Natural Language Processing (NLP)

Scott Contri - 2016

Motivation

- Allows us to apply Machine Learning Techniques to unstructured text by transforming it to vectors.
  - E.g. Classification (spam), clustering (news), even regression!
- Used in Industry.
  - Machine Translation
  - Sentiment Analysis
  - Text Classification
  - Necessary for Speech Recognition

Objectives

- Understand NLP terminology.
- Work Through Text Featurization Pipeline.
    - Tokenization, Stop Words, Stemming/Lemmatization, Bag of Words
- Become familiar with Tf-Idf.
- Compare Tf-Idf vectors using Cosine Similarity.
- Jupyter Notebook the s%$t out of nltk and sklearn

Terminology

- Corpus:
  - A dataset of text (e.g. articles, tweets, journals)
- Document:
  - A single entry from the corpus (e.g. article, tweet, sentence)
- Vocabulary:
  - All words that appear in the corpus
- Token:
  - A single entity (e.g. word, term)

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

- We convert words to lowercase. Surprise! (very useful though)

[‘Students’, ‘are’, ‘learning’, ‘from’, ‘other’, ‘students’]

[‘students’, ‘are’, ‘learning’, ‘from’, ‘other’, ‘students’]

Gettin' Rid of Stop Words

- Words that are too common to be useful in our analysis of text (e.g. 'the', 'and', 'as'). Both nltk and sklearn have a standard list of stop words.

['students', 'are', 'learning', 'from', 'other', 'students']

↓

['students', 'learning', 'other', 'students']

Stemming/Lemmatization

- Both methods reduce words to a root/base form.
  - Stemming - a somewhat crude character based method that removes common morphological endings (e.g. ‘-ed’, ‘-ing’, ‘-al’)
    - Ex: ‘cars’ -> ‘car’, ‘watches’ -> ‘watch’
  - Lemmatization - reduces word down to their lemma, or dictionary form
    - Ex: ‘cars’ -> ‘automobile’, ‘better’ -> ‘good’
  - When to use which depends on use case and toolkit you are calling from (e.g. PorterStemmer, SnowballStemmer, WordNetLemmatizer)

Stemming/Lemmatization Ex.

[‘students’, ‘are’, ‘learning’, ‘from’, ‘other’, ‘students’]

↓

[‘student’, ‘learn, ‘other’, ‘student’]

STEMMED!!!

Corpus of 3 Documents

1. Students are learning from other students
2. I am teaching at Galvanize
3. There are students learning at Galvanize

1. [‘student’, ‘learn’, ‘other’, ‘student’]
2. [‘teach’, ‘galvanize’]
3. [‘student’, ‘learn’, ‘galvanize’]

Bag of Words

- A corpus with documents represented as vectors of word counts is called a ‘bag of words’
- Vocabulary/Features for our corpus: (galvanize, learn, other, student, teach)

document	galvanize	learn	other	student	teach
[‘student’, ‘learn’, ‘other’, ‘student’]					
[‘teach’, ‘galvanize’]					
[‘student’, ‘learn’, ‘galvanize’]					

Bag of Words

- A corpus with documents represented as vectors of word counts is called a ‘bag of words’
- Vocabulary for our corpus: (galvanize, learn, other, student, teach)

document	galvanize	learn	other	student	teach
[‘student’, ‘learn’, ‘other’, ‘student’]	0	1	1	2	0
[‘teach’, ‘galvanize’]	1	0	0	0	1
[‘student’, ‘learn’, ‘galvanize’]	1	1	0	1	0

Issues with Bag of Words

- Bag of words is naive (not in a cool way, like Naive Bayes)
    - Word counts aren’t good enough for me or you or anyone doing NLP
        - Counts emphasize results from longer documents.
    - Every word is given equal weighting if they occur the same number of times in a document.
        - Shouldn’t ‘learn’ and ‘galvanize’ have different predictive power?
- Is there a better way to featurize?
    - *hint* : the answer is an adverb used to express affirmation.

Term Frequency (Tf)

- Normalize counts within a document to frequency.

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

document	galvanize	learn	other	student	teach
[‘student’, ‘learn’, ‘other’, ‘student’]					
[‘teach’, ‘galvanize’]					
[‘student’, ‘learn’, ‘galvanize’]					

Term Frequency (Tf)

- Normalize counts within a document to frequency.

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

document	galvanize	learn	other	student	teach
['student', 'learn', 'other', 'student']	0	$\frac{1}{4} = .25$	$\frac{1}{4} = .25$	$\frac{2}{4} = .5$	0
['teach', 'galvanize']	$\frac{1}{2} = .5$	0	0	0	$\frac{1}{2} = .5$
['student', 'learn', 'galvanize']	$\frac{1}{3} = .333$	$\frac{1}{3} = .333$	0	$\frac{1}{3} = .333$	0

Issues with Tf

- Tf does not reflect the rareness of a word/token/term amongst documents
    - It only relays the rareness of a word in relation to the individual document being considered.
    - It doesn’t consider other documents
- It would make sense to incorporate how often a word occurs throughout the entire corpus.
    - ‘teach’ only occurs once and should therefore have a higher weighting, while ‘student’ occurs twice and should thus be assigned a lower weight.

Inverse Document Frequency (Idf)

- You know those issues with Tf I just addressed ? This helps take of those.

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

IT’S MIRACULOUS!

(Might be necessary to add 1 into denominator to prevent division by zero. This is called smoothing.)

Inverse Document Frequency (Idf)

$$\mathrm{idf}(\mathrm{t}, \mathrm{D}) = \log \frac{\text{total count of documents in corpus D}}{\text{count of documents containing term t}}$$

document	galvanize	learn	other	student	teach
[‘student’, ‘learn’, ‘other’, ‘student’]					
[‘teach’, ‘galvanize’]					
[‘student’, ‘learn’, ‘galvanize’]					
$idf(t, D)$					

Inverse Document Frequency (Idf)

$$\text{idf}(\text{t}, \text{D}) = \log \frac{\text{total count of documents in corpus D}}{\text{count of documents containing term t}}$$

document	galvanize	learn	other	student	teach
['student', 'learn', 'other', 'student']		X	X	X	
['teach', 'galvanize']	X				X
['student', 'learn', 'galvanize']	X	X		X	
$idf(t, D)$	$\log\frac{3}{2} = .405$	$\log\frac{3}{2} = .405$	$\log\frac{3}{1} = 1.10$	$\log\frac{3}{2} = .405$	$\log\frac{3}{1} = 1.10$

Tf-Idf

- By multiplying all tokens' Tf by their associated Idf we are able to apply the most weight to words that occur frequently in a document, but rarely occur the corpus.
  - Alternatively, words that occur infrequently in a document, but very frequently in the corpus are assigned the lowest weight.

Tf-Idf

$$\text{tf–idf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D)$$

document	galvanize	learn	other	student	teach
['student', 'learn', 'other', 'student']					
['teach', 'galvanize']					
['student', 'learn', 'galvanize']					

Tf-Idf

$$\text{tf–idf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D)$$

document	galvanize	learn	other	student	teach
['student', 'learn', 'other', 'student']	0	$.25 \times .405 = .101$	$.25 \times 1.10 = .275$	$.5 \times .405 = .203$	0
['teach', 'galvanize']	$.5 \times .405 = .203$	0	0	0	$.5 \times 1.10 = .549$
['student', 'learn', 'galvanize']	$.333 \times .405 = .135$	$.333 \times .405 = .135$	0	$.333 \times .405 = .135$	0

Cosine Similarity

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

- Used to compare Tf-Idf vectors of documents
- [‘student’, ‘learn’, ‘other’, ‘student’] vs. [‘teach’, ‘galvanize’]
    - (0, .101, .275, .203, 0) vs. (.203, 0, 0, 0, .275)
    - similarity = 0/(.36 x .34) = 0
- [‘student’, ‘learn’, ‘other’, ‘student’] vs. [‘student’, ‘learn’, ‘galvanize’]
    - (0, .101, .275, .203, 0) vs. (.135, .135, 0, .135, 0)
    - similarity = .041/(.36 x .23) = .34

Issues with Current NLP Pipeline

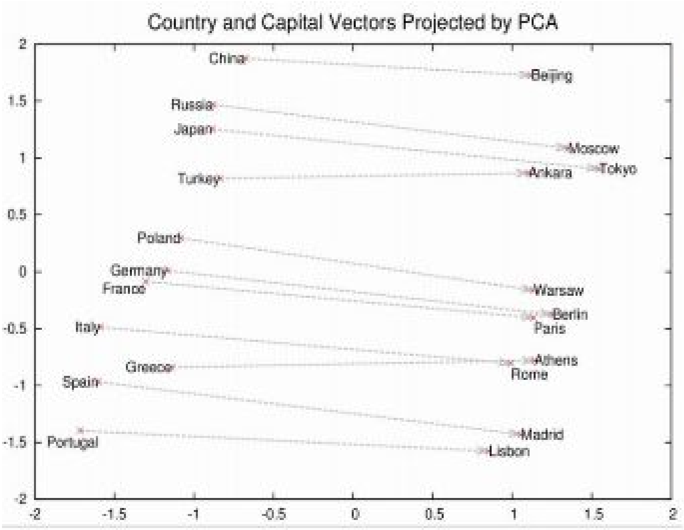
- We are losing a lot of structure/order.
- It’s only accounting for content, not flow or style.
- However, it is very good for search queries.

N-Grams, Skip Grams

- Both N-Grams and Skip Grams try to account for order and context.
- N-grams group words by some set ‘n’.
    - Ex [‘student’, ‘learn’, ‘other’, student’]
    - (*n=2)* => (student, learn),(learn, other), (other, student)
    - (*n=3)* => (student, learn, other), (learn, other, student)
- Skip Grams skip by some gap.
    - (*1-skip-2-gram)* => (student, other), (learn, student)

Word2Vec

- Geometric relationships between vectors represent semantic relationships between words.



vec(“man”) – vec(“king”) + vec(“woman”) = vec(“queen”)