Analysis of unstructured data

Lecture 8 - natural language processing (in NLTK) ¶

Janusz Szwabiński

Outlook:

- NLP - what does it mean?
- First steps with NLTK
- Tokenizing text into sentences
- Tokenizing text into words
- Part-Of_Speach tagging
- Stemming and lemmatization
- An introduction into text classification

References:

- Dive into NLTK, http://textminingonline.com/dive-into-nltk-part-i-getting-started-with-nltk (http://textminingonline.com/dive-into-nltk-part-i-getting-started-with-nltk)
- Natural Language Processing with Python, http://www.nltk.org/book/ (http://www.nltk.org/book/)

In [1]:

```
%matplotlib inline
import matplotlib.pyplot as plt
```

NLP - what does it mean?

- *natural language processing*, NLP
- interdisciplinary domain, combines artificial intelligence and machine learning with linguistics
- challenges in natural language processing frequently involve speech recognition, natural language understanding, natural language generation (frequently from formal, machine-readable logical forms), connecting language and machine perception, dialog systems, or some combination thereof
- natural language generation converts information from computer databases or semantic intents into readable human language
- natural language understanding converts chunks of text into more formal representations such as first-order logic structures that are easier for computer programs to manipulate
- natural language understanding involves the identification of the intended semantic from the multiple possible semantics which can be derived from a natural language expression

Is it difficult?

- text tokenization
    - there are no clear word or sentence boundaries in a written text in some languages (e.g. Chinese, Japanese, Thai)
- no clear grammar (exceptions, exceptions to the exceptions):
    - Potato --> potato es, tomato --> tomato es, hero --> hero es, photo --> ???
- homonyms, synonyms
    - fluke --> a fish, fluke --> fins on a whale's tail, fluke --> end parts of an anchor, fluke --> a stroke of luck
    - a river **bank**, a savings **bank**, a **bank** of switches
    - ranny --> zraniony, ranny --> o poranku (context is important)
    - ranny ptaszek
    - to book a flight, to borrow a book
    - buy - purchase
    - samochód - gablota
- inflexion
    - write - written
    - popiół – o popiele
- grammar is often ambiguous
    - a sentence can have more than only one parse tree
        - *Widziałem chłopca jedzącego zupę i bociana.*
        - *Jest szybka w łóżku*
        - *Every man saw the boy with his binoculars*
- invalid data
    - typos
    - syntax errors
    - OCR
- how smart are we?

> FINISHED FILES ARE
> THE RESULT OF YEARS
> OF SCIENTIFIC STUDY
> COMBINED WITH THE
> EXPERIENCE OF YEARS.
>
> THE
> SILLIEST
> MISTAKE IN
> IN THE WORLD

Two different approaches of NLP

- **grammatical**
  - natural language can be described with help of logical forms
  - comparative linguistics - Jakob Grimm, Rasmus Rask
  - I-language and E-language - Noam Chomsky
- **statistical**
  - analysis of real texts may help you to discover the structure of a natural language, in particular typical word usage patterns
  - it is good to look at a large set of texts
  - it is better to look at a huge set of texts
  - it is even better to... --> statistics
  - first attempts - Markov chains (http://www.cs.princeton.edu/courses/archive/spr05/cos126/assignments/markov.html (http://www.cs.princeton.edu/courses/archive/spr05/cos126/assignments/markov.html)), Shannon game

How the statistical method works?

- *They put the money in the bank*
- How should we interpret the word **bank**? River bank? Savings bank?
- We take all available texts and calculate the probability of words' cooccurence:

$$P_1(money, savings)$$

$$P_2(money, river)$$

- we choose the meaning with higher probability

Text corpora

- **text corpus** - a large and structured set of texts (nowadays usually electronically stored and processed), which is usually used to do statistical analysis and hypothesis testing, checking occurrences or validating linguistic rules within a specific language territory
- essential for linguistic research
- often used as the training and test data for machine learning algorithms
- applications:
  - dictionaries
  - foreign language handbooks
  - search engines optimized for specific languages
  - translators

- worth to visit:
  - Narodowy Korpus języka Polskiego, http://nkjp.pl/ (http://nkjp.pl/)
  - British National Corpus, http://www.natcorp.ox.ac.uk/ (http://www.natcorp.ox.ac.uk/)
  - Das Deutsche Referenzkorpus, http://www1.ids-mannheim.de/kl/projekte/korpora/ (http://www1.ids-mannheim.de/kl/projekte/korpora/)
  - Český národní korpus, http://ucnk.ff.cuni.cz/ (http://ucnk.ff.cuni.cz/)
  - Национальный корпус русского языка, http://www.ruscorpora.ru/ (http://www.ruscorpora.ru/)

Getting started with NLTK

After installing NLTK, you need to install NLTK Data which include a lot of corpora, grammars, models and etc. Without NLTK Data, NLTK is nothing special. You can find the complete nltk data list here: http://nltk.org/nltk_data/ (http://nltk.org/nltk_data/)

The simplest way to install NLTK Data is to run the Python interpreter and to type the following commands:

In [2]:

```
import nltk
```

In [3]:

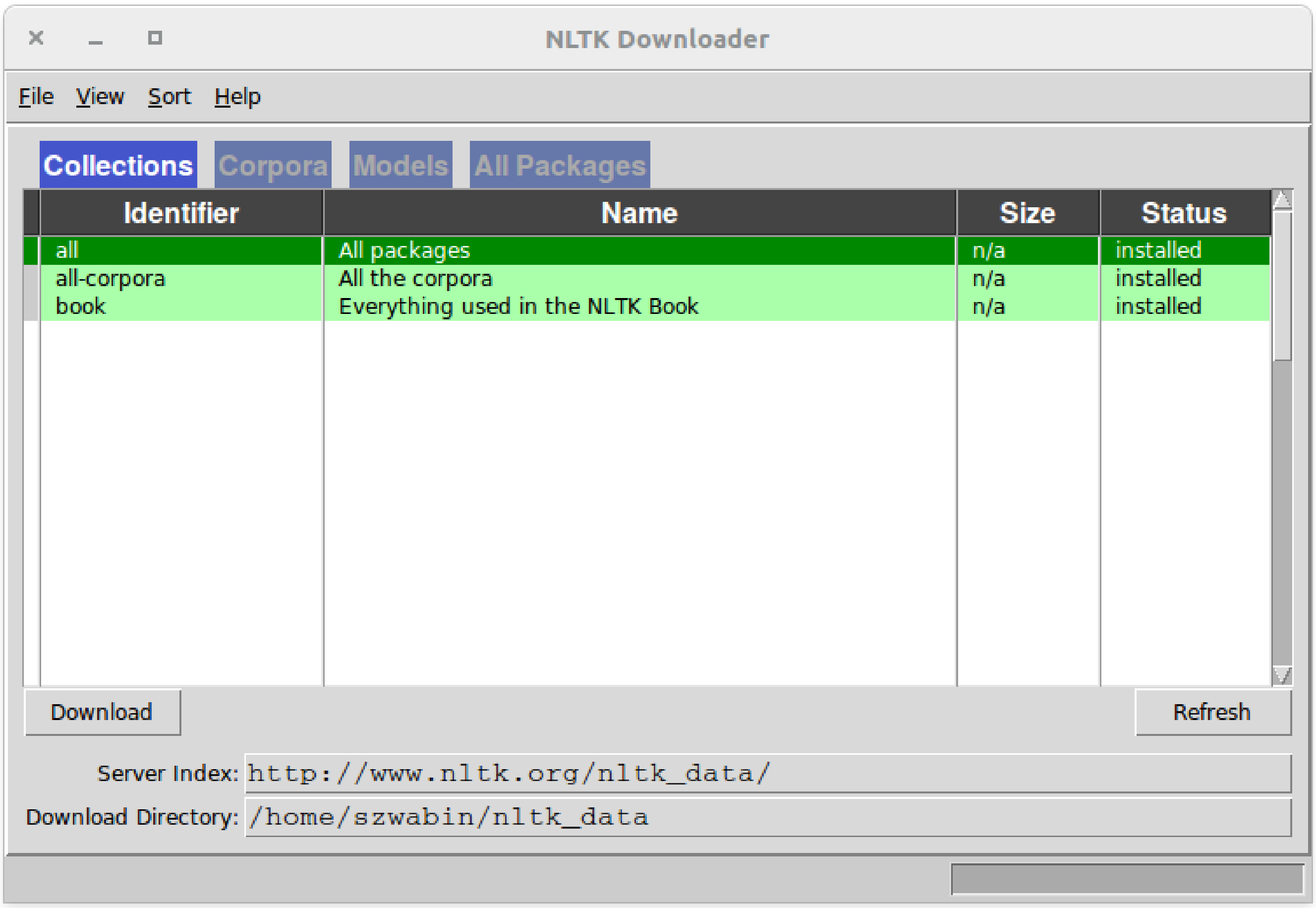
```
nltk.download()
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pag
es/index.xml
```

Out[3]:

```
True
```

After executing the `download()` method, a new window should open, showing the NLTK Downloader:



Let us test the module:

In [4]:

```
from nltk.corpus import brown # Brown University Standard Corpus of Present-Day
 American English
```

In [5]:

```
len(brown.words())
```

Out[5]:

```
1161192
```

In [6]:

```
brown.words()[0:10]
```

Out[6]:

```
['The',
 'Fulton',
 'County',
 'Grand',
 'Jury',
 'said',
 'Friday',
 'an',
 'investigation',
 'of']
```

In [7]:

```
brown.tagged_words()[0:10]
```

Out[7]:

```
[('The', 'AT'),
 ('Fulton', 'NP-TL'),
 ('County', 'NN-TL'),
 ('Grand', 'JJ-TL'),
 ('Jury', 'NN-TL'),
 ('said', 'VBD'),
 ('Friday', 'NR'),
 ('an', 'AT'),
 ('investigation', 'NN'),
 ('of', 'IN')]
```

The meaning of the tags is explained for instance at https://en.wikipedia.org/wiki/Brown_Corpus#Part-of-speech_tags_used (https://en.wikipedia.org/wiki/Brown_Corpus#Part-of-speech_tags_used)

The most important tags are:

- **NN** - singular or mass noun
- **NNS** - plural noun
- **NNP** - possessive singular noun
- **NNSP** - possessive plural noun
- **VB** - verb, base form
- **VBD** - verb, past tense
- **VBP** - verb, non 3rd person, singular, present
- **VBN** - verb, past participle
- **VBZ** - verb, 3rd. singular present
- **JJ** - adjective
- **JJR** - comparative adjective
- **JJS** - semantically superlative adjective (chief, top)
- **RB** - adverb
- **RBR** - comparative adverb
- **RBT** - superlative adverb
- **CD** - cardinal numeral
- **MD** - modal auxiliary (can, should, will)
- **FW** - foreign word
- **PRP** - personal pronoun
- **IN** - preposition
- **CC** - coordinating conjunction

It is also possible to use a simplified set of tags:

In [8]:

```
brown.tagged_words(tagset='universal')[0:10]
```

Out[8]:

```
[('The', 'DET'),
 ('Fulton', 'NOUN'),
 ('County', 'NOUN'),
 ('Grand', 'ADJ'),
 ('Jury', 'NOUN'),
 ('said', 'VERB'),
 ('Friday', 'NOUN'),
 ('an', 'DET'),
 ('investigation', 'NOUN'),
 ('of', 'ADP')]
```

In this case the following universal tags are used:

Tag	Meaning	Examples
ADJ	adjectives	new, good, high, special, big, local
ADP	adpositions	on, of, at, with, by, into, under
ADV	adverbs	really, already, still, early, now
CONJ	conjunctions	and, or, but, if, while, although
DET	determiners	the, a, some, most, every, no, which
NOUN	nouns	year, home, costs, time, Africa
NUM	cardinal numbers	twenty-four, fourth, 1991, 14:24
PRT	particles, other function words	at, on, out, over per, that, up, with
PRON	promouns	he, their, her, its, my, I, us
.	punctuation	. , ; !
X	other (foreign words, typos, etc)	ersatz, esprit, dunno, gr8, univeristy

Tokenizing text into sentences

- sentence boundary disambiguation (SBD)
- also known as sentence breaking
- a problem of deciding where sentences begin and end
- often required by natural language processing tools for a number of reasons
- challenging because punctuation marks are often ambiguous
  - a period may denote an abbreviation, decimal point, an ellipsis, or an email address – not the end of a sentence
  - about 47% of the periods in the Wall Street Journal corpus denote abbreviations
  - question marks and exclamation marks may appear in embedded quotations, emoticons, computer code, and slang
  - languages like Japanese and Chinese have ambiguous sentence-ending markers

In [9]:

```
text = "this's a sent tokenize test. this is sent two. is this sent three? sent
 4 is cool! Now it's your turn."
from nltk.tokenize import sent_tokenize
sent_tokenize_list = sent_tokenize(text)
print(len(sent_tokenize_list))
print(sent_tokenize_list)
```

```
5
["this's a sent tokenize test.", 'this is sent two.', 'is this sent
 three?', 'sent 4 is cool!', "Now it's your turn."]
```

The function `sent_tokenize` uses an instance of the class `PunktSentenceTokenizer` from the module `nltk.tokenize.punkt`. The class was trained for many languages:

In [ ]:

```
# %load /home/szwabin/nltk_data/tokenizers/punkt/README
Pretrained Punkt Models -- Jan Strunk (New version trained after issues 313 and
514 had been corrected)

Most models were prepared using the test corpora from Kiss and Strunk (2006). Ad
ditional models have
been contributed by various people using NLTK for sentence boundary detection.

For information about how to use these models, please confer the tokenization HO
WTO:
http://nltk.googlecode.com/svn/trunk/doc/howto/tokenize.html
and chapter 3.8 of the NLTK book:
http://nltk.googlecode.com/svn/trunk/doc/book/ch03.html#sec-segmentation

There are pretrained tokenizers for the following languages:

File                Language            Source                             Conte
nts                Size of training corpus(in tokens)           Model contribute
d by
================================================================================
================================================================================
=======
czech.pickle        Czech               Multilingual Corpus 1 (ECI)        Lidov
e Noviny                   ~345,000                                Jan Strunk / Tib
or Kiss
                                                                           Liter
arni Noviny
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------
danish.pickle       Danish              Avisdata CD-Rom Ver. 1.1. 1995     Berli
ngske Tidende              ~550,000                                Jan Strunk / Tib
or Kiss
                                        (Berlingske Avisdata, Copenhagen)  Weeke
nd Avisen
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------
dutch.pickle        Dutch               Multilingual Corpus 1 (ECI)        De Li
mburger                    ~340,000                                Jan Strunk / Tib
or Kiss
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------
english.pickle      English             Penn Treebank (LDC)                Wall 
Street Journal             ~469,000                                Jan Strunk / Tib
or Kiss
                    (American)
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------
estonian.pickle     Estonian            University of Tartu, Estonia       Eesti
 Ekspress                  ~359,000                                Jan Strunk / Tib
or Kiss
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------
finnish.pickle      Finnish             Finnish Parole Corpus, Finnish     Books
 and major national        ~364,000                                Jan Strunk / Tib
or Kiss
                                        Text Bank (Suomen Kielen           newsp
```

```
apers
                                        Tekstipankki)
                                        Finnish Center for IT Science
                                        (CSC)
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------
french.pickle       French              Multilingual Corpus 1 (ECI)        Le Mo
nde                        ~370,000                                Jan Strunk / Tib
or Kiss
                    (European)
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------
german.pickle       German              Neue Zürcher Zeitung AG            Neue 
Zürcher Zeitung            ~847,000                                Jan Strunk / Tib
or Kiss
                    (Switzerland)       CD-ROM
                    (Uses "ss"
                     instead of "ß")
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------
greek.pickle        Greek               Efstathios Stamatatos              To Vi
ma (TO BHMA)               ~227,000                                Jan Strunk / Tib
or Kiss
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------
italian.pickle      Italian             Multilingual Corpus 1 (ECI)        La St
ampa, Il Mattino           ~312,000                                Jan Strunk / Tib
or Kiss
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------
norwegian.pickle    Norwegian           Centre for Humanities              Berge
ns Tidende                 ~479,000                                Jan Strunk / Tib
or Kiss
                    (Bokmål and         Information Technologies,
                     Nynorsk)           Bergen
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------
polish.pickle       Polish              Polish National Corpus             Liter
ature, newspapers, etc.  ~1,000,000                                Krzysztof Langne
r
                                        (http://www.nkjp.pl/)
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------
portuguese.pickle   Portuguese          CETENFolha Corpus                  Folha
 de São Paulo              ~321,000                                Jan Strunk / Tib
or Kiss
                    (Brazilian)         (Linguateca)
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------
slovene.pickle      Slovene             TRACTOR                            Delo 
                           ~354,000                                Jan Strunk / Tib
or Kiss
                                        Slovene Academy for Arts
```

```
                                        and Sciences
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------
spanish.pickle      Spanish             Multilingual Corpus 1 (ECI)        Sur
                           ~353,000                                 Jan Strunk / Tib
or Kiss
                    (European)
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------
swedish.pickle      Swedish             Multilingual Corpus 1 (ECI)        Dagen
s Nyheter                  ~339,000                                 Jan Strunk / Tib
or Kiss
                                                                           (and
some other texts)
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------
turkish.pickle      Turkish             METU Turkish Corpus                Milli
yet                        ~333,000                                 Jan Strunk / Tib
or Kiss
                                        (Türkçe Derlem Projesi)
                                        University of Ankara
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------

The corpora contained about 400,000 tokens on average and mostly consisted of ne
wspaper text converted to
Unicode using the codecs module.

Kiss, Tibor and Strunk, Jan (2006): Unsupervised Multilingual Sentence Boundary
Detection.
Computational Linguistics 32: 485-525.

---- Training Code ----

# import punkt
import nltk.tokenize.punkt

# Make a new Tokenizer
tokenizer = nltk.tokenize.punkt.PunktSentenceTokenizer()

# Read in training corpus (one example: Slovene)
import codecs
text = codecs.open("slovene.plain","Ur","iso-8859-2").read()

# Train tokenizer
tokenizer.train(text)

# Dump pickled tokenizer
import pickle
out = open("slovene.pickle","wb")
pickle.dump(tokenizer, out)
out.close()

---------
```

We can specify the module on our own:

In [10]:

```
import nltk.data
tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
tokenizer.tokenize(text)
```

Out[10]:

```
["this's a sent tokenize test.",
 'this is sent two.',
 'is this sent three?',
 'sent 4 is cool!',
 "Now it's your turn."]
```

It works for Polish too:

In [11]:

```
text = "Czy to miało sens? Nie byłem pewien."
tokenizer = nltk.data.load('tokenizers/punkt/polish.pickle')
res = tokenizer.tokenize(text)
for sent in res:
    print(sent)
```

```
Czy to miało sens?
Nie byłem pewien.
```

However, it does not work all the time:

In [12]:

```
text = "Zapytaj o to dr. Kowalskiego."
res = tokenizer.tokenize(text)
for sent in res:
    print(sent)
```

```
Zapytaj o to dr.
Kowalskiego.
```

In [13]:

```
text = u"Nie widzę gdzie leży por. Magda chyba go wyrzuciła."
tokenizer = nltk.data.load('tokenizers/punkt/polish.pickle')
res = tokenizer.tokenize(text)
for i in res:
    print(i)
```

```
Nie widzę gdzie leży por. Magda chyba go wyrzuciła.
```

Tokenizing text into words

In [14]:

```
from nltk.tokenize import word_tokenize
print(word_tokenize('Hello World.'))
print(word_tokenize("this's a test"))
```

```
['Hello', 'World', '.']
['this', "'s", 'a', 'test']
```

The `world_tokenize` function is a wrapper of the `TreebankWordTokenizer`. However, other tokenizers are also available in NLTK:

In [15]:

```
text = "At eight o'clock on Thursday morning Arthur didn't feel very good."
```

In [16]:

```
print(word_tokenize(text))
```

```
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning', 'Arthur', 'd
id', "n't", 'feel', 'very', 'good', '.']
```

In [17]:

```
from nltk.tokenize import WordPunctTokenizer
word_punct_tokenizer = WordPunctTokenizer()
print(word_punct_tokenizer.tokenize(text))
```

```
['At', 'eight', 'o', "'", 'clock', 'on', 'Thursday', 'morning', 'Art
hur', 'didn', "'", 't', 'feel', 'very', 'good', '.']
```

Part-of-speech tagging

From Wikipedia:

> In corpus linguistics, part-of-speech tagging (POS tagging or POST), also called grammatical tagging or word-category disambiguation, is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition, as well as its context—i.e. relationship with adjacent and related words in a phrase, sentence, or paragraph. A simplified form of this is commonly taught to school-age children, in the identification of words as nouns, verbs, adjectives, adverbs, etc.
>
> Once performed by hand, POS tagging is now done in the context of computational linguistics, using algorithms which associate discrete terms, as well as hidden parts of speech, in accordance with a set of descriptive tags. POS-tagging algorithms fall into two distinctive groups: rule-based and stochastic. E. Brill’s tagger, one of the first and most widely used English POS-taggers, employs rule-based algorithms.

Tokenization of the text into words is required in `NLTK` before the tagging:

In [18]:

```
import nltk
text = "Part-of-speech tagging is harder than just having a list of words and th
eir parts of speech"
text = nltk.word_tokenize(text)
nltk.pos_tag(text)
```

Out[18]:

```
[('Part-of-speech', 'JJ'),
 ('tagging', 'NN'),
 ('is', 'VBZ'),
 ('harder', 'JJR'),
 ('than', 'IN'),
 ('just', 'RB'),
 ('having', 'VBG'),
 ('a', 'DT'),
 ('list', 'NN'),
 ('of', 'IN'),
 ('words', 'NNS'),
 ('and', 'CC'),
 ('their', 'PRP$'),
 ('parts', 'NNS'),
 ('of', 'IN'),
 ('speech', 'NN')]
```

It is possible to check the meaning of a tag:

In [19]:

```
nltk.help.upenn_tagset('JJ')
```

```
JJ: adjective or numeral, ordinal
    third ill-mannered pre-war regrettable oiled calamitous first se
parable
    ectoplasmic battery-powered participatory fourth still-to-be-nam
ed
    multilingual multi-disciplinary ...
```

In [20]:

```
nltk.help.upenn_tagset('NN')
```

```
NN: noun, common, singular or mass
    common-carrier cabbage knuckle-duster Casino afghan shed thermos
tat
    investment slide humour falloff slick wind hyena override subhum
anity
    machinist ...
```

Languages other than English

The default tagger in NLTK was trained on the PENN Treebank corpus of English (https://www.cis.upenn.edu/~treebank/ (https://www.cis.upenn.edu/~treebank/)). However, NLTK includes corpora of other languages that may be used to train the taggers for languages different than English. Let us have a look at the Polish corpus:

In [21]:

```
# Find the directory where the corpus lives
import nltk
pl196x_dir = nltk.data.find('corpora/pl196x')
```

In [22]:

```
print(pl196x_dir)
```

```
/home/szwabin/nltk_data/corpora/pl196x
```

In [23]:

```
#Create a new corpus reader object
from nltk.corpus.reader import pl196x
pl = pl196x.Pl196xCorpusReader(pl196x_dir,r'.*\.xml',textids='textids.txt',cat_f
ile="cats.txt")
```

In [24]:

```
#Use the new corpus object
print(pl.fileids())
```

```
['a-publi.xml', 'b-prasa.xml', 'c-popul.xml', 'd-proza.xml', 'e-dram
at.xml', 'flib.xml', 'fslib.xml', 'iso88592.xml', 'morf.xml', 'pl196
x.xml']
```

In [25]:

```
#Look at tagged words
twords = pl.tagged_words(fileids=pl.fileids(),categories='cats.txt')
for w in twords[:10]:
    print(w)
```

```
('Sztuka', 'SSNF---------P')
('utraciła', 'VS-F-3POD----P')
('swoją', 'ASAFP--------P')
('moc', 'SSAF---------P')
('pobudzającą', 'ASAXX--------P')
(':', 'dwukropek')
('przykrym', 'ASIPP--------P ASIAP--------P ASIIP--------P ASINP----
----P ASLPP--------P ASLAP--------P ASLIP--------P ASLNP--------P AP
DOP--------P APDRP--------P')
('widowiskiem', 'SSIN---------P')
('staje', 'VS---3TONZ---P')
('się', 'Z-N----------P Z-G-------N--P Z-A-------N--P')
```

Important note If you want to use the `pl196X` corpus with Python 3.X, then you have to edit the reader file (`/usr/local/lib/python3.5/dist-packages/nltk/corpus/reader/pl196x.py` on Ubuntu) and replace the following line in the `_resolve` method

```
if len(filter(lambda accessor: accessor is None,
                      (fileids, categories, textids))) != 1:
```

by

```
if len(list(filter(lambda accessor: accessor is None,
                      (fileids, categories, textids)))) != 1:
```

In `pl196X` corpus every tag consists of 14 characters. Their meaning may be checked at: [http://clip.ipipan.waw.pl/PL196x?action=AttachFile&do=view&target=taksonomia.pdf (http://clip.ipipan.waw.pl/PL196x?action=AttachFile&do=view&target=taksonomia.pdf)](http://clip.ipipan.waw.pl/PL196x?action=AttachFile&do=view&target=taksonomia.pdf)

Now, we can use the corpus to train the `UnigramTagger`:

In [26]:

```
tsents = pl.tagged_sents(fileids=pl.fileids(),categories='cats.txt')[:3000]
tagger = nltk.UnigramTagger(tsents)
```

Let us check, how it works:

In [27]:

```
tekst = "To jest przykładowe zdanie w języku polskim"
tagger.tag(tekst.split())
```

Out[27]:

```
[('To', 'Z-N-------XX-P'),
 ('jest', 'VS---3TON----P'),
 ('przykładowe', None),
 ('zdanie', 'SSAN---------P'),
 ('w', 'P-L----------P'),
 ('języku', 'SSGI---------P SSLI---------P SSVI---------P'),
 ('polskim', 'ASIPP--------P ASIAP--------P ASIIP--------P ASINP----
----P')]
```

We can also check the accuracy of the tagger:

In [28]:

```
test_sents = pl.tagged_sents(fileids=pl.fileids(),categories='cats.txt')[3000:60
00]
tagger.evaluate(test_sents)
```

Out[28]:

```
0.6597164434535492
```

- accuracy is 65%
- it may be improved by taking a larger training set or by taking only a set of most frequent tags in the corpus

Other taggers

The `UnigramTagger` is not the only tagger contained in `NLTK`. Among other taggers we have:

- `DefaultTagger` - the simplest possible tagger assigns the same tag to each token. It establishes an important baseline for tagger performance (it allows to tag each word with the most likely tag). Useful backoff in case a more advanced tagger fails to tag a word.
- `RegexpTagger` - assigns tags to tokens on the basis of matching patterns.

```
patterns =
    [(r'^-?[0-9]+(.[0-9]+)?$', 'CD'),  # cardinal numbers
     (r'.*able$', 'JJ'),               # adjectives
     (r'.*ness$', 'NN'),               # nouns formed from adjectives
     (r'.*ly$', 'RB'),                 # adverbs
     (r'.*ing$', 'VBG'),               # gerunds
     (r'.*ed$', 'VBD'),                # past tense verbs
     (r'^[A-Z].*s$', 'NNPS'),          # plural proper nouns
     (r'.*s$', 'NNS'),                 # plural nouns
     (r'^[A-Z].*$', 'NNP'),            # singular proper nouns
     (r'.*', 'NN')]                    # singular nouns (default)

tagger = nltk.RegexpTagger(patterns)
print(tagger.tag("..."))
```

- `NgramTagger` - a generalization of a unigram tagger whose context is the current word together with the part-of-speech tags of the $n-1$ preceding tokens
- `BigramTagger` - a special case of the NgramTagger for $n=2$
- `TrigramTagger` - a special case of the NgramTagger for $n=3$
- `AffixTagger` - a tagger that chooses a token's tag based on a leading or trailing substring of its word string
- `BrillTagger` - it uses an initial tagger (such as `DefaultTagger`) to assign an initial tag sequence to a text and then applies an ordered list of transformational rules to correct the tags of individual tokens

In [29]:

```
tagger = nltk.UnigramTagger(tsents, backoff=nltk.DefaultTagger('NN'))
```

In [30]:

```
tekst = "To jest przykładowe zdanie w języku polskim"
tagger.tag(tekst.split())
```

Out[30]:

```
[('To', 'Z-N-------XX-P'),
 ('jest', 'VS---3TON----P'),
 ('przykładowe', 'NN'),
 ('zdanie', 'SSAN---------P'),
 ('w', 'P-L----------P'),
 ('języku', 'SSGI---------P SSLI---------P SSVI---------P'),
 ('polskim', 'ASIPP--------P ASIAP--------P ASIIP--------P ASINP----
----P')]
```

The above example illustrates the possibility of linking taggers with each other. A more complex configuration could be:

```
BigramTagger --> UnigramTagger --> RegexpTagger
```

The are some benefits of such an approach:

- improved accuracy of the resulting tagger
- data reduction - for instance, we do not have to tag nouns in the corpus. Instead, we simply assign the 'NN' tag to all unknown words with the `DefaultTagger`

TreeTagger

- homepage: [http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/ (http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/)](http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/)
- Python module available: `treetagggerwrapper`
- support for German, English, French, Italian, Dutch, Spanish, Bulgarian, Russian, Portuguese, Galician, Chinese, Swahili, Slovak, Slovenian, Latin, Estonian, Polish, Romanian
- adding new languages possible, given a dictionary and a manually tagged corpus

In [31]:

```
!echo "On czyta książkę." | /home/szwabin/Tools/TreeTagger/cmd/tree-tagger-polis
h
```

```
        reading parameters ...
        tagging ...
On      ppron3:sg:nom:m1:ter:akc:npraep on
czyta   fin:sg:ter:imperf       czytać
książkę subst:sg:acc:f  książka
.       SENT    .
         finished.
```

In [32]:

```
!echo "To jest przykładowe zdanie w języku polskim." | /home/szwabin/Tools/TreeT
agger/cmd/tree-tagger-polish
```

```
        reading parameters ...
        tagging ...
To      subst:sg:nom:n  to
jest    fin:sg:ter:imperf       być
przykładowe     adj:pl:acc:n:pos        przykładowy
zdanie  subst:sg:acc:n  zdanie
w       prep:loc:nwok   w
języku  subst:sg:loc:m3 język
polskim adj:sg:loc:m3:pos       polski
.       SENT    .
         finished.
```

Stemming and lemmatization

Stemming

From Wikipedia:

> In linguistic morphology and information retrieval, **stemming** is the process for reducing inflected (or sometimes derived) words to their stem, base or root form—generally a written word form. The stem need not be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root. Algorithms for stemming have been studied in computer science since the 1960s. Many search engines treat words with the same stem as synonyms as a kind of query expansion, a process called conflation.

`NLTK` provides several famous stemmers interfaces, such as Porter stemmer, Lancaster Stemmer, Snowball Stemmer etc. Using those stemmers is very simple.

Porter stemmer

A very good explanation of the Porter algorithm may be found at:
http://snowballstem.org/algorithms/porter/stemmer.html
(http://snowballstem.org/algorithms/porter/stemmer.html)

In [33]:

```
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()
```

In [34]:

```
words = ['caresses', 'flies', 'dies', 'mules', 'denied',
           'died', 'agreed', 'owned', 'humbled', 'sized',
           'meeting', 'stating', 'siezing', 'itemization',
           'sensational', 'traditional', 'reference', 'colonizer',
           'plotted']
for w in words:
    print(w, stemmer.stem(w))
```

```
caresses caress
flies fli
dies die
mules mule
denied deni
died die
agreed agre
owned own
humbled humbl
sized size
meeting meet
stating state
siezing siez
itemization item
sensational sensat
traditional tradit
reference refer
colonizer colon
plotted plot
```

Snowball stemmer

Snowball stemmer represents actually a family of stemmers based on the Snowball language created by Martin Porter.

Invoking it is easy:

In [35]:

```
from nltk.stem.snowball import SnowballStemmer
stemmer = SnowballStemmer("english")
for w in words:
    print(w, stemmer.stem(w))
```

```
caresses caress
flies fli
dies die
mules mule
denied deni
died die
agreed agre
owned own
humbled humbl
sized size
meeting meet
stating state
siezing siez
itemization item
sensational sensat
traditional tradit
reference refer
colonizer colon
plotted plot
```

We can tell the stemmer to omit stop words:

In [36]:

```
stemmer2 = SnowballStemmer("english", ignore_stopwords=True)
```

In [37]:

```
print(stemmer.stem("having"))
```

```
have
```

In [38]:

```
print(stemmer2.stem("having"))
```

```
having
```

The 'english' stemmer seems to be better than the original 'porter' stemmer:

In [39]:

```
print(SnowballStemmer("english").stem("generously"))
```

```
generous
```

In [40]:

```
print(SnowballStemmer("porter").stem("generously"))
```

```
gener
```

Let us see which languages are supported:

In [41]:

```
print(" ".join(SnowballStemmer.languages))
```

```
arabic danish dutch english finnish french german hungarian italian
 norwegian porter portuguese romanian russian spanish swedish
```

Unfortunately, Polish is not included.

Polish language and stemming

`pl_stemmer.py` program is a very simple python stemmer for Polish language based on Porter's Algorithm (https://github.com/Tutanchamon/pl_stemmer/blob/master/pl_stemmer.py (https://github.com/Tutanchamon/pl_stemmer/blob/master/pl_stemmer.py)).

In [42]:

```
! cat email.txt
```

```
Kariera na językach to wydarzenie zorganizowane z myślą o studentach
i absolwentach znających języki obce na poziomie co najmniej dobrym
Będą oni mieli okazję zastanowić się nad kierunkami
rozwoju własnej kariery zawodowej w oparciu o informacje
na temat możliwości wykorzystania swoich
umiejętności lingwistycznych na współczesnym rynku pracy
```

In [43]:

```
!cat email.txt | python pl_stemmer.py -f
```

```
(<Values at 0x7fab558e31b8: {'debug': True, 'expected_stem_locatio
n': None, 'black_list_file_location': None}>, [])
Debug: True
Blacklist: None
ExpectedFile: None
kariera|karier
na|na
językach|język
to|to
wydarzenie|wydarz
zorganizowane|zorganizowane
z|z
myślą|myśl
o|o
studentach|studen

i|i
absolwentach|absolwen
znających|znaj
języki|język
obce|obce
na|na
poziomie|poziom
co|co
najmniej|najmn
dobrym|dobrym

będą|będą
oni|oni
mieli|miel
okazję|okazj
zastanowić|zastanow
się|się
nad|nad
kierunkami|kierunk

rozwoju|rozwoj
własnej|własn
kariery|karier
zawodowej|zawodow
w|w
oparciu|opar
o|o
informacje|informacje

na|na
temat|temat
możliwości|możliwośc
wykorzystania|wykorzyst
swoich|swoich

umiejętności|umiejętnośc
lingwistycznych|lingwistyczn
na|na
współczesnym|współczesnym
rynku|rynk
pracy|prac
```

Lemmatization

From Wikipedia:

> **Lemmatisation** (or lemmatization) in linguistics, is the process of grouping together the different inflected forms of a word so they can be analysed as a single item.
>
> In computational linguistics, lemmatisation is the algorithmic process of determining the lemma for a given word. Since the process may involve complex tasks such as understanding context and determining the part of speech of a word in a sentence (requiring, for example, knowledge of the grammar of a language) it can be a hard task to implement a lemmatiser for a new language.
>
> Lemmatisation is closely related to stemming. The difference is that a stemmer operates on a single word without knowledge of the context, and therefore cannot discriminate between words which have different meanings depending on part of speech. However, stemmers are typically easier to implement and run faster, and the reduced accuracy may not matter for some applications.

The NLTK Lemmatization method is based on WordNet's built-in morphy function.

From WordNet official website (https://wordnet.princeton.edu/ (https://wordnet.princeton.edu/)):

> WordNet® is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. The resulting network of meaningfully related words and concepts can be navigated with the browser. WordNet is also freely and publicly available for download. WordNet's structure makes it a useful tool for computational linguistics and natural language processing.
>
> WordNet superficially resembles a thesaurus, in that it groups words together based on their meanings. However, there are some important distinctions. First, WordNet interlinks not just word forms—strings of letters—but specific senses of words. As a result, words that are found in close proximity to one another in the network are semantically disambiguated. Second, WordNet labels the semantic relations among words, whereas the groupings of words in a thesaurus does not follow any explicit pattern other than meaning similarity.

In [44]:

```
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
print(wordnet_lemmatizer.lemmatize('better', pos='a'))
print(wordnet_lemmatizer.lemmatize('meeting',pos='v'))
```

```
good
meet
```

In [45]:

```
import nltk
text = "Part-of-speech tagging is harder than just having a list of words and th
eir parts of speech"
text = nltk.word_tokenize(text)
for w in text:
    print(w, " | ", wordnet_lemmatizer.lemmatize(w))
```

```
Part-of-speech  |  Part-of-speech
tagging  |  tagging
is  |  is
harder  |  harder
than  |  than
just  |  just
having  |  having
a  |  a
list  |  list
of  |  of
words  |  word
and  |  and
their  |  their
parts  |  part
of  |  of
speech  |  speech
```

In the default setting, the lemmatizer assumes that every word is a noun, i.e. for each word it searches for the closest noun. We can change it by the `pos` flag:

In [46]:

```
for w in text:
    print(w, " | ", wordnet_lemmatizer.lemmatize(w,pos='v'))
```

```
Part-of-speech  |  Part-of-speech
tagging  |  tag
is  |  be
harder  |  harder
than  |  than
just  |  just
having  |  have
a  |  a
list  |  list
of  |  of
words  |  word
and  |  and
their  |  their
parts  |  part
of  |  of
speech  |  speech
```

In [47]:

```
print(wordnet_lemmatizer.lemmatize('better', pos='a'))
print(wordnet_lemmatizer.lemmatize('meeting',pos='v'))
```

```
good
meet
```

In [48]:

```
print(wordnet_lemmatizer.lemmatize('better'))
print(wordnet_lemmatizer.lemmatize('meeting'))
```

```
better
meeting
```

Thus, we have to use POS Tagging before word lemmatization. Moreover, since the POS tagger uses the Treebank tag set (https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html (https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html)), we have to translate it into WordNet compatible tags:

In [49]:

```
from nltk.corpus import wordnet

def get_wordnet_pos(treebank_tag):

    if treebank_tag.startswith('J'):
        return wordnet.ADJ
    elif treebank_tag.startswith('V'):
        return wordnet.VERB
    elif treebank_tag.startswith('N'):
        return wordnet.NOUN
    elif treebank_tag.startswith('R'):
        return wordnet.ADV
    else:
        return ''
```

In [50]:

```
import nltk
from nltk.stem.wordnet import WordNetLemmatizer

tokens = ['better','meeting','churches','abaci','are','is']
tagged = nltk.pos_tag(tokens)
print(tagged)

results = []
lemmatizer = WordNetLemmatizer()
for word, tag in tagged:
    wntag = get_wordnet_pos(tag)
    if wntag is None:# not supply tag in case of None
        lemma = lemmatizer.lemmatize(word)
    else:
        lemma = lemmatizer.lemmatize(word, pos=wntag)
    results.append(lemma)
print(results)
```

```
[('better', 'RBR'), ('meeting', 'NN'), ('churches', 'NNS'), ('abac
i', 'NN'), ('are', 'VBP'), ('is', 'VBZ')]
['well', 'meeting', 'church', 'abacus', 'be', 'be']
```

Lemmatization in Polish

- Morfeusz, http://sgjp.pl/morfeusz/morfeusz.html (http://sgjp.pl/morfeusz/morfeusz.html) (Python API)
- Lametyzator, http://www.cs.put.poznan.pl/dweiss/xml/projects/lametyzator/index.xml (http://www.cs.put.poznan.pl/dweiss/xml/projects/lametyzator/index.xml) (currently a part of the Morfologik project, https://github.com/morfologik/ (https://github.com/morfologik/))

In [51]:

```
%%python2
# coding: utf-8
import morfeusz2
lem = morfeusz2.Morfeusz()
print lem

text = u'Mam próbkę analizy morfologicznej'
for i in lem.analyse(text):
    print len(i),i
```

```
<morfeusz2.Morfeusz object at 0x7f7b622d6450>
3 (0, 1, (u'Mam', u'mami\u0107', 'impt:sg:sec:imperf', [], []))
3 (0, 1, (u'Mam', u'mie\u0107', 'fin:sg:pri:imperf', [], []))
3 (0, 1, (u'Mam', u'mama', 'subst:pl:gen:f', [u'nazwa pospolita'],
 []))
3 (1, 2, (u'pr\xf3bk\u0119', u'pr\xf3bka', 'subst:sg:acc:f', [u'nazw
a pospolita'], []))
3 (2, 3, (u'analizy', u'analiza', 'subst:sg:gen:f', [u'nazwa pospoli
ta'], []))
3 (2, 3, (u'analizy', u'analiza', 'subst:pl:nom.acc.voc:f', [u'nazwa
pospolita'], []))
3 (3, 4, (u'morfologicznej', u'morfologiczny', 'adj:sg:dat:f:pos',
 [], []))
3 (3, 4, (u'morfologicznej', u'morfologiczny', 'adj:sg:gen:f:pos',
 [], []))
3 (3, 4, (u'morfologicznej', u'morfologiczny', 'adj:sg:loc:f:pos',
 [], []))
```

An introduction into text classification

From Wikipedia:

> Document classification or document categorization is a problem in library science, information science and computer science. The task is to assign a document to one or more classes or categories. This may be done “manually” (or “intellectually”) or algorithmically. The intellectual classification of documents has mostly been the province of library science, while the algorithmic classification of documents is used mainly in information science and computer science. The problems are overlapping, however, and there is therefore also interdisciplinary research on document classification.

Text classification is a very important technique of text analysis. It has many potential applications:

- spam filtering
- sentiment analysis
- language identification
- genre identification

Classification in NLTK

- requirements - labeled category data, which can be used as a training set
- example: NLTK Name Corpus
- goal: a Gender Identification classifier

In [52]:

```
from nltk.corpus import names
import random
names = ([(name, 'male') for name in names.words('male.txt')] + [(name,
'female') for name in names.words('female.txt')])
```

In [53]:

```
random.shuffle(names)
```

In [54]:

```
len(names)
```

Out[54]:

```
7944
```

In [55]:

```
names[:10]
```

Out[55]:

```
[('Tracey', 'female'),
 ('Tucker', 'male'),
 ('Dix', 'female'),
 ('Berti', 'female'),
 ('Breanne', 'female'),
 ('Joela', 'female'),
 ('Flossie', 'female'),
 ('Dotti', 'female'),
 ('Chicky', 'female'),
 ('Rea', 'female')]
```

- **feature** - the most important thing for a text classifier
  - can be very flexible
  - defined by a human engineer
- in our example - the final letter of a given name

In [56]:

```
def gender_features(word):
    return {'last_letter': word[-1]}

gender_features('Gary')
```

Out[56]:

```
{'last_letter': 'y'}
```

- dictionary returned by the helper function is called a **feature set**
- it maps from features' names to their values
- it is a core part for NLTK Classifier

In [57]:

```
featuresets = [(gender_features(n), g) for (n, g) in names]
```

In [58]:

```
featuresets[0:10]
```

Out[58]:

```
[({'last_letter': 'y'}, 'female'),
 ({'last_letter': 'r'}, 'male'),
 ({'last_letter': 'x'}, 'female'),
 ({'last_letter': 'i'}, 'female'),
 ({'last_letter': 'e'}, 'female'),
 ({'last_letter': 'a'}, 'female'),
 ({'last_letter': 'e'}, 'female'),
 ({'last_letter': 'i'}, 'female'),
 ({'last_letter': 'y'}, 'female'),
 ({'last_letter': 'a'}, 'female')]
```

We have to segment the feature set into the training set and the test one:

In [59]:

```
train_set, test_set = featuresets[500:], featuresets[:500]
```

In [60]:

```
len(train_set)
```

Out[60]:

7444

In [61]:

```
len(test_set)
```

Out[61]:

500

First, we use the `NaiveBayes` classifier (see https://en.wikipedia.org/wiki/Naive_Bayes_classifier (https://en.wikipedia.org/wiki/Naive_Bayes_classifier) for explanation):

In [62]:

```
from nltk import NaiveBayesClassifier
nb_classifier = NaiveBayesClassifier.train(train_set)
```

In [63]:

```
print(nb_classifier.classify(gender_features('Gary')))
print(nb_classifier.classify(gender_features('Grace')))
```

```
female
female
```

We use the test set to check the accuracy of the classifier:

In [64]:

```
from nltk import classify
classify.accuracy(nb_classifier, test_set)
```

Out[64]:

0.808

In [65]:

```
nb_classifier.show_most_informative_features(5)
```

```
Most Informative Features
             last_letter = 'a'            female : male   =     33.0
: 1.0
             last_letter = 'k'              male : female =     31.9
: 1.0
             last_letter = 'f'              male : female =     16.7
: 1.0
             last_letter = 'p'              male : female =     11.9
: 1.0
             last_letter = 'v'              male : female =     10.6
: 1.0
```

Here is how to train a Maximum Entropy Classifier (https://en.wikipedia.org/wiki/Multinomial_logistic_regression (https://en.wikipedia.org/wiki/Multinomial_logistic_regression)) for Gender Identification:

In [66]:

```
from nltk import MaxentClassifier
me_classifier = MaxentClassifier.train(train_set)
```

```
==> Training (100 iterations)

      Iteration    Log Likelihood    Accuracy
      ---------------------------------------
             1          -0.69315        0.369
             2          -0.37759        0.760
             3          -0.37718        0.760
             4          -0.37694        0.760
             5          -0.37677        0.760
             6          -0.37666        0.760
             7          -0.37657        0.760
             8          -0.37650        0.760
             9          -0.37645        0.760
            10          -0.37640        0.760
            11          -0.37637        0.760
            12          -0.37633        0.760
            13          -0.37631        0.760
            14          -0.37628        0.760
            15          -0.37626        0.760
            16          -0.37625        0.760
            17          -0.37623        0.760
            18          -0.37622        0.760
            19          -0.37620        0.760
            20          -0.37619        0.760
            21          -0.37618        0.760
            22          -0.37617        0.760
            23          -0.37616        0.760
            24          -0.37615        0.760
            25          -0.37615        0.760
            26          -0.37614        0.760
            27          -0.37613        0.760
            28          -0.37613        0.760
            29          -0.37612        0.760
            30          -0.37612        0.760
            31          -0.37611        0.760
            32          -0.37611        0.760
            33          -0.37610        0.760
            34          -0.37610        0.760
            35          -0.37609        0.760
            36          -0.37609        0.760
            37          -0.37609        0.760
            38          -0.37608        0.760
            39          -0.37608        0.760
            40          -0.37608        0.760
            41          -0.37607        0.760
            42          -0.37607        0.760
            43          -0.37607        0.760
            44          -0.37607        0.760
            45          -0.37606        0.760
            46          -0.37606        0.760
            47          -0.37606        0.760
            48          -0.37606        0.760
            49          -0.37606        0.760
            50          -0.37605        0.760
            51          -0.37605        0.760
            52          -0.37605        0.760
            53          -0.37605        0.760
            54          -0.37605        0.760
            55          -0.37605        0.760
            56          -0.37604        0.760
            57          -0.37604        0.760
```

```
             58          -0.37604        0.760
             59          -0.37604        0.760
             60          -0.37604        0.760
             61          -0.37604        0.760
             62          -0.37604        0.760
             63          -0.37603        0.760
             64          -0.37603        0.760
             65          -0.37603        0.760
             66          -0.37603        0.760
             67          -0.37603        0.760
             68          -0.37603        0.760
             69          -0.37603        0.760
             70          -0.37603        0.760
             71          -0.37603        0.760
             72          -0.37603        0.760
             73          -0.37602        0.760
             74          -0.37602        0.760
             75          -0.37602        0.760
             76          -0.37602        0.760
             77          -0.37602        0.760
             78          -0.37602        0.760
             79          -0.37602        0.760
             80          -0.37602        0.760
             81          -0.37602        0.760
             82          -0.37602        0.760
             83          -0.37602        0.760
             84          -0.37602        0.760
             85          -0.37602        0.760
             86          -0.37601        0.760
             87          -0.37601        0.760
             88          -0.37601        0.760
             89          -0.37601        0.760
             90          -0.37601        0.760
             91          -0.37601        0.760
             92          -0.37601        0.760
             93          -0.37601        0.760
             94          -0.37601        0.760
             95          -0.37601        0.760
             96          -0.37601        0.760
             97          -0.37601        0.760
             98          -0.37601        0.760
             99          -0.37601        0.760
          Final          -0.37601        0.760
```

In [67]:

```
print(me_classifier.classify(gender_features('Gary')))
print(me_classifier.classify(gender_features('Grace')))
```

```
female
female
```

In [68]:

```
classify.accuracy(me_classifier, test_set)
```

Out[68]:

```
0.808
```

In [69]:

```
me_classifier.show_most_informative_features(5)
```

```
   6.644 last_letter==' ' and label is 'female'
   6.644 last_letter=='c' and label is 'male'
  -4.864 last_letter=='a' and label is 'male'
  -3.503 last_letter=='k' and label is 'female'
  -2.700 last_letter=='f' and label is 'female'
```

It seems that Naive Bayes and Maxent model have the same result on this gender task. However, let us look what happens if we define a more complex feature extractor function and train the models again:

In [70]:

```
def gender_features2(name):
    features = {}
    features["firstletter"] = name[0].lower()
    features["lastletter"] = name[-1].lower()
    for letter in 'abcdefghijklmnopqrstuvwxyz':
        features["count(%s)" % letter] = name.lower().count(letter)
        features["has(%s)" % letter] = (letter in name.lower())
    return features
```

In [71]:

```
gender_features2('Gary')
```

Out[71]:

```
{'count(a)': 1,
 'count(b)': 0,
 'count(c)': 0,
 'count(d)': 0,
 'count(e)': 0,
 'count(f)': 0,
 'count(g)': 1,
 'count(h)': 0,
 'count(i)': 0,
 'count(j)': 0,
 'count(k)': 0,
 'count(l)': 0,
 'count(m)': 0,
 'count(n)': 0,
 'count(o)': 0,
 'count(p)': 0,
 'count(q)': 0,
 'count(r)': 1,
 'count(s)': 0,
 'count(t)': 0,
 'count(u)': 0,
 'count(v)': 0,
 'count(w)': 0,
 'count(x)': 0,
 'count(y)': 1,
 'count(z)': 0,
 'firstletter': 'g',
 'has(a)': True,
 'has(b)': False,
 'has(c)': False,
 'has(d)': False,
 'has(e)': False,
 'has(f)': False,
 'has(g)': True,
 'has(h)': False,
 'has(i)': False,
 'has(j)': False,
 'has(k)': False,
 'has(l)': False,
 'has(m)': False,
 'has(n)': False,
 'has(o)': False,
 'has(p)': False,
 'has(q)': False,
 'has(r)': True,
 'has(s)': False,
 'has(t)': False,
 'has(u)': False,
 'has(v)': False,
 'has(w)': False,
 'has(x)': False,
 'has(y)': True,
 'has(z)': False,
 'lastletter': 'y'}
```

In [72]:

```
featuresets = [(gender_features2(n), g) for (n, g) in names]
train_set, test_set = featuresets[500:], featuresets[:500]
```

In [73]:

```
nb2_classifier = NaiveBayesClassifier.train(train_set)
```

In [74]:

```
classify.accuracy(nb2_classifier, test_set)
```

Out[74]:

0.822

In [75]:

```
me2_classifier = MaxentClassifier.train(train_set)
```