Announcements

HW2

- Due Mon 2/28

Course feedback

- Thank you!
- Stay tuned to Piazza for summary of responses

Plan

Wrap up Graphs

Free text and NLP

15-388/688 - Practical Data Science:
Free text and natural language processing

Pat Virtue
Carnegie Mellon University
Spring 2022

Slide credits: CMU AI, Zico Kolter

Outline

Free text in data science

Bag of words and TFIDF

Language models and N-grams

Outline

Free text in data science

Bag of words and TFIDF

Language models and N-grams

Free text in data science vs. NLP

A large amount of data in many real-world data sets comes in the form of free text (user comments, but also any “unstructured” field)

(Computational) natural language processing: write computer programs that can understand natural language

This lecture: try to get some meaningful information out of unstructured text data

Understanding language is hard

Multiple potential parse trees:

> “While hunting in Africa, I shot an elephant in my pajamas. How he got into my pajamas, I don't know.” – Groucho Marx

Winograd schemas:

> “The city council members refused the demonstrators a permit because they [feared/advocated] violence.”

Basic point: We use an incredible amount of context to understand what natural language sentences mean

But is it always hard?

Two reviews for a movie (Star Wars Episode 7)

1. "… truly, a stunning exercise in large-scale filmmaking; a beautifully-assembled picture in which Abrams combines a magnificent cast with a marvelous flair for big-screen, sci-fi storytelling."

2. "It's loud and full of vim -- but a little hollow and heartless."

Which one is positive?

We can often very easily tell the "overall gist" of natural language text without understanding the sentences at all

But is it always hard?

Two reviews for a movie (Star Wars Episode 7):

1. "… truly, a **stunning** exercise in large-scale filmmaking; a beautifully-assembled picture in which Abrams combines a **magnificent** cast with a **marvelous** flair for big-screen, sci-fi storytelling."

2. "It's loud and full of vim -- but a little **hollow** and **heartless**."

Which one is positive?

We can often very easily tell the "overall gist" of natural language text without understanding the sentences at all

Natural language processing for data science

In many data science problems, we don't need to truly understand the text in order to accomplish our ultimate goals (e.g., use the text in forming another prediction)

In this lecture we will discuss two simple but very useful techniques that can be used to infer some meaning from text *without* deep understanding

1. Bag of words approaches and TFIDF matrices
2. N-gram language models

Note: These methods are no longer sufficient for text processing in data science, due to advent of deep-learning-based text models; in later lectures we will cover deep learning methods for text

Natural language processing for data science

Some key ideas that are also important to understand deep learning approaches to text/language

- Representing text as a vector (of numbers)

- Measuring distance (or similarity) between vectors

- Language models

Outline

Free text in data science

Bag of words and TFIDF

Language models and N-grams

Brief note on terminology

In this lecture, we will talk about “documents”, which mean individual groups of free text

(Could be actual documents, or e.g. separate text entries in a table)

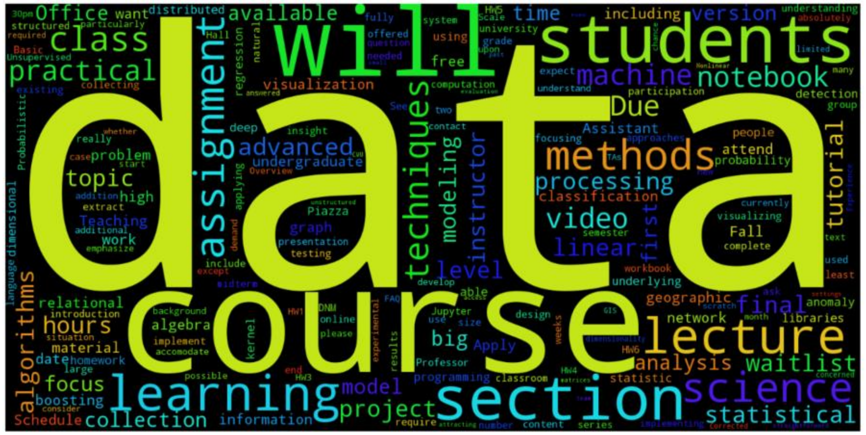
“Words” or “terms” refer to individual words (tokens separated by whitespace) and often also punctuation

“Corpus” refers to a collection of documents

“Vocabulary” or “dictionary” is the set of possible words in a language or just the set of unique terms in a corpus

Bag of words

AKA, the word cloud view of documents



Word cloud of class webpage

Represent each document as a vector of word frequencies

Order of words is irrelevant, only matters how often words occur

Bag of words example

“The goal of this lecture is to explain the basics of free text processing”

“The bag of words model is one such approach”

“Text processing via bag of words”

$$X = \begin{array}{ccccccccccc|l}
\text{the} & \text{is} & \text{of} & \text{goal} & \text{lecture} & \text{bag} & \text{words} & \text{via} & \text{text} & & \text{approach} & \\
2 & 1 & 2 & 1 & 1 & 0 & 0 & 0 & 1 & & 0 & \text{Document 1} \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & \dots & 1 & \text{Document 2} \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & & 0 & \text{Document 3}
\end{array}$$

Term frequency

“Term frequency” just refers to the counts of each word in a document

Denoted $\mathrm{tf}_{i,j}$ = frequency of word j in document i (sometimes indices are reversed, we use these for consistency with matrix above)

Often (as in the previous slide), this just means the raw count, but there are also other possibilities

1. $\mathrm{tf}_{i,j} \in \{0,1\}$ – does word occur in document or not
2. $\log\left(1 + \mathrm{tf}_{i,j}\right)$ – log scaling of counts
3. $\mathrm{tf}_{i,j} / \max_j \mathrm{tf}_{i,j}$ – scale by document’s most frequent word

Inverse document frequency

Term frequencies tend to be “overloaded” with very common words (“the”, “is”, “of”, etc). Note: really common words, called “stop words,” may even be removed.

Idea of *inverse document frequency* weight words negatively in proportion to how often they occur in the entire set of documents

$$\mathrm{idf}_j = \log\left(\frac{\#\ \mathrm{documents}}{\#\ \mathrm{documents\ with\ word}\ j}\right)$$

As with term frequency, there are other version as well with different scalings, but the log scaling above is most common

Note that inverse document frequency is just defined for *words* not for word-document pairs, like term frequency

Inverse document frequency examples

$$X = \begin{array}{ccccccccccc|l} \text{the} & \text{is} & \text{of} & \text{goal} & \text{lecture} & \text{bag} & \text{words} & \text{via} & \text{text} & & \text{approach} & \\ \hline 2 & 1 & 2 & 1 & 1 & 0 & 0 & 0 & 1 & & 0 & \text{Document 1} \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & \ldots & 1 & \text{Document 2} \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & & 0 & \text{Document 3} \end{array}$$

$$\text{idf}_{\text{of}} = \log\left(\frac{3}{3}\right) = 0$$

$$\text{idf}_{\text{is}} = \log\left(\frac{3}{2}\right) = 0.405$$

$$\text{idf}_{\text{goal}} = \log\left(\frac{3}{1}\right) = 1.098$$

TFIDF

Term frequency inverse document frequency = $\mathrm{tf}_{i,j} \times \mathrm{idf}_j$

Just replace the entries in the X matrix with their TFIDF score instead of their raw counts (also common to remove “stop words” beforehand)

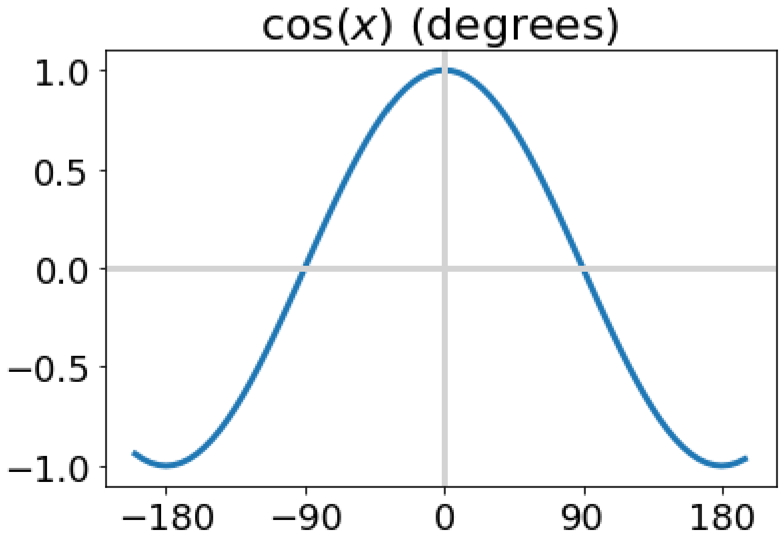
This seems to work much better than using raw scores for e.g. computing similarity between documents or building machine learning classifiers on the documents

$$X = \begin{array}{c} \begin{array}{ccccc} \text{the} & \text{is} & \text{of} & \text{goal} & \\ \end{array} \\ \begin{bmatrix} 0.8 & 0.4 & 0 & 1.1 & \\ 0.4 & 0.4 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & \end{bmatrix} \end{array}$$

Cosine similarity

Geometric interpretation: two vectors are similar if the angle between them is small (regardless of the magnitude of the vectors)

$\text{CosineSimilarity}(x, \text{z}) = \cos(\theta)$, where θ is the angle between x and z



Cosine similarity

A fancy name for “normalized inner product”

Given two documents x, z represented by TFIDF vectors (or just term frequency vectors), cosine similarity is just

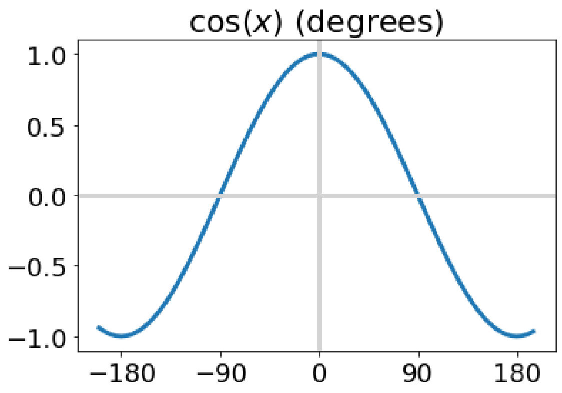
$$\mathrm{CosineSimilarity}(x, \mathrm{z}) = \frac{x^T z}{\|x\|_2 \cdot \|z\|_2} = \cos(\theta)$$

Higher numbers mean documents more similar

Equivalent to the (1 minus) the squared distance between the two normalized document vectors

$$\frac{1}{2}\|\tilde{x} - \tilde{z}\|_2^2 = 1 - \mathrm{CosineSimilarity}(x, z), \qquad \text{where } \tilde{x} = \frac{x}{\|x\|_2}, \tilde{z} = \frac{z}{\|z\|_2}$$

Poll 1



What is the range of possible values for $\mathrm{CosineSimilarity}(x, \mathrm{z})$?

A. $[0, 1]$ B. $[0, \infty)$ C. $[-1, 1]$ D. $(-\infty, \infty)$ E. None of the above

$\mathrm{CosineSimilarity}(x, \mathrm{z}) = \cos(\theta)$, where θ is the angle between x and z

$$\mathrm{CosineSimilarity}(x, \mathrm{z}) = \frac{x^T z}{\|x\|_2 \cdot \|z\|_2}$$

$$\frac{1}{2}\|\tilde{x} - \tilde{z}\|_2^2 = 1 - \mathrm{CosineSimilarity}(x, z), \qquad \text{where } \tilde{x} = \frac{x}{\|x\|_2}, \tilde{z} = \frac{z}{\|z\|_2}$$

Cosine similarity example

“The goal of this lecture is to explain the basics of free text processing”

“The bag of words model is one such approach”

“Text processing via bag of words”

$$M = \begin{bmatrix} 1 & 0.068 & 0.078 \\ 0.068 & 1 & 0.103 \\ 0.078 & 0.103 & 1 \end{bmatrix}$$

Term frequencies as vectors

You you think of individual words in a term-frequencies model as being “one-hot” vectors in an $\#\text{words}$ dimensional space (here $\#\text{words}$ is total number of unique words in corpus)

$$\text{“pittsburgh”} \equiv e_{\text{pittsburgh}} \in \mathbb{R}^{\#\text{words}} = \begin{bmatrix} \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix} \begin{matrix} \\ \text{pitted} \\ \text{pittsburgh} \\ \text{pivot} \\ \\ \end{matrix}$$

Document vectors are sums of their word vectors

$$x_{\text{doc}} = \sum_{\text{word} \in \text{doc}} e_{\text{word}}$$

“Distances” between words

No notion of similarity in term frequency vector space:

$$\left\|e_{\text{pittsburgh}} - e_{\text{boston}}\right\|_2 = \left\|e_{\text{pittsburgh}} - e_{\text{banana}}\right\|_2 = 1$$

But, some words are inherently more related that others

- “Pittsburgh has some excellent new restaurants”
- “Boston is a city with great cuisine”
- “PostgreSQL is a relational database management system”

Under TFIDF cosine similarity (if we don’t remove stop words), then the second two sentences are more similar than the first and second

- Preview of *word embeddings*, to be discussed in later lecture

Outline

Free text in data science

Bag of words and TFIDF

Language models and N-grams

Language models

While the bag of words model is surprisingly effective, it is clearly throwing away a lot of information about the text

The terms “boring movie and not great” is not the same in a movie review as “great movie and not boring”, but they have the exact same bag of words representations

To move beyond this, we would like to build a more accurate model of how words really relate to each other: language model

Probabilistic language models

We haven’t covered probability much yet, but with apologies for some forward references, a (probabilistic) language model aims at providing a probability distribution over every word, given all the words before it

$$P(\text{word}_i | \text{word}_1, \ldots, \text{word}_{i-1})$$

E.g., you probably have a pretty good sense of what the next word should be:

- “Data science is the study and practice of how we can extract insight and knowledge from large amounts of”

$$P(\text{word}_i = \text{“data”} | \text{word}_1, \ldots, \text{word}_{i-1}) = ?$$
$$P(\text{word}_i = \text{“pizza”} | \text{word}_1, \ldots, \text{word}_{i-1}) = ?$$

Building language models

Building a language model that captures the true probabilities of natural language is still a distant goal

Instead, we make simplifying assumptions to build approximate tractable models

N-gram model: the probability of a word depends only on the $n - 1$ words preceding it

$$P(\mathrm{word}_i | \mathrm{word}_1, \ldots, \mathrm{word}_{i-1}) \approx P(\mathrm{word}_i | \mathrm{word}_{i-n+1}, \ldots, \mathrm{word}_{i-1})$$

This puts a hard limit on the *context* that we can use to make a prediction, but also makes the modeling more tractable

“large amounts of data” vs. “large amounts of pizza”

Estimating probabilities

A simple way (but *not* the only way) to estimate the conditional probabilities is simply by counting

$$P(\mathrm{word}_i | \mathrm{word}_{i-n+1}, \ldots, \mathrm{word}_{i-1}) = \frac{\#(\mathrm{word}_{i-n+1}, \ldots, \mathrm{word}_{\mathrm{i}})}{\#(\mathrm{word}_{i-n+1}, \ldots, \mathrm{word}_{i-1})}$$

E.g.:

$$P(\text{“data”}|\text{“large amounts of”}) = \frac{\#(\text{“large amounts of data”})}{\#(\text{“large amounts of”})}$$

Example of estimating probabilities

Very short corpus:

“The goal of this lecture is to explain the basics of free text processing”

Using a 2-gram model

$$P(\text{word}_i | \text{word}_{i-1} = \text{“of”}) = ?$$

Laplace smoothing

Estimating language models with raw counts tends to estimate a lot of zero probabilities (especially if estimating the probability of some new text that was not used to build the model)

Simple solution: allow for any word to appear with some small probability

$$P(\mathrm{word}_i|\mathrm{word}_{i-n+1},\ldots,\mathrm{word}_{i-1}) = \frac{\#(\mathrm{word}_{i-n+1},\ldots,\mathrm{word}_{\mathrm{i}}) + \alpha}{\#(\mathrm{word}_{i-n+1},\ldots,\mathrm{word}_{i-1}) + \alpha D}$$

where α is some number and D is total size of dictionary

Also possible to have “backoffs” that use a lower degree n-gram when the probability is zero

Laplace smoothing

$$P(\text{word}_i|\text{word}_{i-n+1},\ldots,\text{word}_{i-1}) = \frac{\#(\text{word}_{i-n+1},\ldots,\text{word}_\text{i}) + \alpha}{\#(\text{word}_{i-n+1},\ldots,\text{word}_{i-1}) + \alpha D}$$

Example:

Assume vocabulary with D words and $\alpha = 1$ (add-one smoothing)

Corpus with just three sentences:

“San Francisco is in California”
“The Golden Gate bridge is in San Francisco”
“Oakland is near San Francisco”

$$P(\text{"Francisco"} \mid \text{"San"}) = \frac{\#(\text{"San Francisco"}) + 1}{\#(\text{"San"}) + D} = \frac{3}{3+D}$$

$$P(\text{"Antonio"} \mid \text{"San"}) = \frac{\#(\text{"San Antonio"}) + 1}{\#(\text{"San"}) + D} = \frac{1}{3+D}$$

How do we pick n?

Hyperparameter

Lower n: less context, but more samples of each possible n-gram

Higher n: more context, but less samples

“Correct” choice is to use some measure of held-out cross-validation

In practice: use $n = 3$ for large datasets (i.e., triplets) , $n = 2$ for small ones

Examples

Random samples from language model trained on Shakespeare:

n=1: "in as , stands gods revenge ! france pitch good in fair hoist an what fair shallow-rooted , . that with wherefore it what a as your . , powers course which thee dalliance all"

n=2: "look you may i have given them to the dank here to the jaws of tune of great difference of ladies . o that did contemn what of ear is shorter time ; yet seems to"

n=3: "believe , they all confess that you withhold his levied host , having brought the fatal bowels of the pope ! ' and that this distemper'd messenger of heaven , since thou deniest the gentle desdemona ,"

More examples

n=7: `“so express'd : but what of that ? 'twere good you do so much for charity . i cannot find it ; 'tis not in the bond . you , merchant , have you any thing to say ? but little”`

This is starting to look a lot like Shakespeare, because it is Shakespeare

As we have higher order n-grams, the previous (n-1) words have only appeared very few times in the corpus, so we will always just sample the next word that occurred

Evaluating language models

How do we know how well a language model performs

Common strategy is to estimate the probability of some held out portion of data, and evaluate *perplexity*

$$\text{Perplexity} = 2^{-\frac{\log_2 P(\text{word}_1,\ldots\text{word}_N)}{N}} = \left(\frac{1}{P(\text{word}_1,\ldots\text{word}_N)}\right)^{\frac{1}{N}}$$

where we can evaluate the probability using

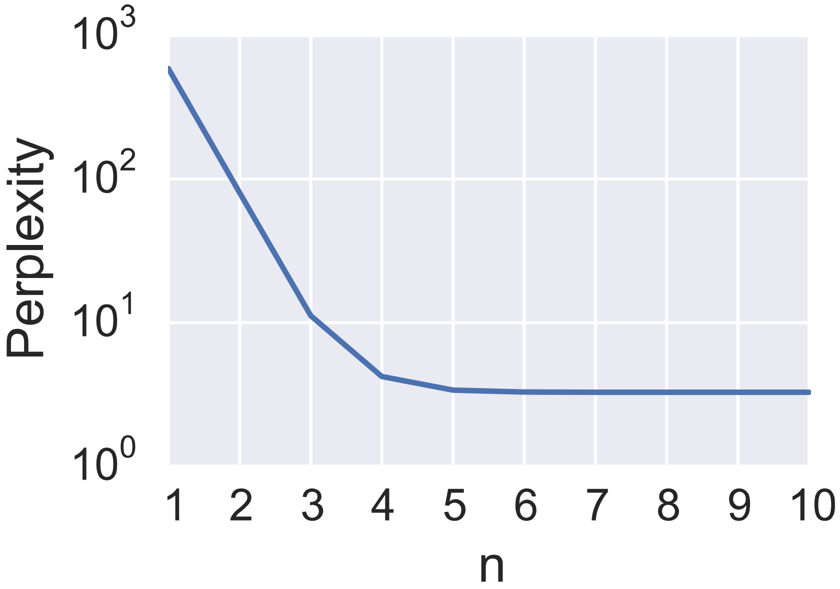
$$P(\text{word}_1,\ldots\text{word}_n) = \prod_{i=n}^{N} P(\text{word}_i|\text{word}_{i-n+1},\ldots,\text{word}_{i-1})$$

(note that you can compute the log of this quantity directly)

Evaluating perplexity

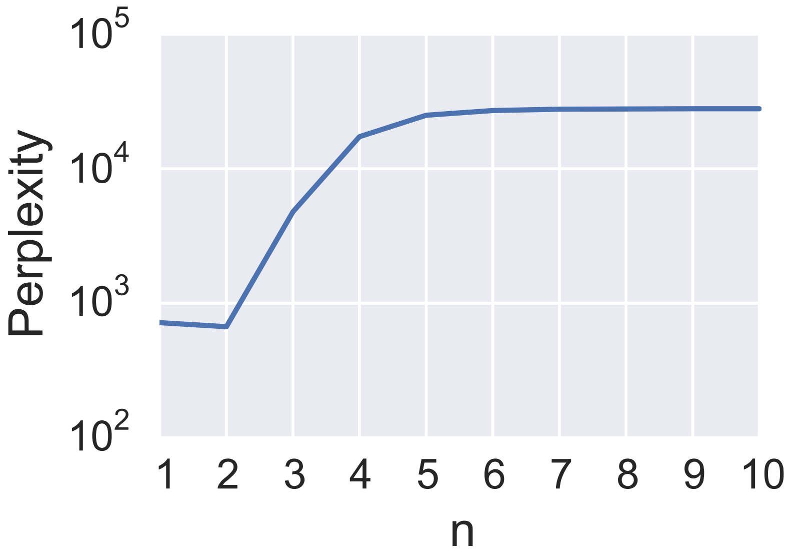
Perplexity on the corpus used to build the model will always decrease using higher n (fewer choices of what comes next means higher probability of observed data)

Note: this is only strictly true when $\alpha = 0$



Evaluating perplexity

What really matters is how well the model captures text from the “same” distribution that was *not* used to train the model



This is a preview of overfitting/model selection, which we will talk about a lot in the machine learning section