STEVENS INSTITUTE OF TECHNOLOGY

Natural Language Processing (NLP)

Rensheng Wang,
https://sit.instructure.com/courses/29876
January 24, 2019

What is text?

You can think of text as a sequence of

- Characters
- Words
- Phrases and named entities
- Sentences
- Paragraphs
- ...

What is a word?

↳ a meaningful structured unit

- It seems natural to think of a text as a sequence of words
- ☞ A word is a meaningful sequence of characters
- How to find the boundaries of words?
- ☞ In English we can split a sentence by spaces or punctuation
- **Input:**

  Fridends, No smoking, Let us talk;
- **Output:**

  Friends | No | smoking | Let | us | talk

Tokenization

Tokenization is a process that splits an input sequence into so-called tokens

- You can think of a token as a useful unit for semantic processing
- Can be a word, sentence, paragraph, etc.

Tokenization

Tokenization is a process that splits an input sequence into so-called tokens

- You can think of a token as a useful unit for semantic processing
- Can be a word, sentence, paragraph, etc.

An example of simple whitespace tokenizer

- `nltk.tokenize.WhitespaceTokenizer`

  This | is | Tom's | book, | isn't | it?

☞ Problem: "it" and "it?" are different tokens with the same meaning

Tokenization

Lets try to also split by punctuation

- `nltk.tokenize.WordPunctTokenizer`

  This is Tom ' s book , isn ' t it ?

- Problem: "s", "isn", "'t" are not very meaningful

Tokenization

Lets try to also split by punctuation

- `nltk.tokenize.WordPunctTokenizer`

  This | is | Tom | ' | s | book | , | isn | ' | t | it | ?

- Problem: “s”, “isn”, “'t” are not very meaningful

We can come up with a set of rules

- `nltk.tokenize.TreebankWordTokenizer`

  This | is | Tom | 's | book | , | is | n't | it | ?

- “s” and “n't” are more meaningful for processing

Python tokenization example

- `import nltk`

Token normalization

We may want the same token for different forms of the word

- wolf, wolves $\rightarrow$ wolf
- talk, talks $\rightarrow$ talk

Stemming

- A process of removing and replacing suffixes to get to the root form of the word, which is called the **stem**
- Usually refers to heuristics that chop off suffixes

Lemmatization

- Usually refers to doing things properly with the use of a vocabulary and morphological analysis
- Returns the base or dictionary form of a word, which is known as the **lemma**

Stemming example

Porters stemmer

- 5 heuristic phases of word reductions, applied sequentially
- Example of phase 1 rules:

Rule	Example
SSES → SS	caresses → caress
IES → I	ponies → poni
SS → SS	caress → caress
S →	cats → cat

- `nltk.stem.PorterStemmer`
- Examples:

feet → feet	cats → cat
wolves → wolv	talked → talk

- Problem: fails on irregular forms, produces non-words

Lemmatization example

WordNet lemmatizer

- Uses the WordNet Database to lookup lemmas
- `nltk.stem.WordNetLemmatizer`
- Examples:

feet $\rightarrow$ foot	cats $\rightarrow$ cat
wolves $\rightarrow$ wolf	talked $\rightarrow$ talked

- Problems: not all forms are reduced

Lemmatization example

WordNet lemmatizer

- Uses the WordNet Database to lookup lemmas
- `nltk.stem.WordNetLemmatizer`
- Examples:

feet → foot	cats → cat
wolves → wolf	talked → talked

- Problems: not all forms are reduced

☞ Conclusion: we need to try stemming or lemmatization and choose best for our task

Python stemming example

Further normalization

Normalizing capital letters

- Us, us $\rightarrow$ us (if both are pronoun)
- us, US (could be pronoun and country)
- We can use heuristics:
  - lowercasing the beginning of the sentence
  - lowercasing words in titles
  - leave mid-sentence words as they are
- Or we can use machine learning to retrieve true casing $\rightarrow$ hard

Acronyms

- eta, e.t.a., E.T.A. $\rightarrow$ E.T.A.
- We can write a bunch of regular expressions $\rightarrow$ hard

Transforming tokens into features: BOW

Bag of words (BOW)

- Lets count occurrences of a particular token in our text
- Motivation: we are looking for marker words like "excellent" or "disappointed"
- For each token we will have a feature column, this is called text vectorization.

good movie
not a good movie
did not like

$\Rightarrow$

good	movie	not	a	did	like
1	1	0	0	0	0
1	1	1	1	0	0
0	0	1	0	1	1

1 if it occurs
0 if not

- Problems:
  - we loose word order, hence the name "bag of words"
  - counters are not normalized

Lets preserve some ordering

order of the sentence

We can count token pairs, triplets, etc.

- Also known as n-grams
  - 1-grams for tokens
  - 2-grams for token pairs
  - ...

good movie
not a good movie
did not like

$\Rightarrow$

good movie	movie	did not	a	...
1	1	0	0	...
1	1	1	1	...
0	0	1	0	...

- Problems:
  - too many features

Remove some n-grams

Lets remove some n-grams from features based on their occurrence frequency in documents of our corpus

Remove some n-grams

Lets remove some n-grams from features based on their occurrence frequency in documents of our corpus

- High frequency n-grams:
  - Articles, prepositions, etc. (example: and, a, the)
  - They are called **stop-words**, they won't help us to discriminate texts → remove them
- Lowfrequencyn-grams:
  - Typos, rare n-grams
  - We dont need them either, otherwise we will likely overfit
- Mediumfrequencyn-grams:
  - Those are good n-grams

Therere a lot of medium frequency n-grams

- It proved to be useful to look at n-gram frequency in our corpus for filtering out bad n-grams
- What if we use it for ranking of medium frequency n-grams?
- **Idea:** the n-gram with smaller frequency can be more discriminating because it can capture a specific issue in the review

TF-IDF

Term frequency (TF)

- tf(t, d) – frequency for term (or n-gram) t in document d
- Variants:

weighting scheme	TF weight
binary	0, 1
raw count	$f_{t,d}$
term frequency	$f_{t,d} / \sum_{t' \in d} f_{t',d}$
log normalization	$1 + \log(f_{t,d})$

TF-IDF

eg: finding no. of citations for a research paper

Inverse document frequency (IDF)

- $N = |D|$: total number of documents in corpus
- $|\{d \in D : t \in d\}|$: number of documents where the term t appears
- $\text{idf}(t, D) = \log \frac{N}{|\{d \in D:\ t \in d\}|}$

TF-IDF

local global how freqntly it appears? where?

- $\text{tfidf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D)$
- A high weight in TF-IDF is reached by a high term frequency (in the given document) and a low document frequency of the term in the whole collection of documents

Better BOW

- Replace counters with TF-IDF
- Normalize the result row-wise (divide by L_2-norm)

good movie
not a good movie
did not like

$\Rightarrow$

good movie	movie	did not	$\cdots$
0.17	0.17	0	$\cdots$
0.17	0.17	0	$\cdots$
0	0	0.47	$\cdots$

Python TF-IDF example

First text classification model: Sentiment classification

IMDB movie reviews dataset

- http://ai.stanford.edu/~amaas/data/sentiment/
- Contains 25000 positive and 25000 negative reviews
- Contains at most 30 reviews per movie
- At least 7 stars out of 10 $\rightarrow$ positive (label $= 1$)
- At most 4 stars out of 10 $\rightarrow$ negative (label $= 0$)
- 50/50 train/test split
- Evaluation:accuracy

Sentiment classification

Features: bag of 1-grams with TF-IDF values

- 25000 rows, 74849 columns for training
- Extremely sparse feature matrix : 99.8% are zeros

acting	actingjob	actings	actingwise
0.000000	0.0	0.0	0.0
0.000000	0.0	0.0	0.0
0.053504	0.0	0.0	0.0
0.033293	0.0	0.0	0.0
0.000000	0.0	0.0	0.0

Sentiment classification

Model: Logistic regression

- $p(\mathbf{y} = 1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x})$
- Linear classification model
- Can handle sparse data
- Fast to train
- Weights can be interpreted

Sentiment classification

Logistic regression over bag of 1-grams with TF-IDF

- Accuracy on test set: 88.5%
- Let's look at learnt weights:

ngram	weight
great	9.042803
excellent	8.487379
perfect	6.907277
best	6.440972
wonderful	6.237365

Top positive

VS

ngram	weight
worst	-12.748257
awful	-9.150810
bad	-8.974974
waste	-8.944854
boring	-8.340877

Top negative

Better sentiment classification

Let us try to add 2-grams

- Throw away n-grams seen less than 5 times
- 25000 rows, 156821 columns for training

and am	and amanda	and amateur	and amateurish	and amazing
0.068255	0.0	0.0	0.0	0.0
0.000000	0.0	0.0	0.0	0.0
0.000000	0.0	0.0	0.0	0.0
0.000000	0.0	0.0	0.0	0.0
0.000000	0.0	0.0	0.0	0.0

Better sentiment classification

Logistic regression over bag of 1,2-grams with TF-IDF

- Accuracy on test set: 89.9% (+1.5%)
- Lets look at learnt weights:

well worth	13.788515
best	13.633200
rare	13.570259
better than	13.500025

Near top positive

VS

bad	-24.467648
poor	-24.319746
the worst	-23.773352
waste	-22.880340

Near top negative

How to make it even better

- Play around with tokenization
  - Special tokens like emoji, :) and !!! can help Try to normalize tokens
- Adding stemming or lemmatization Try different models
  - SVM, Naive Bayes, ...
- Throw BOW away and use Deep Learning
  - `https://arxiv.org/pdf/1512.08183.pdf`
  - Accuracy on test set in 2016: 92.14% (+2.5%)