Natural language processing in Python using NLTK

Iulia Cioroianu - Ph.D. Student, New York University

April 23, 2013

Before we start

- Download the code file: `http://goo.gl/15N19`
- Run the following code in Python:

```
>>> import nltk
>>> nltk.download()
```

- Wait a few seconds, it will open a downloader. From "Collections", download "book".
- This will import the data needed for the examples.

Natural language processing

- NLP
  - broad sense: any kind of computer manipulation of natural language
  - from word frequencies to "understanding" meaning
- Applications
  - text processing
  - information extraction
  - document classification and sentiment analysis
  - document similarity
  - automatic summarizing
  - discourse analysis

What is NLTK?

- Suite of open source Python libraries and programs for NLP.
  - Python: open source programming language
- Developed for educational purposes by Steven Bird, Ewan Klein and Edward Loper.
- Very good online documentation.
- Other options out there?
  - R cran.r-project.org/web/views/NaturalLanguageProcessing.html - many packages, should do many of the same things as NLTK.
  - OpenNLP - Java, R - similar to NLTK
  - LingPipe - Java
  - Many commercial applications that do specific tasks for business clients: SAS Text Analytics, various SPSS tools.
- NLTK most widely used

Downloads and installation

- Installation instructions at: http://nltk.org/install.html
- You need Python 2.7.
- Also download corpora, packages and the data used for examples in the book.
- From Python:

```
>>> import nltk
>>> nltk.download()
```

- Opens the NLTK downloader, you can choose what to download.

Resources used

- Presentation based almost entirely on the NLTK manual:
- Natural Language Processing with Python- Analyzing Text with the Natural Language Toolkit
  - Steven Bird, Ewan Klein and Edward Loper
  - free online
- Also useful:
- Python Text Processing with NLTK 2.0 Cookbook
  - Jacob Perkins

Strings

- Lowest level of text processing

```
>>> monty = "Monty Python's "\
...  "Flying Circus."
>>> monty*2 + "plus just last word:" + monty[-7:]
"Monty Python's Flying Circus.Monty Python's Flying Circus.plus
just last word:Circus."
>>> monty.find('Python') #finds position of substring within string
6
>>> monty.upper() +' and '+ monty.lower()
"MONTY PYTHON'S FLYING CIRCUS. and monty python's flying circus."
>>> monty.replace('y', 'x')
"Montx Pxthon's Flxing Circus."
```

Texts as lists

- As opposed to strings, lists are flexible about the elements they contain.

```
>>> sent1 = ['Monty', 'Python']
>>> sent2 = ['and', 'the', 'Holy', 'Grail']
>>> len(sent2)
4
>>> sent1[1]
'Python'
>>> sent2.append("1975")
>>> sent1 + sent2
['Monty', 'Python', 'and', 'the', 'Holy', 'Grail', '1975']
>>> sorted(sent1 + sent2)
['1975', 'Grail', 'Holy', 'Monty', 'Python', 'and', 'the']
>>> ' '.join(['Monty', 'Python'])
'Monty Python'
>>> 'Monty Python'.split()
['Monty', 'Python']
```

Search, context

- Using Inaugural Address text.
- Search and display word in context

```
>>> from nltk.book import text4
>>> text4.concordance("vote")
Displaying 3 of 8 matches:
determined by a majority of a single vote , and that can be
procured by a part
e is applied it may be overcome by a vote of two - thirds of both
Houses of Co
ess and the canvass of the electoral vote .  Our people have
already worthily o
```

Context, collocations

- Find words that appear in similar contexts

```
>>> text4.similar("vote")
nation abandon achieve adopt all approach assemble balance band
basis
beacon beginning board body breath campaign career
```

- Collocations - words that appear together frequently

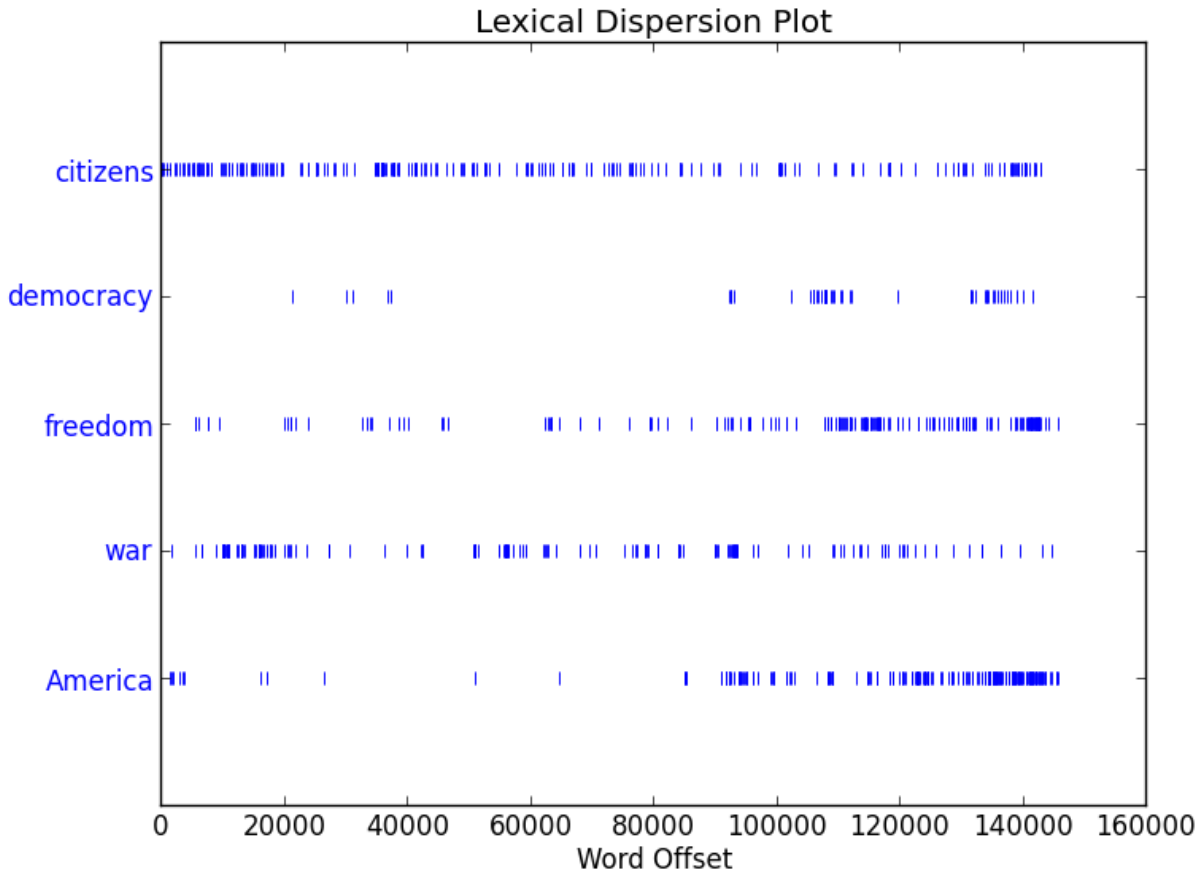
```
>>> text4.collocations()
United States; fellow citizens; four years; years ago; Federal
Government; General Government; American people; Vice President;
Old
World; Almighty God; Fellow citizens; Chief Magistrate; Chief
Justice;
God bless; every citizen; Indian tribes; public debt; foreign
nations; political parties
```

Counting

- Counting vocabulary: the length of a text from start to finish.
- How many distinct words?
- Richness of the text.

```
>>> len(text4)
145735
>>> len(set(text4)) #types
9754
>>> len(text4) / len(set(text4))
14.941049825712529
>>> 100 * text4.count('democracy') / len(text4)
0.03568120218204275
```

Positions of words in text



Generating similar text

Generating text in the style of the inaugural address:

```
>>> text4.generate()
Building ngram index...
Fellow - Citizens :  Called upon to make it strong ; where we may
safely give the assurance of perfect security which is the challenge
to our nation the duty of those who can limit the world promises only
such meager justice as the rule of law , based on the part of a
continent , saved the union , and opportunity .  So many events have
proved faithful both in peace and prosperity of both was effected by
this philosophy , many of them seriously convulsed .  Destructive wars
ensued , which has always worked perfectly .  It will
```

List elements operations

- List comprehension

```
>>> len(set([word.lower() for word in text4 if len(word)>5]))
7339
>>> [w.upper() for w in text4[0:5]]
['FELLOW', '-', 'CITIZENS', 'OF', 'THE']
```

- Loops and conditionals

```
for word in text4[0:5]:
    if len(word)<5 and word.endswith('e'):
        print word, ' is short and ends with e'
    elif word.istitle():
        print word, ' is a titlecase word'
    else:
        print word, 'is just another word'
```

Text corpora

- Corpus
  - Large collection of text
  - Raw or categorized
  - Concentrate on a topic or open domain
- Examples:
  - Brown - first, largest corpus, categorized by genre
  - Webtext - reviews, forums, etc.
  - Reuters - news corpus
  - Inaugural - US presidents' inaugural addresses
  - udhr - multilingual

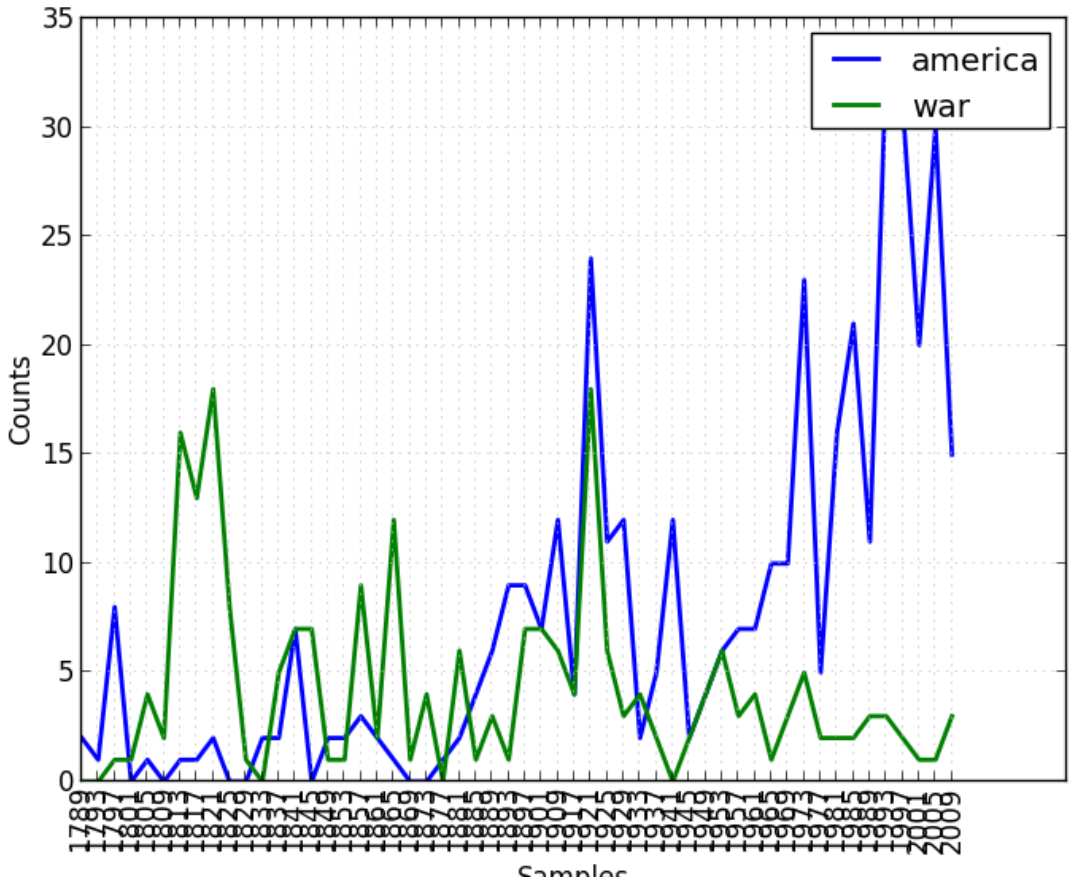
Basic corpus operations

- fileids() and categories()
- work with raw content, words, sentences, locations

```
>>> from nltk.corpus import inaugural
>>> inaugural.fileids()[:2]
['1789-Washington.txt', '1793-Washington.txt']
```

```
cfd = nltk.ConditionalFreqDist(
    (target, fileid[:4])
    for fileid in inaugural.fileids()
    for w in inaugural.words(fileid)
    for target in ['america', 'war']
    if w.lower().startswith(target))
cfd.plot()
```

Conditional frequencies



Differences between categories

- Modal verbs in various genres

```
>>> from nltk import FreqDist
>>> verbs=["should", "may", "can"]
>>> genres=["news", "government", "romance"]
>>> for g in genres:
...  words=brown.words(categories=g)
...  freq=FreqDist([w.lower() for w in words if w.lower() in
verbs])
...  print g, freq
news <FreqDist:  'can':  94, 'may':  93, 'should':  61>
government <FreqDist:  'may':  179, 'can':  119, 'should':  113>
romance <FreqDist:  'can':  79, 'should':  33, 'may':  11>
```

WordNet

- Structured, semantically oriented English dictionary
- Synonyms, antonyms, hyponims, hypernims, depth of a synset, trees, entailments, etc.

```
>>> from nltk.corpus import wordnet as wn
>>> wn.synsets('motorcar')
[Synset('car.n.01')]
>>> wn.synset('car.n.01').lemma_names
['car', 'auto', 'automobile', 'machine', 'motorcar']
>>> wn.synset('car.n.01').definition
'a motor vehicle with four wheels; usually propelled by an internal
combustion engine'
>>> for synset in wn.synsets('car')[1:3]:
...  print synset.lemma_names
['car', 'railcar', 'railway_car', 'railroad_car']
['car', 'gondola']
>>> wn.synset('walk.v.01').entailments()# Walking involves stepping
[Synset('step.v.01')]
```

Importing online text

- NLTK provides a helper function for getting text out of HTML

```
>>> from urllib import urlopen
>>> url = "http://www.bbc.co.uk/news/science-environment-21471908"
>>> html = urlopen(url).read()
html[:60]
>>> raw = nltk.clean_html(html)
'<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML+RDFa 1.0//EN" "http'
>>> tokens = nltk.word_tokenize(raw)
>>> tokens[:15]
['BBC', 'News', '-', 'Exoplanet', 'Kepler', '37b', 'is', 'tiniest',
'yet', '-', 'smaller', 'than', 'Mercury', 'Accessibility', 'links']
```

- Still needs some cleaning.

User input and local data

- User input

```
>>> s = raw_input("Enter some text:  ")
```

- Use your own files on disk

```
>>> f = open('C:\Data\Files\UK_natl_2010_en_Lab.txt')
>>> raw = f.read()
>>> print raw[:100]
#Foreword by Gordon Brown.
This General Election is fought as our troops are bravely fighting
to def
```

Import files as corpus

- Import your own files as corpus

```
>>> from nltk.corpus import PlaintextCorpusReader
>>> corpus_root = "C:/Data/Files/"
>>> wordlists = PlaintextCorpusReader(corpus_root, '.*\.txt')
>>> wordlists.fileids()[:3]
['UK_natl_1987_en_Con.txt', 'UK_natl_1987_en_Lab.txt',
'UK_natl_1987_en_LibSDP.txt']
>>> wordlists.words('UK_natl_2010_en_Lab.txt')
['#', 'Foreword', 'by', 'Gordon', 'Brown', '.', 'This', ...]
```

Regular expressions

- Review

	What does it do?
.	Wildcard, matches any character
^abc	Some pattern abc at the start of a string
abc$	Some pattern abc at the end of a string
[abc]	One of a set of characters
[A-Z0-9]	One of a range of characters
ed|ing|s	One of the specified strings (disjunction)
*	Zero or more of previous item, e.g. a*, [a-z]*
+	One or more of previous item, e.g. a+, [a-z]+
{n}	Exactly n repeats where n is a non-negative integer
a(b|c)+	Parentheses that indicate the scope of the operators

- Exercise: find and count all vowels in the string "supercalifragilisticexpialidocious"

Tokenization, normalization

- Tokenize raw text - divide into tokens. Multiple methods.
  ```
  tokens = nltk.word_tokenize(raw)
  tokens[:10]
  ```
  Regular expression tokenizer: nltk.regexp_tokenize(text, pattern)
- Test your tokenized text against raw text that has been manually tokenized, for example a sample of the Penn Treebank Data.
- Normalize tokens - remove upper case and capital letter.
  ```
  set(w.lower() for w in tokens)
  ```

Stemming and lemmatization

- Stemming - strip off affixes.

```
porter = nltk.PorterStemmer()
lancaster = nltk.LancasterStemmer()
[porter.stem(t) for t in tokens]
[lancaster.stem(t) for t in tokens]
```

- Porter stemmer lying - lie, women - women;
- Lancaster stemmer lying - lying, women - wom.
- Lemmatization - only removes affixes if the resulting word is in the dictionary.

```
wnl = nltk.WordNetLemmatizer()
[wnl.lemmatize(t) for t in tokens]
```

- lying - lying, women - woman.

Segmentation - sentence

- The Punkt sentence segmenter.
- Problems: some periods mark abbreviations (and end a sentence)
  - `>>> sent_tokenizer=nltk.data.load('tokenizers/punkt/english.pickle')`
  - `>>> sents = sent_tokenizer.tokenize(raw)`
  - `>>> pprint.pprint(sents[182:185])`
  - `['We have made the new fiscal responsibility framework legally binding, and we will maintain our inflation target of two per cent so that mortgage rates can be kept as low as possible.',`
  - `'#Rebuilding our banking system.',`
  - `'The international banking system played a key role in fuelling the most severe global recession since the Second World War.']`

Segmentation - word

- Word segmentation
  - in some cases there are no visual representations of word boundaries
  - foreign languages, processing of spoken language
  - search and identify - optimize based on a scoring function (objective function)

Writing output to file

- Save separated sentences text to a new file

```
output_file = open('C:\Data\Files\output.txt', 'w')
words = set(sents)
for word in sorted(words):
output_file.write(word + "\n")
```

- Can also write non-text data, but you must first convert it to string: str()
- Avoid filenames that contain space characters or that are identical except for case distinctions.

POS tagging

- The process of classifying words into their parts of speech and labeling them accordingly
  - Words grouped into classes, such as nouns, verbs, adjectives, and adverbs.
- Parts of speech are also known as word classes or lexical categories.
- The collection of tags used for a particular task is known as a tagset.
- NLTK: tagging text automatically.
- But.. why??
  - predicting the behavior of previously unseen words
  - analyzing word usage in corpora
  - text-to-speech systems
  - powerful searches
  - classification

Methods for tagging

- Multiple methods available:
  - default tagger, regular expression tagger, unigram tagger and n-gram taggers.
- Can be combined using a technique known as backoff.
  - when a more specialized model (such as a bigram tagger) cannot assign a tag in a given context, we backoff to a more general model (such as a unigram tagger).
- Taggers can be trained and evaluated using tagged corpora.

Tagging examples

- Some corpora already tagged

```
>>> nltk.corpus.brown.tagged_words()
[('The', 'AT'), ('Fulton', 'NP-TL'), ...]
```

- A simple example:

```
>>> nltk.pos_tag(text)
[('And', 'CC'), ('now', 'RB'), ('for', 'IN'), ('something', 'NN'),
('completely', 'RB'), ('different', 'JJ')]
```

  - CC is coordinating conjunction; RB is adverb; IN is preposition; NN is noun; JJ is adjective.
  - Lots of others - foreign term, verb tenses, "wh" determiner, etc.

- An example with homonyms:

```
>>> text = nltk.word_tokenize("They refuse to permit us to obtain
the refuse permit")
>>> nltk.pos_tag(text)
[('They', 'PRP'), ('refuse', 'VBP'), ('to', 'TO'), ('permit',
'VB'), ('us', 'PRP'), ('to', 'TO'), ('obtain', 'VB'), ('the',
'DT'), ('refuse', 'NN'), ('permit', 'NN')]
```

Unigram and N-gram tagging

- Unigram tagging - nltk.UnigramTagger()
  - assign the tag that is most likely for that particular token
  - train it specifying tagged sentence data as a parameter when we initialize the tagger
  - separate training and testing data
- N-gram tagging
  - context is the current word together with the part-of-speech tags of the n-1 preceding tokens
  - evaluate performance
  - contexts that were not present in the training data - accuracy vs. coverage
  - combine taggers

Information extraction

- Search large bodies of unrestricted text for specific types of entities and relations
- Put them in well-organized databases.
- Databases can then be used to find answers for specific questions.
- Steps:
  - segmenting, tokenizing, and part-of-speech tagging the text.
  - resulting data is then searched for specific types of entity.
  - examine entities that are mentioned near one another in the text to determine if specific relationships hold between those entities.

Chunking, entity recognition and relation extraction

- Entity recognition performed using chunkers
  - segment multi-token sequences, and label them with the appropriate entity type.
  - ORGANIZATION, PERSON, LOCATION, DATE, TIME, MONEY, and GPE (geo-political entity).
- Constructing chunkers:
  - using rule-based systems, such as the RegexpParser class provided by NLTK
  - using machine learning techniques, such as the ConsecutiveNPChunker
  - POS tags are very important in this context.
- Relation extraction
  - using rule-based systems which look for specific patterns in the text that connect entities and the intervening words
  - using machine-learning systems which attempt to learn patterns automatically from a training corpus.

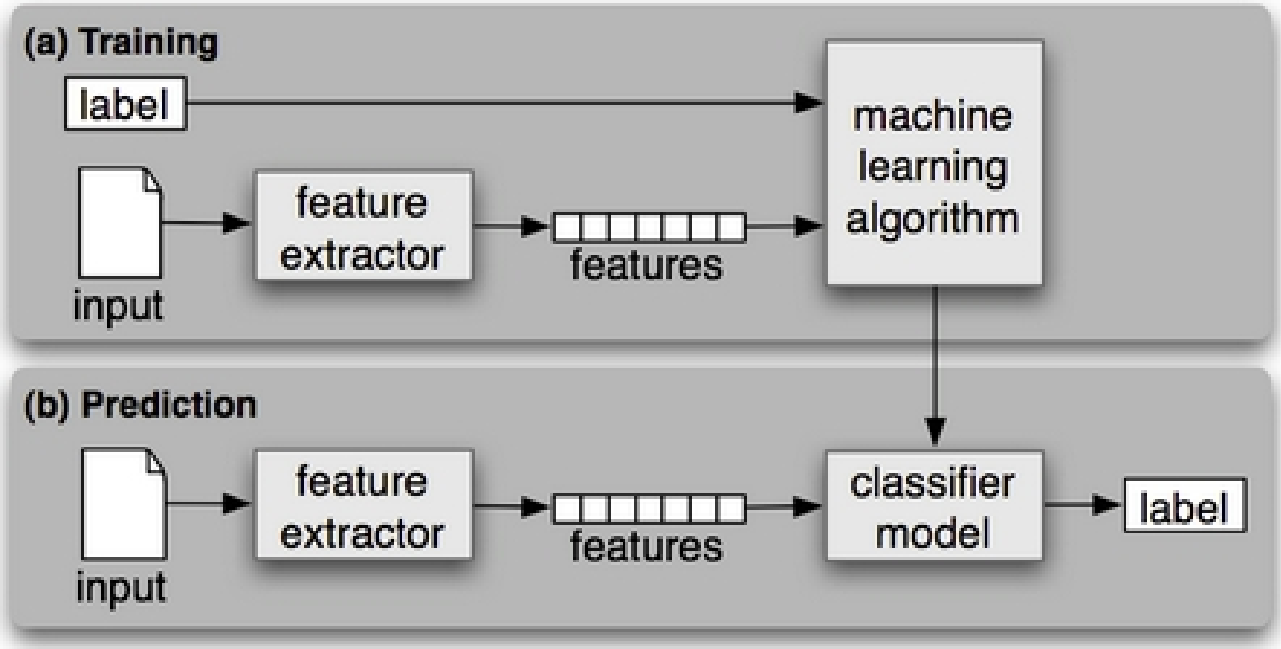
The basics

- What do we have to do?
  - choose a particular class label for a given input
  - identify particular features of language data that are salient for classifying it
  - construct models of language that can be used to perform language processing tasks automatically
  - learn about text/language from these models
- Machine learning techniques used:
  - decision trees
  - naive Bayes' classifiers
  - maximum entropy classifiers

Application

- Determining the topic of an article or a book
- Deciding if an email is spam or not
- Determining who wrote a text
- Determining the meaning of a word in a particular context
- Open-class classification - set of labels is not defined in advance
- Multi-class classification - each instance may be assigned multiple labels
- Sequence classification - a list of inputs are jointly classified.

Supervised classification



Example: gender identification by name

- Relevant feature: last letter
- Create a feature set (a dictionary) that maps from features' names to their values

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

- Import names, shuffle them

```
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')])
random.shuffle(names)
```

Example: gender identification

- Divide list of features into training set and test set

```
featuresets = [(gender_features(n), g) for (n,g) in names]
from nltk.classify import apply_features # use apply if you're
working with large corpora
train_set = apply_features(gender_features, names[500:])
test_set = apply_features(gender_features, names[:500])
```

- Use training set to train a naive Bayes classifier

```
classifier = nltk.NaiveBayesClassifier.train(train_set)
```

Example: gender identification

- Test the classifier on unseen data

```
>>> classifier.classify(gender_features('Neo'))
'male'
>>> classifier.classify(gender_features('Trinity'))
'female'
>>> print nltk.classify.accuracy(classifier, test_set)
0.744
```

- Examine the classifier to see which feature is most effective at distinguishing between classes

```
>>> classifier.show_most_informative_features(5)
Most Informative Features
last_letter = 'a' female :  male = 35.7 :  1.0
last_letter = 'k' male :  female = 31.7 :  1.0
last_letter = 'f' male :  female = 16.6 :  1.0
last_letter = 'p' male :  female = 11.9 :  1.0
last_letter = 'v' male :  female = 10.5 :  1.0
```

Example: document classification

- Use corpora where documents have been labeled with categories
  - build classifiers that will automatically tag new documents with appropriate category labels
- Use the Movie Review Corpus, which categorizes reviews as positive or negative to construct a list of documents
- Define a feature extractor for documents - feature for each of the most frequent 2000 words in the corpus
- Define a feature extractor that checks if words are present in a document
- Train a classifier to label new movie reviews

Example: document classification

- Compute accuracy on the test set

```
>>> print nltk.classify.accuracy(classifier, test_set)
>>> 0.79
```

  - evaluation issues: size of the test set depends on number of labels, their balance and the diversity of the test.

- Show most informative features

```
>>> classifier.show_most_informative_features(5)
Most Informative Features
contains(outstanding) = True pos :  neg = 11.2 :  1.0
contains(mulan) = True pos :  neg = 8.9 :  1.0
contains(wonderfully) = True pos :  neg = 8.5 :  1.0
contains(seagal) = True neg :  pos = 8.3 :  1.0
contains(damon) = True pos :  neg = 6.0 :  1.0
```

More: context, sequence classification, Markov models

- Contextual features often provide powerful clues for classification
- Context-dependent feature extractor: pass in a complete (untagged) sentence, along with the index of the target word.
- Joint classifier models - choose an appropriate labeling for a collection of related inputs.
  - sequence classification - jointly choose part-of-speech tags for all the words in a given sentence.
  - consecutive classification - find the most likely class label for the first input, then to use that answer to help find the best label for the next input, repeat.
  - feature extraction function needs to take a "history" argument - list of tags predicted so far

Markov models

- Hidden Markov models
  - use inputs and the history of predicted tags
  - generate a probability distribution over tags
  - combine probabilities to calculate scores for sequences
  - choose tag sequence with the highest probability
- More advanced models: Maximum Entropy Markov Models and Linear-Chain Conditional Random Field Models.

Other things to learn

- Analyzing sentence structure
- Analyzing the meaning of sentences
  - natural language understanding
  - discourse processing
- Managing data
  - Working with XML and Toolbox data

- For questions and comments, please email me at: iulia.cioroianu[AT]nyu[DOT]edu

- Thank you!