Introduction to Natural Language Processing

Lecture 2. Tokenization and word counts

Ekaterina Chernyak, Dmitry Ilvovsky

echernyak@hse.ru, dilvovsky@hse.ru

National Research University
Higher School of Economics (Moscow)

July 13, 2015

How many words?

“The rain in Spain stays mainly in the plain.”
9 **tokens**: The, rain, in, Spain, stays, mainly, in, the, plain
7 (or 8) **types**: T / the rain, in, Spain, stays, mainly, plain

Type and token

Type is an element of the vocabulary.
Token is an instance of that type in the text.

N = number of tokens
V - vocabulary (i.e. all types)
$|V|$ = size of vocabulary (i.e. number of types)
How are N and $|V|$ related?

Zipf's law

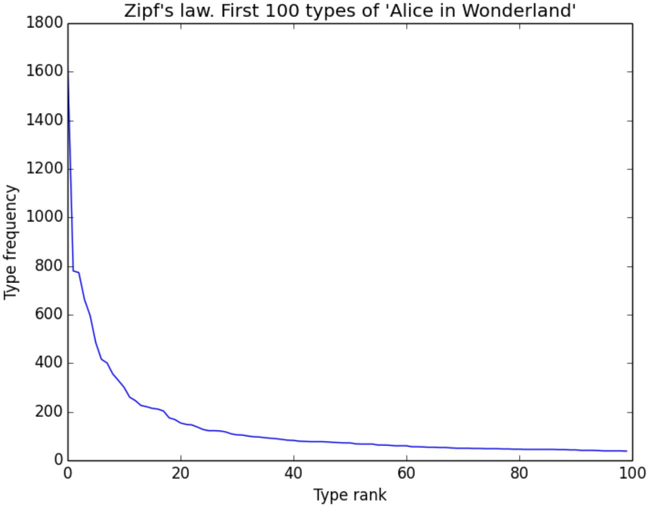
Zipf's law [Gelbukh, Sidorov, 2001])

In any large enough text, the frequency ranks (starting from the highest) of types are inversely proportional to the corresponding frequencies:

$$f = \frac{1}{r}$$

f – frequency of a type
r – rank of a type (its position in the list of all types in order of their frequency of occurrence)

Zipf's law: example



Heaps' law

Heaps' law [Gelbukh, Sidorov, 2001])

The number of different types in a text is roughly proportional to an exponent of its size:

$$|V| = K * N^b$$

N = number of tokens
$|V|$ = size of vocabulary (i.e. number of types)
K, b – free parameters, $K \in [10, 100], b \in [0.4, 0.6]$

Why tokenization is difficult?

- Easy example: “Good muffins cost $3.88 in New York. Please buy me two of them. Thanks.”
  - is “.” a token?
  - is $3.88 a single token?
  - is “New York” a single token?
- Real data may contain noise in it: code, markup, URLs, faulty punctuation
- Real data contains misspellings: “an dthen she aksed”
- Period “.” does not always mean the end of sentence: m.p.h., PhD.

Nevertheless tokenization is important for all other text processing steps. There are rule-based and machine learning-based approaches to development of tokenizers.

Rule-based tokenization

For example, define a token as a sequence of upper and lower case letters: A-Za-z. Reqular expression is a nice tool for programming such rules.

RE in Python

```
In[1]:  import re
In[2]:  prog = re.compile('[A-Za-z]+')
In[3]:  prog.findall("Words, words, words.")
Out[1]:  ['Words', 'words', 'words']
```

Sentence segmentation (1)

What are the sentence boundaries?

- ?, ! are usually unambiguous
- Period "." is an issue
- Direct speech is also an issue: She said, "What time will you be home?" and I said, "I don't know! ". Even worse in Russian!

Let us learn a classifier for sentence segmentation.

Binary classifier

A binary classifier $f : X \implies 0, 1$ takes input data X (a set of sentences) and decides EndOfSentence (0) or NotEndOfSentence (1).

Sentence segmentation (2)

What can be the features for classification? I am a period, am I EndOfSentence?

- Lots of blanks after me?
- Lots of lower case letters and ? or ! after me?
- Do I belong to abbreviation?
- etc.

We need a lot of hand-markup.

Do we need to program this?

No! There is Natural Language Toolkit (NLTK) for everything.

NLTK tokenizers

```
In[1]: from nltk.tokenize import RegexpTokenizer,
wordpunct_tokenize
In[2]: s = 'Good muffins cost $3.88 in New York. Please
buy me two of them. Thanks.'
In[3]: tokenizer = RegexpTokenizer('\w+| \$ [\d \.]+ | S
\+')
In[4]: tokenizer.tokenize(s)
In[5]: wordpunct_tokenize(s)
```

Learning to tokenize

`nltk.tokenize.punkt` is a tool for learning to tokenize from your data. It includes pre-trained Punkt tokenizer for English.

Punkt tokenizer

```
In[1]:  import nltk.data
In[2]:  sent_detector =
nltk.data.load('tokenizers/punkt/english.pickle')
In[3]:  sent_detector.tokenize(s)
```

Exercise 1.1 Word counts

Now it is time for some programming!

Exercise 1.1

Input: Alice in Wonderland (alice.txt) or your text
Output 1: number of tokens
Output 2: number of types

Use `nltk.FreqDist()` to count types. `nltk.FreqDist()` is a frequency dictionary: [key, frequency(key)].

Lemmatization (Normalization)

Each word has a base form:

- has, had, have $\Longrightarrow$ have
- cats, cat, cat's $\Longrightarrow$ cat
- Windows $\Longrightarrow$ window or Windows?

Lemmatization [Jurafsky, Martin, 1999]

Lemmatization (or normalization) is used to reduce inflections or variant forms to base forms. A dictionary with headwords is required.

Lemmatization

```
In[1]:  from nltk.stem import WordNetLemmatizer
In[2]:  lemmatizer = WordNetLemmatizer()
In[3]:  lemmatizer.lemmatize(t)
```

Stemming

A word is built with morphems: *word* = *stem* + *affixes*. Sometimes we do not need affixes.

translate, translation, translator $\Longrightarrow$ translat

Stemming [Jurafsky, Martin, 1999]

Reduce terms to their stems in information retrieval and text classification. Porter's algorithm is the most common English stemmer.

Stemming

```
In[1]:  from nltk.stem.porter import PorterStemmer
In[2]:  stemmer = PorterStemmer()
In[3]:  stemmer.stem(t)
```

Exercise 1.2 Word counts (continued)

Exercise 1.2

Input: Alice in Wonderland (alice.txt) or your text
Output 1: 20 most common lemmata
Output 2: 20 most common stems

Use `FreqDist()` to count lemmata and stems. Use `FreqDist().most_common()` to find most common lemmata and stems.

Exercise 1.3 Do we need all words?

Stopword is a not meaningful word: prepositions, adjunctions, pronouns, articles, etc.

Stopwords

```
In[1]:  from nltk.corpus import stopwords
In[2]:  print stopwords.words('english')
```

Exercise 1.3

Input: Alice in Wonderland (alice.txt) or your text
Output 1: 20 most common lemmata without stop words

Use `not in` operator to exclude not stopwords in a cycle.