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

J. Zico Kolter
Carnegie Mellon University

Fall 2016

Announcements

HW 3 out tonight

Feedback on tutorial topics sent tomorrow

Some minor glitch with 388 <-> 688 switch poll, everyone who replied should have been contacted, and we will finalize these switches tomorrow

Some guidance on tutorial: going to replace the geographic information systems lecture with a tutorial that can serve as an example of what we are looking for

Switching website hosts tomorrow AM, website may be down briefly

Outline

Free text in data science

Bag of words and TFIDF

Language models and N-grams

(Next time) Libraries for handling free text

Outline

Free text in data science

Bag of words and TFIDF

Language models and N-grams

Libraries for handling free text

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 councilmen 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 (the latest Star Wars):

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 (the latest Star Wars):

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: this lecture may be subject to change in the upcoming years, as massive improvements in "off-the-shelf" language understanding are ongoing (for example, this has already happened to image understanding)

Outline

Free text in data science

Bag of words and TFIDF

Language models and N-grams

Libraries for handling free text

Brief note on terminology

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

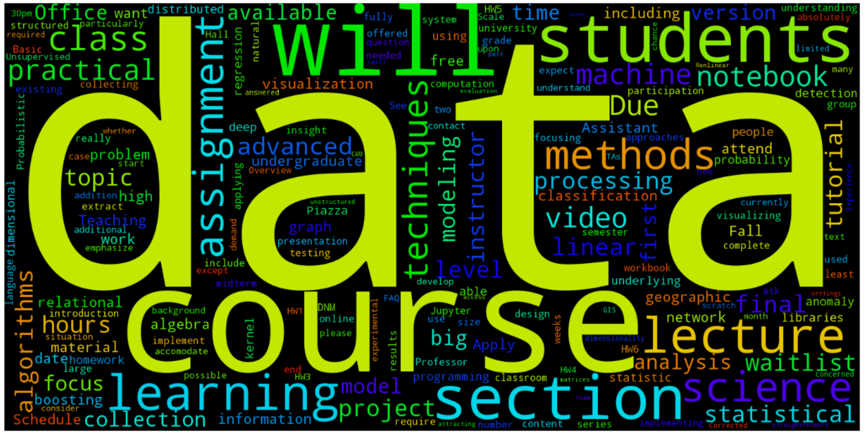
(Could be actual documents, or e.g. separate text entries in a table)

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

“Corpus” refers to a collection of documents

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”

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

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. $\log(1 + \mathrm{tf}_{i,j})$ – 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)

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

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

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

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

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

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

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

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

A fancy name for “normalized inner product”

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

$$\text{Cosine-Similarity} = \frac{x^T y}{\|x\|_2 \times \|y\|_2}$$

Between zero and one, higher numbers mean the documents are more similar

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”

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

Outline

Free text in data science

Bag of words and TFIDF

Language models and N-grams

Libraries for handling free text

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”

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

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 but tractable models

$\boldsymbol{n}$**-gram model:** the probability of a word depends only on the $n-1$ word preceding it

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

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 hotdogs”

Estimating probabilities

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

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

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 an 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

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

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

How do we pick n?

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

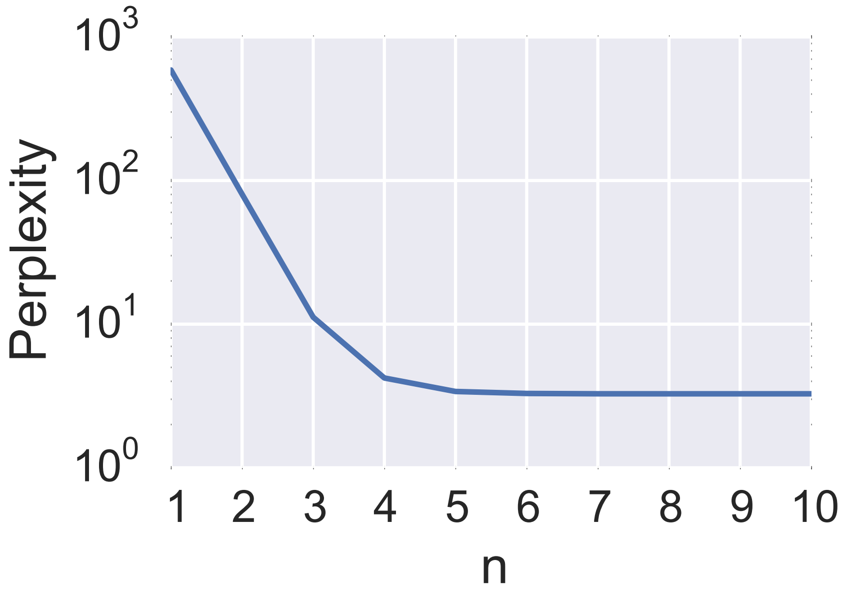
$$P(\text{word}_1, \ldots \text{word}_n) = \prod_{i=n}^{N} P(\text{word}_i | \text{word}_{i-n+1}, \ldots, \text{word}_{i-1})$$

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

Evaluating perplexity

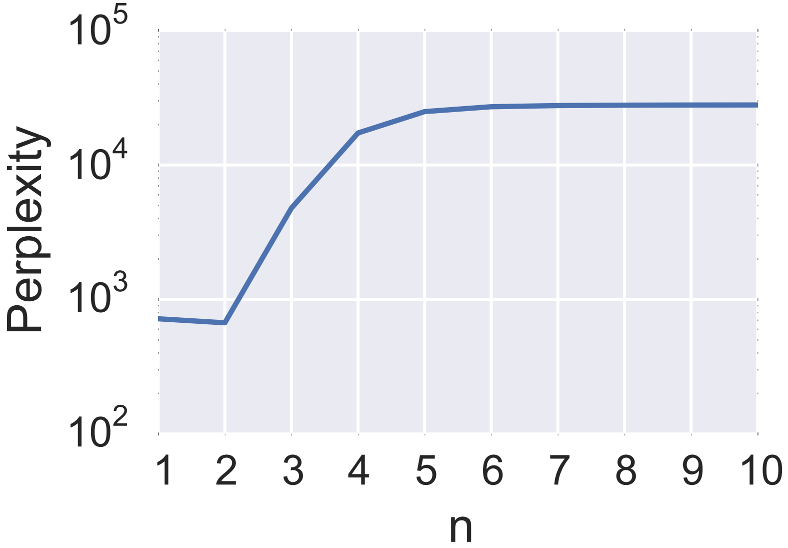
Perplexity on the corpus used to build the model will always decrease using higher n (fewer choices of what comes next means higher probability of observed data)

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

Outline

Free text in data science

Bag of words and TFIDF

Language models and N-grams

Libraries for handling free text

NLTK library

The NLTK (natural language toolkit) library (http://www.nltk.org) is the standard Python library for handling text and natural language data

Note: NLTK is a massive library, and is a bit more geared towards things like tagging, parsing, and more complex processes instead of the techniques described previously

Additionally, it actually doesn't contain much of what we want to do (no TFIDF creation, there was an n-gram language model but it was removed due to bugs)

You may want to look at some other options: spacy, CoreNLP

Reading and tagging documents

Load nltk and download necessary files:

```
import nltk
import nltk.corpus
#nltk.download() # just run this once
```

Tokenize a document

```
sentence = "The goal of this lecture isn't to explain complex free text
processing"
tokens = nltk.word_tokenize(sentence)
# ['The', 'goal', 'of', 'this', 'lecture', 'is', "n't", 'to', 'explain',
'complex', 'free', 'text', 'processing']
```

Tag parts of speech

```
pos = nltk.pos_tag(tokens)
# [('The', 'DT'), ('goal', 'NN'), ('of', 'IN'), ('this', 'DT'),
('lecture', 'NN'), ('is', 'VBZ'), ("n't", 'RB'), ('to', 'TO'),
('explain', 'VB'), ('complex', 'JJ'), ('free', 'JJ'), ('text', 'NN'),
('processing', 'NN')]
```

Stop words and n-grams

Get list of English stop words (common words)

```
stopwords = nltk.corpus.stopwords.words("English")
print [a for a in tokens if a.lower() not in stopwords]
# ['goal', 'lecture', "n't", 'explain', 'complex', 'free', 'text',
'processing']
```

Generate n-grams from document

```
list(nltk.ngrams(tokens, 3))
# [('The', 'goal', 'of'), ('goal', 'of', 'this'), ('of', 'this',
'lecture'), ('this', 'lecture', 'is'), ('lecture', 'is', "n't"), ('is',
"n't", 'to'), ("n't", 'to', 'explain'), ('to', 'explain', 'complex'),
('explain', 'complex', 'free'), ('complex', 'free', 'text'), ('free',
'text', 'processing')]

# code below does the same thing, without nltk
zip(*[tokens[i:] for i in range(3)])
```

Scikit Learn library

Scikit Learn library (http://scikit-learn.org/) is one of the standard Python libraries for machine learning

This lecture is odd place to introduce it (we'll use Scikit Learn much more in the machine learning portions of this course), but the library also contains useful routines for building word count matrices

Creating “TFIDF” matrix

Create frequency counts and TFIDF matrices

```
data = [
    "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"
]


from sklearn.feature_extraction.text import TfidfTransformer, 
CountVectorizer
vectorizer = CountVectorizer(stop_words='english')
transformer = TfidfTransformer()

# get frequency counts (sparse) matrix
freq_matrix = vectorizer.fit_transform(data)

# get TFIDF (sparse) matrix; Note: uses tf*(1+idf), not previous formula
tfidf_matrix = transformer.fit_transform(freq_matrix)
```