - Counts emphasize results from longer documents
  - Every word has equal weighting
    - “other” and “student” have different predictive power.

Term Frequency - Inverse Document Frequency

- Measures the relevance of a word
- Adjusts word count by document length and how often it shows up in the corpus

Term Frequency

Normalize counts within a document to frequency

$$tf(t,d) = \frac{total\ count\ of term\ t\ in\ document\ d}{total\ count\ of\ all\ terms\ in\ document\ d}$$

document	galvanize	learn	other	student	teach
Doc 1					
Doc 2					
Doc 3					

Term Frequency

Normalize counts within a document to frequency

$$tf(t,d) = \frac{total\ count\ of term\ t\ in\ document\ d}{total\ count\ of\ all\ terms\ in\ document\ d}$$

document	galvanize	learn	other	student	teach
Doc 1	0	¼ = 0.25	¼ = 0.25	2/4 = 0.5	0
Doc 2	½ = 0.5	0	0	0	½ = 0.5
Doc 3	⅓ = 0.33	⅓ = 0.33	0	⅓ = 0.33	0

Term Frequency

Issues with term frequency:

- Words found only in one document should have highest weighting
- Words found in every document should have lowest weighting

Inverse Document Frequency

$$idf(t, D) = log\frac{total\ number\ of\ document\ incorpus\ D}{count\ of\ document\ containing\ term\ t}$$

document	galvanize	learn	other	student	teach
Doc 1		X	X	X	
Doc 2	X				X
Doc 3	X	X		X	
idf(t,D)	log(3/2)	log(3/2)	log(3/1)	log(3/2)	log(3/1)

TF-IDF

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

document	galvanize	learn	other	student	teach
Doc 1	0	0.25xlog(3/2) = 0.101	0.25xlog(3/1) = 0.275	0.5xlog(3/2) = 0.203	0
Doc 2	0.5xlog(3/2) = 0.203	0	0	0	0.5xlog(3/1) = .549
Doc 3	0.33xlog(3/2) = 0.135	0.33xlog(3/2) = 0.135	0	0.33xlog(3/2) = 0.135	0

Comparing TF-IDF vectors of documents

Cosine similarity: $$similarity = cos\theta = \frac{A \cdot B}{\|A\| \, \|B\|}$$

- Doc 1 vs. Doc 2:
  - [“student”, “learn”, “other”, “student”] vs [“teach”, “galvanize”]

$$(0, 0.101, 0.275, 0.203, 0) \; vs. \; (0.203, 0, 0, 0, 0.275)$$

$$similarity = \frac{0}{0.36 \times 0.34} = 0$$

- Doc 1 vs. Doc 3:
  - [“student”, “learn”, “other”, “student”] vs [“student”, “learn”, “galvanize”]

$$(0, 0.101, 0.275, 0.203, 0) \; vs. \; (0.135, 0.135, 0, 0.135, 0)$$

$$similarity = \frac{0.041}{0.36 \times 0.23} = 0.34$$

N-Grams & Skip Grams

- Sliding window on text
- Handle phrases or words that go together (to capture the relationship between the consecutive words)

“The rain in Spain falls mainly on the plain”

<u>Bi-gram:</u> [“the rain”, “rain in”, “in spain”, “spain falls”, …]

<u>1-skip-bi-gram:</u> [“the in”, “rain spain”, “in falls”, ...]

Morning Assignment

Document Classification with Naive Bayes

Objectives for the afternoon

- Explain why Naive Bayes is naive
- Describe when it is best to use Naive Bayes
- Explain why we need Laplace Smoothing
- Implement a Naive Bayes algorithm

Motivation

Data	Labels
Email 1	spam
Email 2	not spam
.	.
.	.
Email n	not spam

Motivation

Data	Labels
Email 1	spam
Email 2	not spam
.	.
.	.
Email n	not spam

in this case

features >> observations

(p >> n)

Naive Bayes

- Computationally very efficient for high dimensions
  - Probabilities
  - No distance calculations

Goal

Classify Spam:

We want P(spam | email) or P(not spam | email)

Bayes Rule

$$P(spam|email) = \frac{P(email|spam)P(spam)}{P(email)}$$

$$P(spam) = \frac{\#\ spams}{\#\ all\ emails}$$

“Naive” Bayes

$$P(email|spam)$$

- Break down into words
- Assume independence between words (“naive”)

Naive Bayes

$$P(email|spam) = \prod_{i=1}^{p} P(word_i|spam)$$

$$P(word_i|spam) = \frac{\#\ of\ word_i\ in\ spam\ emails}{\#of\ total\ words\ in\ spam\ emails}$$

What happens if a word is not in our corpus?

Naive Bayes

Laplace smoothing:

- We want to prevent probabilities of 0
- Assume that every event happens at least α times

$$P(word_i|spam) = \frac{\#\ of\ word_i\ in\ spam\ emails + \alpha}{\#\ of\ words\ in\ spam\ emails + \alpha p}$$

- *p* is the number of features (i.e. number of words in your corpus)
- α ~ 0.01

Naive Bayes

$$P(spam|email) = P(spam) \times \prod_{i=1}^{p} P(word_i|spam)$$

$$log(P(spam|email)) = \\ log(P(spam)) + \sum_{i=1}^{p} log(P(word_i|spam))$$

Afternoon Assignment