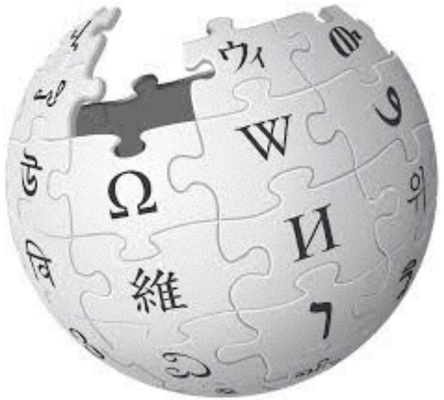
# Applied Text Mining in Python

*Introduction to Text Mining*

# Text is Everywhere!



# Text data is growing fast!

- **Data continues to grow exponentially**
  - Estimated to be 2.5 Exabytes (2.5 million TB) a day
  - Grow to 40 Zettabytes (40 billion TB) by 2020 (50-times that of 2010)

- **Approximately 80% of all data is estimated to be unstructured, text-rich data**
  - >40 million articles (5 million in English) in Wikipedia
  - >4.5 billion Web pages
  - >500 million tweets a day, 200 billion a year
  - >1.5 trillion queries / searches on Google a year

# Data hidden in plain sight

Social network

Author

Description

Location

Tweet
- Topic
- Sentiment

Time

Popularity

TWEETS 14.6K | FOLLOWING 994 | FOLLOWERS 391K | LIKES 49 | LISTS 3 | Follow

**UN Spokesperson**
@UN_Spokesperson

Official Twitter account of the Office of the Spokesperson for United Nations Secretary-General Ban Ki-moon.

New York, USA
un.org/sg/spokesperso...
Joined May 2010

Tweet to UN Spokesperson

3,008 Photos and videos

Tweets | Tweets & replies | Media

**UN Spokesperson** @UN_Spokesperson · 3h
Maintaining unity is crucial in tackling security challenges on Korean Peninsula & beyond: #UNSG on #DPRK sanctions bit.ly/2gVeX7z
2 7 14

**UN Spokesperson** @UN_Spokesperson · 17h
"Ethics are built right into the ideals and objectives of the United Nations" #UNSG @ NY Society for Ethical Culture bit.ly/2guVelr
6 13 27

**UN Spokesperson** @UN_Spokesperson · 23h
Ban on Amb.Joseph V. Reed: The UN family is fortunate to have had such a wonderful supporter, wonderful leader. bit.ly/2gFU1yp

# So, what can be done with text?

- Parse text
- Find / Identify / Extract relevant information from text
- Classify text documents
- Search for relevant text documents
- Sentiment analysis
- Topic modeling
- …

# Applied Text Mining in Python

*Handling Text in Python*

# Primitive constructs in Text

- Sentences / input strings
- Words or Tokens
- Characters
- Document, larger files

And their properties …

# Let's try it out!

```
>>> text1 = "Ethics are built right into the ideals and objectives
of the United Nations "
>>> len(text1)
76
>>> text2 = text1.split(' ')
>>> len(text2)
13
>>> text2
['Ethics', 'are', 'built', 'right', 'into', 'the', 'ideals', 'and',
'objectives', 'of', 'the', 'United', 'Nations', '']
```

# Finding specific words

- **Long words: Words that are most than 3 letters long**

```
>>> [w for w in text2 if len(w) > 3]
['Ethics', 'built', 'right', 'into', 'ideals', 'objectives', 'United',
'Nations']
```

- **Capitalized words**

```
>>> [w for w in text2 if w.istitle()]
['Ethics', 'United', 'Nations']
```

- **Words that end with s**

```
>>> [w for w in text2 if w.endswith('s')]
['Ethics', 'ideals', 'objectives', 'Nations']
```

# Finding unique words: using set()

```
>>> text3 = 'To be or not to be'
>>> text4 = text3.split(' ')
>>> len(text4)
6
>>> len(set(text4))
5
>>> set(text4)
set(['not', 'To', 'or', 'to', 'be'])
>>> len(set([w.lower() for w in text4]))
4
>>> set([w.lower() for w in text4])
set(['not', 'to', 'or', 'be']
```

# Some word comparison functions …

- s.startswith(t)
- s.endswith(t)
- t in s
- s.isupper(); s.islower(); s.istitle()
- s.isalpha(); s.isdigit(); s.isalnum()

# String Operations

- s.lower(); s.upper(); s.titlecase()
- s.split(t)
- s.splitlines()
- s.join(t)
- s.strip(); s.rstrip()
- s.find(t); s.rfind(t)
- s.replace(u, v)

# From words to characters

```
>>> text5 = 'ouagadougou'
>>> text6 = text5.split('ou')
>>> text6
['', 'agad', 'g', '']
>>> 'ou'.join(text6)
'ouagadougou'
```

```
>>> text5.split('')
Traceback (most recent call last):
  File "<stdin>", line 1, in
<module>
ValueError: empty separator
>>> list(text5)
['o', 'u', 'a', 'g', 'a', 'd',
'o', 'u', 'g', 'o', 'u']
>>> [c for c in text5]
['o', 'u', 'a', 'g', 'a', 'd',
'o', 'u', 'g', 'o', 'u']
```

# Cleaning Text

```
>>> text8 = '    A quick brown fox jumped over the lazy dog. '
>>> text8.split(' ')
['', '', '\t', 'A', 'quick', 'brown', 'fox', 'jumped', 'over',
'the', 'lazy', 'dog.', '']
>>> text9 = text8.strip()
>>> text9.split(' ')
['A', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'lazy',
'dog.']
```

# Changing Text

- **Find and replace**

```
>>> text9
'A quick brown fox jumped over the lazy dog.'
>>> text9.find('o')
10
>>> text9.rfind('o')
40
>>> text9.replace('o', 'O')
'A quick brOwn fOx jumped Over the lazy dOg.'
```

# Handling Larger Texts

- **Reading files line by line**

```
>>> f = open('UNDHR.txt', 'r')
>>> f.readline()
'Universal Declaration of Human Rights\n’
```

- **Reading the full file**

```
>>> f.seek(0)
>>> text12 = f.read()
>>> len(text12)
10891
>>> text13 = text12.splitlines()
>>> len(text13)
158
>>> text13[0]
'Universal Declaration of Human Rights'
```

# File Operations

- f = open(*filename*, *mode*)
- f.readline(); f.read(); f.read(*n*)
- for *line* in f: doSomething(*line*)
- f.seek(*n*)
- f.write(*message*)
- f.close()
- f.closed

# Issues with reading text files

```
>>> f = open('UNDHR.txt', 'r')
>>> text14 = f.readline()
'Universal Declaration of Human Rights\n'
```

- **How do you remove the last newline character?**

```
>>> text14.rstrip()
'Universal Declaration of Human Rights'
```

  - Works also for DOS newlines (`^M`) that shows up as `'\r'` or `'\r\n'`

# Take Home Concepts

- Handling text sentences
- Splitting sentences into words, words into characters
- Finding unique words
- Handling text from documents

# Applied Text Mining in Python

*Regular Expressions*

# Processing Free-text

```
>>> text10 = '"Ethics are built right into the ideals and
objectives of the United Nations" #UNSG @ NY Society for Ethical
Culture bit.ly/2guVelr @UN @UN_Women'
>>> text11 = text10.split(' ')
>>> text11
['"Ethics', 'are', 'built', 'right', 'into', 'the', 'ideals',
'and', 'objectives', 'of', 'the', 'United', 'Nations"', '#UNSG',
'@', 'NY', 'Society', 'for', 'Ethical', 'Culture', 'bit.ly/
2guVelr', '@UN', '@UN_Women']
```

- **How do you find all Hashtags? Callouts?**

# Finding Specific Words

- **Hashtags**

```
>>> [w for w in text11 if w.startswith('#')]
['#UNSG']
```

- **Callouts**

```
>>> [w for w in text11 if w.startswith('@')]
['@', '@UN', '@UN_Women']
```

# Finding patterns with regular expressions

- Callouts are more than just tokens beginning with '@'

  @UN_Spokesperson @katyperry @coursera

- Match *something* after '@'
  - Alphabets
  - Numbers
  - Special symbols like '_'

@[A-Za-z0-9_]+

# Let's try it out!

```
>>> text10 = '"Ethics are built right into the ideals and objectives of the
United Nations" #UNSG @ NY Society for Ethical Culture bit.ly/2guVelr @UN
@UN_Women'
>>> text11 = text10.split(' ')
>>> [w for w in text11 if w.startswith('@')]
['@', '@UN', '@UN_Women']
```

## Import regular expressions first!

```
>>> import re
>>> [w for w in text11 if re.search('@[A-Za-z0-9_]+', w)]
['@UN', '@UN_Women']
```

# Parsing the callout regular expression

```
@[A-Za-z0-9_]+
```

```
@        A-Za-z0-9_        [ ]+
```

- starts with @
- followed by any alphabet (upper or lower case), digit, or underscore
- that repeats at least once, but any number of times

# Meta-characters: Character matches

`.` : wildcard, matches a single character
`^` : start of a string
`$` : end of a string
`[ ]` : matches one of the set of characters within `[ ]`
`[a-z]` : matches one of the range of characters `a`, `b`, …, `z`
`[^abc]` : matches a character that is not `a`, `b`, or, `c`
`a|b` : matches either `a` or `b`, where `a` and `b` are strings
`( )` : Scoping for operators
`\` : Escape character for special characters (`\t`, `\n`, `\b`)

# Meta-characters: Character symbols

`\b` : **Matches word boundary**

`\d` : **Any digit, equivalent to** `[0-9]`

`\D` : **Any non-digit, equivalent to** `[^0-9]`

`\s` : **Any whitespace, equivalent to** `[ \t\n\r\f\v]`

`\S` : **Any non-whitespace, equivalent to** `[^ \t\n\r\f\v]`

`\w` : **Alphanumeric character, equivalent to** `[a-zA-Z0-9_]`

`\W` : **Non-alphanumeric, equivalent to** `[^a-zA-Z0-9_]`

# Meta-characters: Repetitions

`*` : matches zero or more occurrences

`+` : matches one or more occurrences

`?` : matches zero or one occurrences

`{n}` : exactly `n` repetitions, `n`$\geq$ 0

`{n,}` : at least `n` repetitions

`{,n}` : at most `n` repetitions

`{m,n}` : at least `m` and at most `n` repetitions

# Recall the callout regular expression

```
>>> text10 = ‘”Ethics are built right into the ideals and
objectives of the United Nations” #UNSG @ NY Society for Ethical
Culture bit.ly/2guVelr @UN @UN_Women’
>>> text11 = text10.split(' ')

>>> [w for w in text11 if re.search('@[A-Za-z0-9_]+', w)]
['@UN', '@UN_Women']

>>> [w for w in text11 if re.search('@\w+', w)]
['@UN', '@UN_Women']
```

# Let’s look at some more examples!

- **Finding specific characters**

```
>>> text12 = ‘ouagadougou’

>>> re.findall(r'[aeiou]', text12)
['o', 'u', 'a', 'a', 'o', 'u', 'o', 'u']

>>> re.findall(r'[^aeiou]', text12)
['g', 'd', 'g’]
```

# Case study: Regular expression for Dates

- **Date variations for 23rd October 2002**

```
23-10-2002
23/10/2002
23/10/02
10/23/2002
23 Oct 2002
23 October 2002
Oct 23, 2002
October 23, 2002
```

\d{2}[/-]\d{2}[/-]\d{4}

# Regular Expression for Dates (contd.)

```
>>> dateStr = '23-10-2002\n23/10/2002\n23/10/02\n10/23/2002\n23 Oct 2002\n23
October 2002\nOct 23, 2002\nOctober 23, 2002\n'

>>> re.findall(r'\d{2}[/-]\d{2}[/-]\d{4}', dateStr)
['23-10-2002', '23/10/2002', '10/23/2002']

>>> re.findall(r'\d{2}[/-]\d{2}[/-]\d{2,4}', dateStr)
['23-10-2002', '23/10/2002', '23/10/02', '10/23/2002']

>>> re.findall(r'\d{1,2}[/-]\d{1,2}[/-]\d{2,4}', dateStr)
['23-10-2002', '23/10/2002', '23/10/02', '10/23/2002']
```

```
23-10-2002
23/10/2002
23/10/02
10/23/2002
```

# Regex for Dates (contd.)

```
23 Oct 2002
23 October 2002
Oct 23, 2002
October 23, 2002
```

```
>>> re.findall(r'\d{2} (Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)
\d{4}', dateStr)
['Oct']
```

```
>>> re.findall(r'\d{2} (?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)
\d{4}', dateStr)
['23 Oct 2002']
```

```
>>> re.findall(r'\d{2} (?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)[a-
z]* \d{4}', dateStr)
['23 Oct 2002', '23 October 2002']
```

# Regex for Dates (contd.)

```
23 Oct 2002
23 October 2002
Oct 23, 2002
October 23, 2002
```

```
>>> re.findall(r'(?:\d{2} )?(?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|
Dec)[a-z]* (?:\d{2}, )?\d{4}', dateStr)
['23 Oct 2002', '23 October 2002', 'Oct 23, 2002', 'October 23, 2002']
```

```
>>> re.findall(r'(?:\d{1,2} )?(?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|
Dec)[a-z]* (?:\d{1,2}, )?\d{4}', dateStr)
['23 Oct 2002', '23 October 2002', 'Oct 23, 2002', 'October 23, 2002']
```

# Take Home Concepts

- What are regular expressions?
- Regular expression meta-characters
- Building a regular expression to identify dates

# Applied Text Mining in Python

*Internationalization*

# World of Languages

# English and ASCII

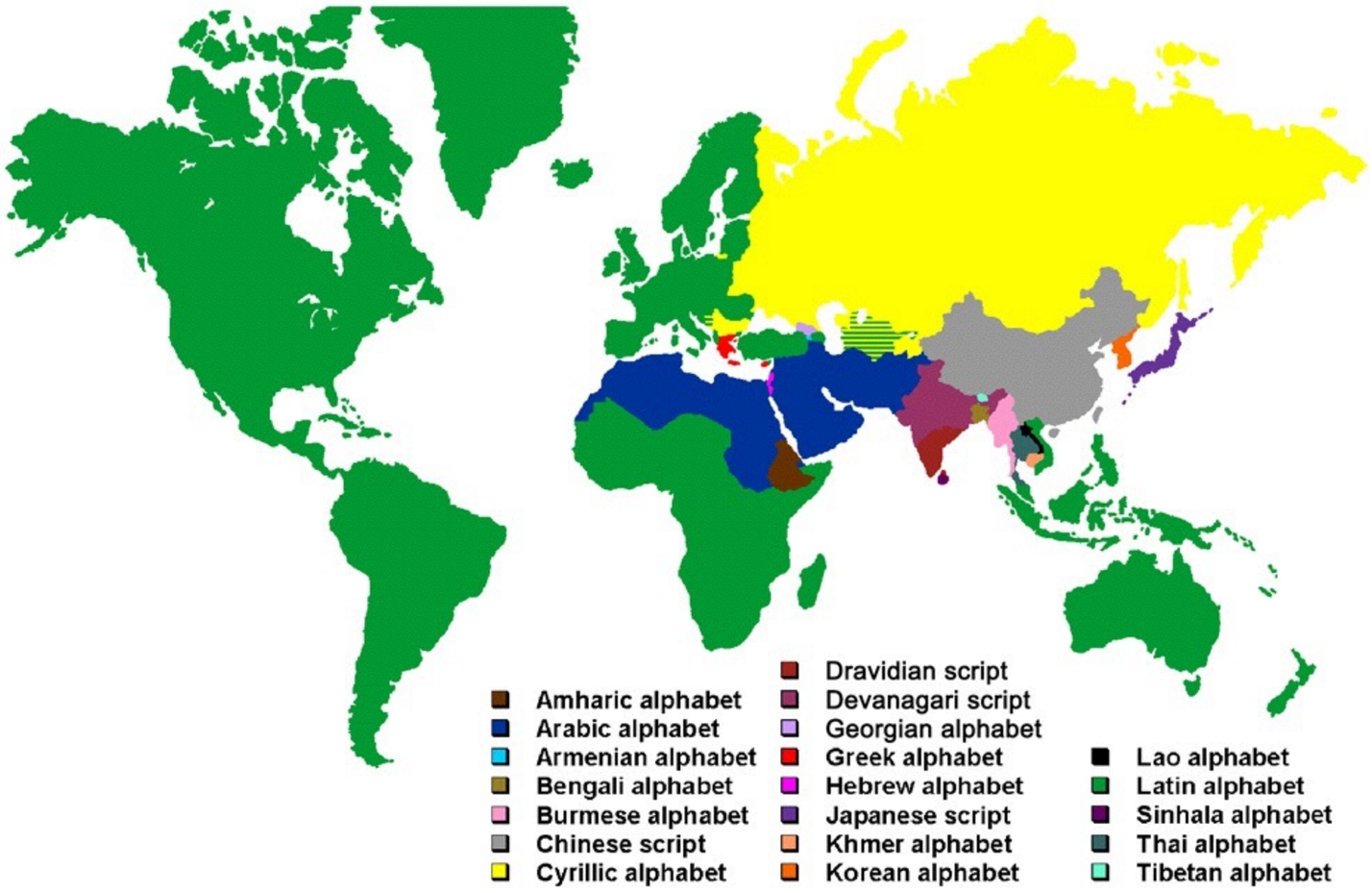
- **ASCII: American Standard Code for Information Interchange**
  - 7-bit character encoding standard: 128 valid codes
  - Range: 0x00 – 0x7F [$(0000\ 0000)_2$ to $(0111\ 1111)_2$]
  - Includes alphabets (upper and lower cases), digits, punctuations, common symbols, control characters
  - Worked (relatively) well for English typewriting

# Resume vs. Résumé

- **Diacritics**
  - résumé :: resume
  - naïve :: naive
  - café :: cafe
  - Québec
  - Zürich
  - Fédération Internationale de Football Association (FIFA)

- **International languages**
  - 基本上 सहायक ασπασθ универсальной
  - ♪ ♫ ♩
  - ☺ ☹

# Written Scripts

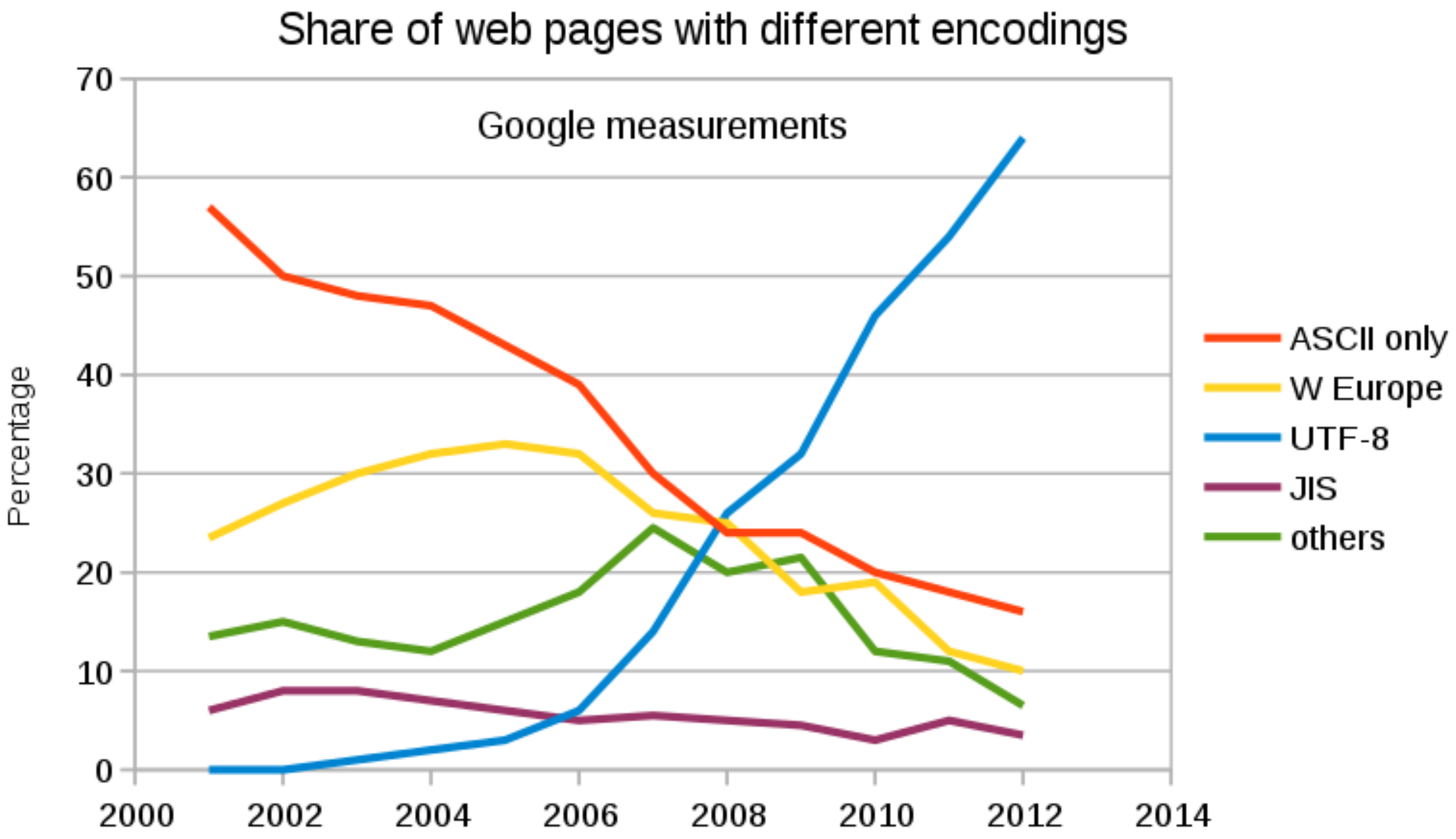



- Latin: 36% (2.6B people)
- Chinese: 18% (1.3B)
- Devanagari: 14% (1B)
- Arabic: 14% (1B)
- Cyrillic: 4% (0.3B)
- Dravidian: 3.5% (0.25B)

# Other Character Encodings

- IBM EBCDIC
- Latin-1
- JIS: Japanese Industrial Standards
- CCCII: Chinese Character Code for Information Interchange
- EUC: Extended Unix Code
- Numerous other national standards

- Unicode and UTF-8


Share of web pages with different encodings
Google measurements
Percentage
70
60
50
40
30
20
10
0
2000
2002
2004
2006
2008
2010
2012
2014
ASCII only
W Europe
UTF-8
JIS
others

# Unicode

- Industry standard for encoding and representing text
- Over 128,000 characters from 130+ scripts and symbol sets
- Can be implemented by different character endings
  - UTF-8: One byte to up to four bytes
  - UTF-16: One or two 16-bit code units
  - UTF-32: One 32-bit code unit

# UTF-8

- Unicode Transformational Format – 8-bits
- Variable length encoding: One to four bytes
- Backward compatible with ASCII
  - One byte codes same as ASCII
- Dominant character encoding for the Web

- How to handle in Python?
  - Default in Python 3
  - In Python 2:

```
# -*- coding: utf-8 -*-
```

# Let's see an example: Résumé

## Python 3

```
>>> text1="Résumé"
>>> len(text1)
6
>>> text1
'Résumé'

>>> [c for c in text1]
['R', 'é', 's', 'u', 'm', 'é']
```

## Python 2

```
>>> text1="Résumé"
>>> len(text1)
8
>>> text1
'R\xc3\xa9sum\xc3\xa9'

>>> [c for c in text1]
['R', '\xc3', '\xa9', 's', 'u', 'm', '\xc3', '\xa9']

>>> text2=u'Résumé'
>>> len(text2)
6
>>> text2
u'R\xe9sum\xe9'
>>> [c for c in text2]
[u'R', u'\xe9', u's', u'u', u'm', u'\xe9']
```

# Take Home Concepts

- Diversity in Text
- ASCII and other character encodings
- Handling text in UTF-8

# Applied Text Mining in Python

*Basic Natural Language Processing*

# What is Natural Language?

- Language used for everyday communication by humans
  - English
  - 中文
  - हिन्दी
  - ру́сский язы́к
  - español

c u 2nite

compared to the artificial computer languages

# What is Natural Language Processing?

- Any computation, manipulation of natural language

- Natural languages evolve
  - new words get added
  - old words lose popularity
  - meanings of words change
  - language rules themselves may change

selfie

thou

learn

position of verbs in sentences!

# NLP Tasks: A Broad Spectrum

- Counting words, counting frequency of words
- Finding sentence boundaries
- Part of speech tagging
- Parsing the sentence structure
- Identifying semantic roles
- Identifying entities in a sentence
- Finding which pronoun refers to which entity

and much more …

# Applied Text Mining in Python

*Basic NLP tasks with NLTK*

# An Introduction to NLTK

- NLTK: Natural Language Toolkit
- Open source library in Python
- Has support for most NLP tasks
- Also provides access to numerous text corpora

# Let’s set it up first!

```
>>> import nltk
```

- **Let's get some text corpora**

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

```
>>> from nltk.book import *
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

# Let's set it up first! (2)

```
>>> text1
<Text: Moby Dick by Herman Melville 1851>

>>> sents()
sent1: Call me Ishmael .
sent2: The family of Dashwood had long been settled in Sussex .
sent3: In the beginning God created the heaven and the earth .
sent4: Fellow - Citizens of the Senate and of the House of Representatives :
sent5: I have a problem with people PMing me to lol JOIN
sent6: SCENE 1 : [ wind ] [ clop clop clop ] KING ARTHUR : Whoa there !
sent7: Pierre Vinken , 61 years old , will join the board as a nonexecutive director Nov.
29 .
sent8: 25 SEXY MALE , seeks attrac older single lady , for discreet encounters .
sent9: THE suburb of Saffron Park lay on the sunset side of London , as red and ragged as
a cloud of sunset .

>>> sent1
['Call', 'me', 'Ishmael', '.']
```

# Simple NLP tasks (1)

- **Counting vocabulary of words**

```
>>> text7
<Text: Wall Street Journal>
>>> sent7
['Pierre', 'Vinken', ',', '61', 'years', 'old', ',', 'will', 'join', 'the',
'board', 'as', 'a', 'nonexecutive', 'director', 'Nov.', '29', '.']
>>> len(sent7)
18
>>> len(text7)
100676
>>> len(set(text7))
12408
```

# Simple NLP tasks (2)

```
>>> list(set(text7))[:10]
[u'Mortimer', u'foul', u'Heights', u'four', u'spiders', u'railing', u'centimeter',
u'Until', u'payoff', u'Germany-based']
```

- **Frequency of words**

```
>>> dist = FreqDist(text7)
>>> len(dist)
12408
>>> vocab1 = dist.keys()
>>> list(vocab1)[:10]
[u'Mortimer', u'foul', u'Heights', u'four', u'spiders', u'railing', u'centimeter',
u'Until', u'payoff', u'Germany-based']
>>> dist[u'four']
20
>>> freqwords = [w for w in vocab1 if len(w) > 5 and dist[w] > 100]
>>> freqwords
[u'million', u'shares', u'market', u'president', u'trading', u'billion', u'company',
u'program', u'because']
```

# Normalization and Stemming

- **Different forms of the same "word"**

```
>>> input1 = "List listed lists listing listings"
>>> words1 = input1.lower().split(' ')
>>> words1
['list', 'listed', 'lists', 'listing', 'listings']

>>> porter = nltk.PorterStemmer()
>>> [porter.stem(t) for t in words1]
[u'list', u'list', u'list', u'list', u'list']
```

# Lemmatization

```
>>> udhr = nltk.corpus.udhr.words('English-Latin1')
>>> udhr[:20]
['Universal', 'Declaration', 'of', 'Human', 'Rights', 'Preamble', 'Whereas',
'recognition', 'of', 'the', 'inherent', 'dignity', 'and', 'of', 'the', 'equal', 'and',
'inalienable', 'rights', 'of']

>>> [porter.stem(t) for t in udhr[:20]]
[u'Univers', u'Declar', u'of', u'Human', u'Right', u'Preambl', u'Wherea', u'recognit',
u'of', u'the', u'inher', u'digniti', u'and', u'of', u'the', u'equal', u'and',
u'inalien', u'right', u'of']
```

- **Lemmatization: Stemming, but resulting stems are all valid words**

```
>>> WNlemma = nltk.WordNetLemmatizer()
>>> [WNlemma.lemmatize(t) for t in udhr[:20]]
['Universal', 'Declaration', 'of', 'Human', 'Rights', 'Preamble', 'Whereas',
'recognition', 'of', 'the', 'inherent', 'dignity', 'and', 'of', 'the', 'equal', 'and',
'inalienable', u'right', 'of']
```

# Tokenization

- **Recall splitting a sentence into words / tokens**

```
>>> text11 = "Children shouldn't drink a sugary drink before bed."
>>> text11.split(' ')
['Children', "shouldn't", 'drink', 'a', 'sugary', 'drink', 'before',
'bed.’]
```

- **NLTK has an in-built tokenizer**

```
>>> nltk.word_tokenize(text11)
['Children', 'should', "n't", 'drink', 'a', 'sugary', 'drink', 'before',
'bed', '.']
```

# Sentence Splitting

- **How would you split sentences from a long text string?**

```
>>> text12 = "This is the first sentence. A gallon of milk in the U.S. costs
$2.99. Is this the third sentence? Yes, it is!”
```

- **NLTK has an in-built sentence splitter too!**

```
>>> sentences = nltk.sent_tokenize(text12)
>>> len(sentences)
4
>>> sentences
['This is the first sentence.', 'A gallon of milk in the U.S. costs
$2.99.', 'Is this the third sentence?', 'Yes, it is!']
```

# Take Home Concepts

- NLTK is a widely used toolkit for text and natural language processing
- NLTK gives access to many corpora and handy tools
- Sentence splitting, tokenization, and lemmatization are important, and non-trivial, pre-processing tasks

# Applied Text Mining in Python

*Advanced NLP tasks with NLTK*

# NLP Tasks

- Counting words, counting frequency of words
- Finding sentence boundaries
- Part of speech tagging
- Parsing the sentence structure
- Identifying semantic role labeling
- Named Entity Recognition
- Co-reference and pronoun resolution

# Part-of-speech (POS) Tagging

- Recall high school grammar: nouns, verbs, adjectives, …
- Many more tags or word classes than just these

| Tag | Word class | Tag | Word class | Tag | Word class |
|---|---|---|---|---|---|
| CC | Conjunction | JJ | Adjective | PRP | Pronoun |
| CD | Cardinal | MD | Modal | RB | Adverb |
| DT | Determiner | NN | Noun | SYM | Symbol |
| IN | Preposition | POS | Possessive | VB | Verb |

…

```
>>> import nltk
>>> nltk.help.upenn_tagset('MD')
MD: modal auxiliary
    can cannot could couldn't dare may might must need ought
    shall should shouldn't will would
```

# POS Tagging with NLTK

- **Recall splitting a sentence into words / tokens**

```
>>> text11 = "Children shouldn't drink a sugary drink
before bed."
>>> text13 = nltk.word_tokenize(text11)
```

- **NLTK's Tokenizer**

```
>>> nltk.pos_tag(text13)
[('Children', 'NNP'), ('should', 'MD'), ("n't", 'RB'),
('drink', 'VB'), ('a', 'DT'), ('sugary', 'JJ'), ('drink',
'NN'), ('before', 'IN'), ('bed', 'NN'), ('.', '.')]
```

# Ambiguity in POS Tagging

- **Ambiguity is common in English**

**Visiting aunts can be a nuisance**

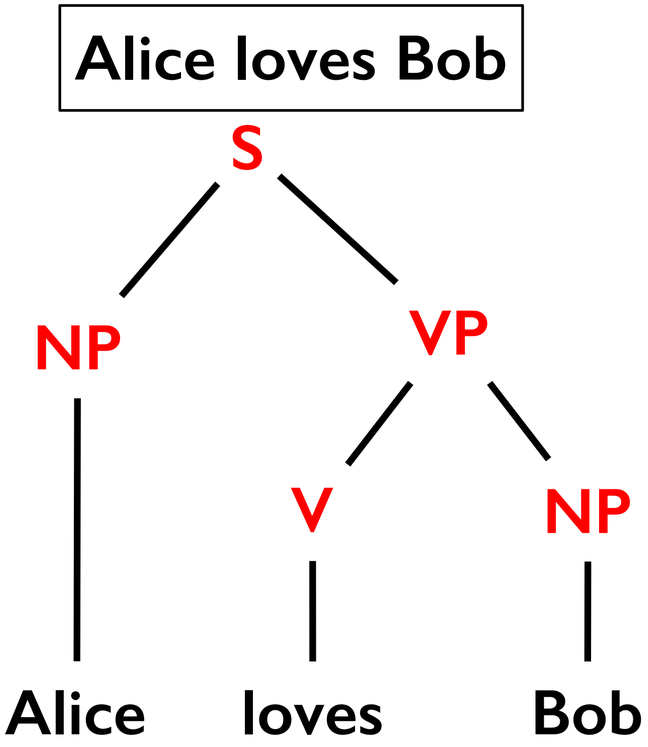
```
>>> text14 = nltk.word_tokenize("Visiting aunts can be a nuisance")
>>> nltk.pos_tag(text14)
[('Visiting', 'VBG'), ('aunts', 'NNS'), ('can', 'MD'), ('be', 'VB'), ('a', 'DT'),
('nuisance', 'NN')]
```

- **Another alternative POS tagging**

```
[('Visiting', 'JJ'), ('aunts', 'NNS'), ('can', 'MD'), ('be', 'VB'), ('a', 'DT'),
('nuisance', 'NN')]
```

# Parsing Sentence Structure

- Making sense of sentences is easy if they follow a well-defined grammatical structure



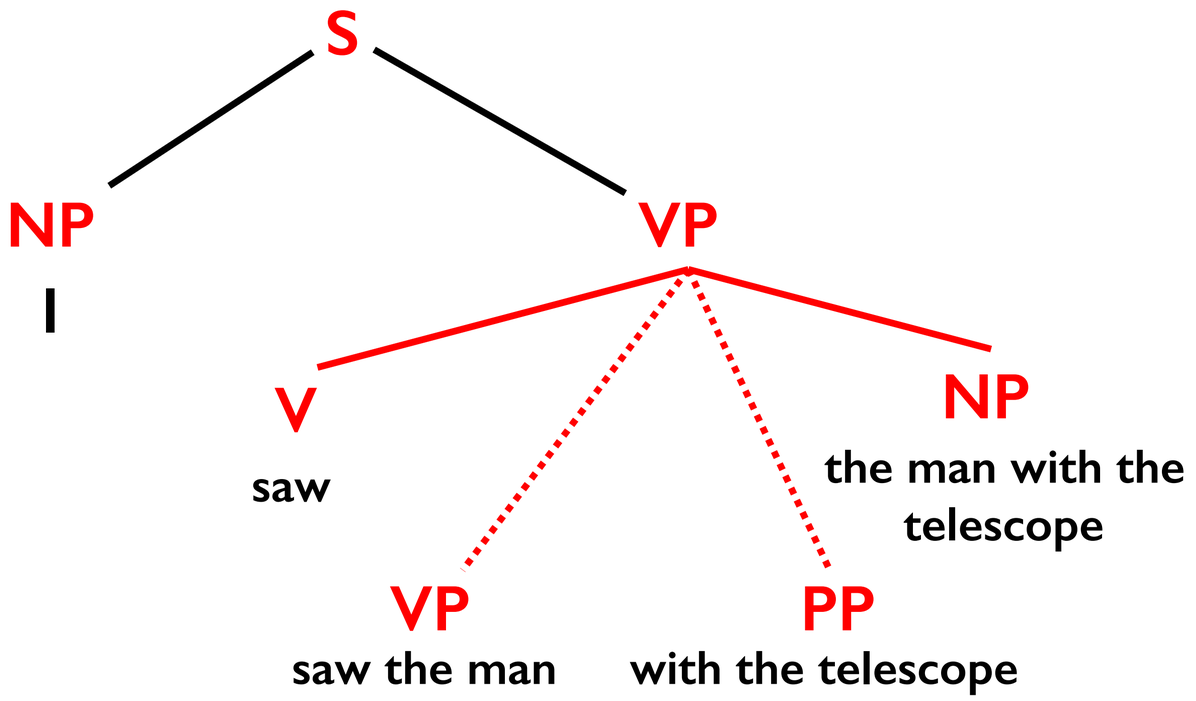

```
>>> text15 = nltk.word_tokenize("Alice loves Bob")
>>> grammar = nltk.CFG.fromstring("""
... S -> NP VP
... VP -> V NP
... NP -> 'Alice' | 'Bob'
... V -> 'loves'
... """)

>>> parser = nltk.ChartParser(grammar)
>>> trees = parser.parse_all(text15)
>>> for tree in trees:
...     print tree
...
(S (NP Alice) (VP (V loves) (NP Bob)))
```

# Ambiguity in Parsing

- Ambiguity may exist even if sentences are grammatically correct!

I saw the man with a telescope

```
S -> NP VP
VP -> V NP | VP PP
PP -> P NP
NP -> DT N | DT N PP | 'I'
DT -> 'a' | 'the'
N -> 'man' | 'telescope'
V -> 'saw'
P -> 'with'
```



# Ambiguity in Parsing

```
>>> text16 = nltk.word_tokenize("I saw the man with a telescope")
>>> grammar1 = nltk.data.load('mygrammar1.cfg')
>>> grammar1
<Grammar with 13 productions>
>>> parser = nltk.ChartParser(grammar1)
>>> trees = parser.parse_all(text16)
>>> for tree in trees:
...   print tree
...
(S
  (NP I)
  (VP
    (VP (V saw) (NP (DT the) (N man)))
    (PP (P with) (NP (DT a) (N telescope)))))
(S
  (NP I)
  (VP
    (V saw)
    (NP (DT the) (N man) (PP (P with) (NP (DT a) (N telescope))))))
```

```
S -> NP VP
VP -> V NP | VP PP
PP -> P NP
NP -> DT N | DT N PP | 'I'
DT -> 'a' | 'the'
N -> 'man' | 'telescope'
V -> 'saw'
P -> 'with'
```

mygrammar1.cfg

# NLTK and Parse Tree Collection

```
>>> from nltk.corpus import treebank
>>> text17 = treebank.parsed_sents('wsj_0001.mrg')[0]
>>> print text17
(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken))
    (, ,)
    (ADJP (NP (CD 61) (NNS years)) (JJ old))
    (, ,))
  (VP
    (MD will)
    (VP
      (VB join)
      (NP (DT the) (NN board))
      (PP-CLR (IN as) (NP (DT a) (JJ nonexecutive) (NN director)))
      (NP-TMP (NNP Nov.) (CD 29))))
  (. .))
```

# POS Tagging & Parsing Complexity

- **Uncommon usages of words**

**The old man the boat**

```
>>> text18 = nltk.word_tokenize("The old man the boat")
>>> nltk.pos_tag(text19)
[('The', 'DT'), ('old', 'JJ'), ('man', 'NN'), ('the', 'DT'), ('boat', 'NN')]
```

- **Well-formed sentences may still be meaningless!**

**Colorless green ideas sleep furiously**

```
>>> text19 = nltk.word_tokenize("Colorless green ideas sleep furiously")
>>> nltk.pos_tag(text18)
[('Colorless', 'NNP'), ('green', 'JJ'), ('ideas', 'NNS'), ('sleep', 'VBP'),
('furiously', 'RB')]
```

# Take Home Concepts

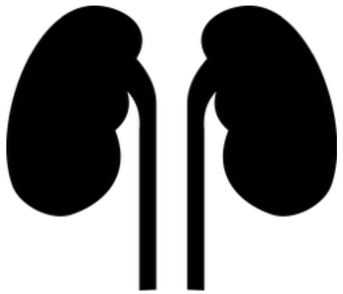
- POS tagging provides insights into the word classes / types in a sentence
- Parsing the grammatical structures helps derive meaning
- Both tasks are difficult, linguistic ambiguity increases the difficulty even more
- Better models could be learned with supervised training
- NLTK provides access to tools and data for training

# Applied Text Mining in Python

*Text Classification*

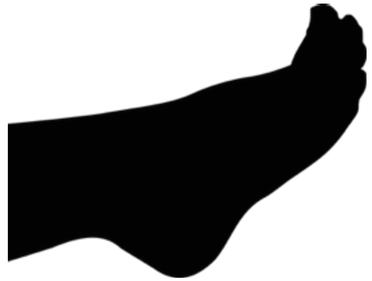
# Which medical speciality does this relate to?

TINEA PEDIS, or ATHLETE'S FOOT, is a very common fungal skin infection of the foot. It often first appears between the toes. It can be a one-time occurrence or it can be chronic. The fungus, known as Trichophyton, thrives under warm, damp conditions so people whose feet sweat a great deal are more susceptible. It is easily transmitted in showers and pool walkways. Those people with immunosuppressive conditions, such as diabetes mellitus, are also more susceptible to athlete's foot.



Nephrology

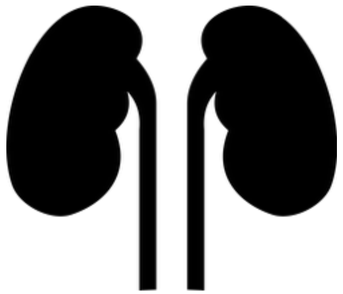
Neurology



Podiatry

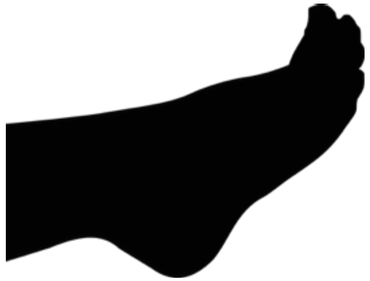
# Which medical speciality does this relate to?

KIDNEY FAILURE, also known as RENAL FAILURE or RENAL INSUFFICIENCY, is a medical condition of impaired kidney function in which the kidneys fail to adequately filter metabolic wastes from the blood.The two main forms are acute kidney injury, which is often reversible with adequate treatment, and chronic kidney disease, which is often not reversible. In both cases, there is usually an underlying cause.



Nephrology

Neurology



Podiatry

# What is Classification?

Given a set of classes:

Classification: Assign the correct class label to the given input

# Examples of Text Classification

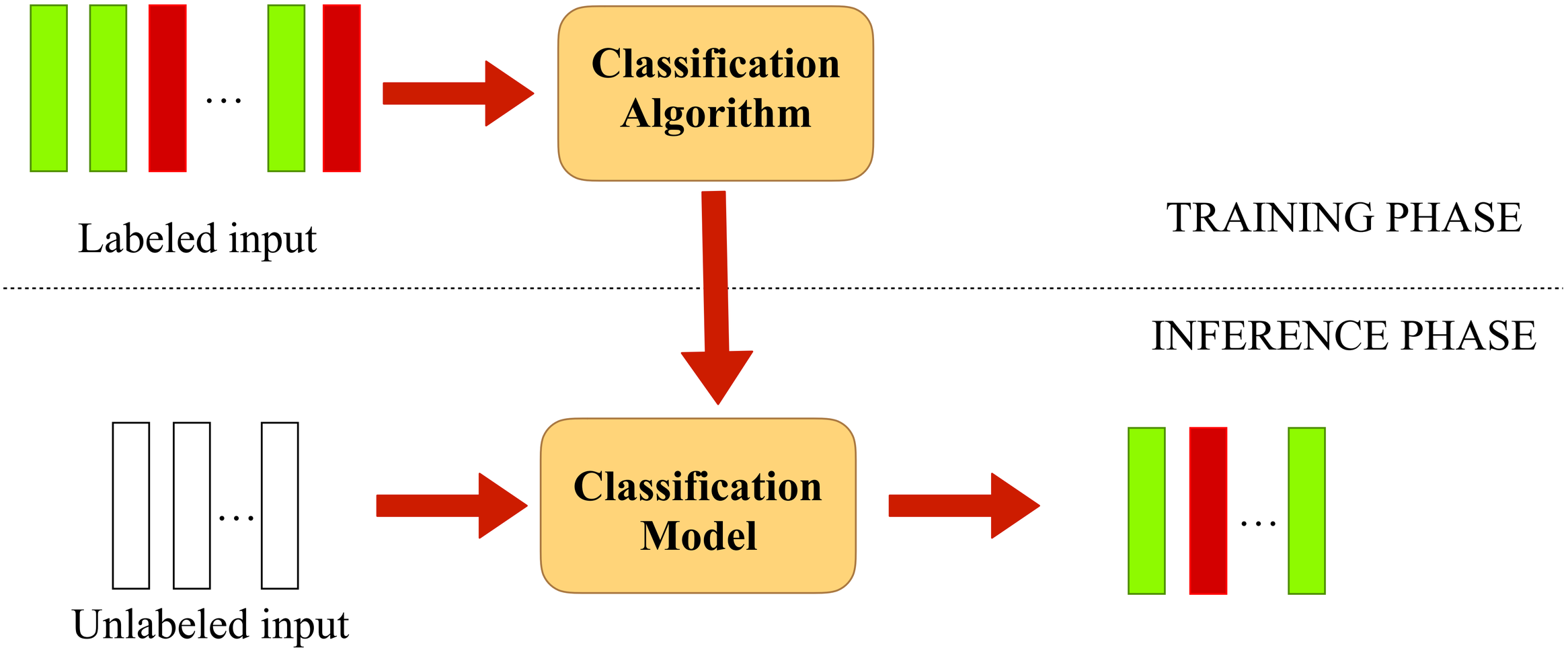
- **Topic identification**: Is this news article about Politics, Sports, or Technology?
- **Spam Detection**: Is this email a spam or not?
- **Sentiment analysis**: Is this movie review positive or negative?
- **Spelling correction**: weather or whether?

  color or colour?

# Supervised Learning

- Humans learn from past experiences, machines learn from past instances!

# Supervised Classification

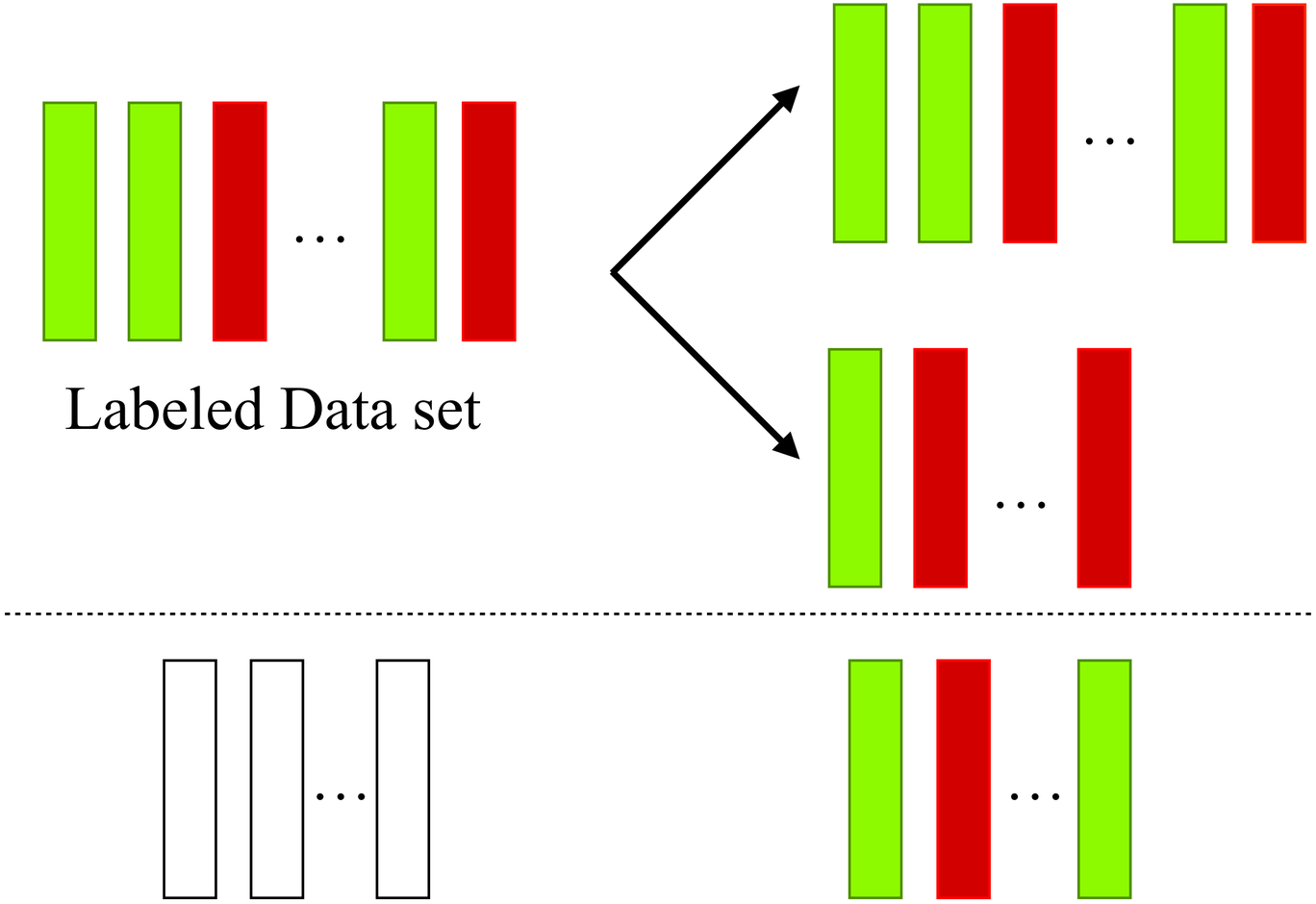


# Supervised Classification

- Learn a classification model on properties ("features") and their importance ("weights") from labeled instances
  - X: Set of attributes or features $\{x_1, x_2, \ldots, x_n\}$
  - y: A "class" label from the label set $Y = \{y_1, y_2, \ldots, y_k\}$

- Apply the model on new instances to predict the label

# Supervised Classification: Phases and Datasets



# Classification Paradigms

- When there are only two possible classes; $|Y| = 2$ :

  Binary Classification

- When there are more than two possible classes; $|Y| > 2$ :

  Multi-class Classification

- When data instances can have two or more labels :

  Multi-label Classification

# Questions to ask in Supervised Learning

- Training phase:
  - What are the features? How do you represent them?
  - What is the classification model / algorithm?
  - What are the model parameters?
- Inference phase:
  - What is the expected performance? What is a good measure?

# Applied Text Mining in Python

*Identifying Features from Text*

# Why is Textual Data Unique?

- Textual data presents a unique set of challenges
- All the information you need is in the text
- But features can be pulled out from text at different granularities!

# Types of Textual Features (1)

- Words
  - By far the most common class of features
  - Handling commonly-occurring words: Stop words
  - Normalization: Make lower case vs. leave as-is
  - Stemming / Lemmatization

# Types of Textual Features (2)

- Characteristics of words : Capitalization
- Parts of speech of words in a sentence
- Grammatical structure, sentence parsing
- Grouping words of similar meaning, semantics
  - {buy, purchase}
  - {Mr., Ms., Dr., Prof.}; Numbers / Digits; Dates

# Types of Textual Features (3)

- Depending on classification tasks, features may come from inside words and word sequences
  - bigrams, trigrams, n-grams: “White House”
  - character sub-sequences in words: “ing”, “ion”, …

# How would you do it?

- Recall lectures from previous week

# Applied Text Mining in Python

*Naïve Bayes Classifier*

# Case study: Classifying text search queries

- Suppose you are interested in classifying search queries in three classes: Entertainment, Computer Science, Zoology
- Most common class of the three is Entertainment.

# Case study: Classifying text search queries

- Suppose the query is "Python"
  - Python, the snake (Zoology)
  - Python, the programming language (Computer Science)
  - Python, as in Monty Python (Entertainment)

- Most common class, given "Python", is Zoology.

# Case study: Classifying text search queries

- Suppose the query is "Python download"
- Most probable class, given "Python download", is Computer Science.

# Probabilistic Model

- Update the likelihood of the class given new information
- Prior Probability: Pr(y = Entertainment), Pr(y = CS), Pr(y=Zoology)
- Posterior probability: Pr(y = Entertainment|x = "Python")

# Bayes' Rule

- $$\text{Posterior probability} = \frac{\text{Prior probability x Likelihood}}{\text{Evidence}}$$

- $$\Pr(y \mid X) = \frac{\Pr(y) \times \Pr(X \mid y)}{\Pr(X)}$$

# Naïve Bayes Classification

- $$\Pr(y=\text{CS} \mid \text{“Python”}) = \frac{\Pr(y=\text{CS}) \times \Pr(\text{“Python”} \mid y=\text{CS})}{\Pr(\text{“Python”})}$$

- $$\Pr(y=\text{Zoology} \mid \text{“Python”}) = \frac{\Pr(y=\text{Zoology}) \times \Pr(\text{“Python”} \mid y=\text{Zoology})}{\Pr(\text{“Python”})}$$

- $\Pr(y=\text{CS} \mid \text{“Python”}) > \Pr(y=\text{Zoology} \mid \text{“Python”})$ ⇨

# Naïve Bayes Classification

- $$\Pr(y \mid X) = \frac{\Pr(y) \times \Pr(X \mid y)}{\Pr(X)}$$

$$y^* = \underset{y}{\operatorname{argmax}} \Pr(y \mid X) = \underset{y}{\operatorname{argmax}} \Pr(y) \times \Pr(X \mid y)$$

- Naïve assumption: Given the class label, features are assumed to be independent of each other

$$y^* = \underset{y}{\operatorname{argmax}} \Pr(y \mid X) = \underset{y}{\operatorname{argmax}} \Pr(y) \times \prod_{i=1}^{n} \Pr(x_i \mid y)$$

# Naïve Bayes Classifier

$$y^* = \underset{y}{\operatorname{argmax}} \Pr(y \mid X) = \underset{y}{\operatorname{argmax}} \Pr(y) \times \prod_{i=1}^{n} \Pr(x_i \mid y)$$

Query: "Python download"

$$y^* = \underset{y}{\operatorname{argmax}} \Pr(y) \times \Pr(\text{“Python”} \mid y) \times \Pr(\text{“download”} \mid y)$$

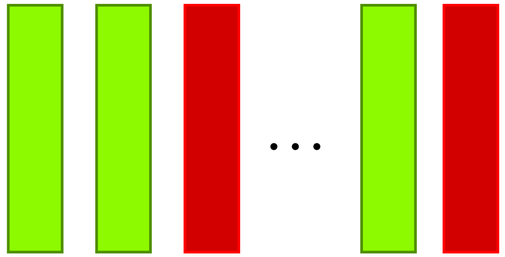
# Naïve Bayes: What are the parameters?

- Prior probabilities: Pr($y$) for all $y$ in $Y$
- Likelihood: Pr($x_i$ | $y$) for all features $x_i$ and labels $y$ in $Y$

- If there are 3 classes (|$Y$| = 3) and 100 features in $X$, how many parameters does naïve Bayes models have?

# Naïve Bayes: Learning parameters

- Prior probabilities: Pr(y) for all y in Y
  - Remember training data?




  - Count the number of instances in each class
  - If there are N instances in all, and n out of those are labeled as class y ⇨ Pr

# Naïve Bayes: Learning parameters

- Likelihood: $Pr(x_i \mid y)$ for all features $x_i$ and labels $y$ in $Y$
  - Count how many times feature $x_i$ appears in instances labeled as class $y$
  - If there are p instances of class $y$, and $x_i$ appears in k of those, $Pr(x_i \mid y) = k / p$

# Naïve Bayes: Smoothing

- What happens if $Pr(x_i \mid y) = 0$?
  - Feature $x_i$ never occurs in documents labeled $y$
  - But then, the posterior probability $Pr(y \mid x_i)$ will be 0!!
- Instead, smooth the parameters
- Laplace smoothing or Additive smoothing: Add a dummy count
  - $Pr(x_i \mid y) = (k+1) / (p+n)$; where n is number of features

# Take Home Concepts

- Naïve Bayes is a probabilistic model
- Naïve, because it assumes features are independent of each other, given the class label
- For text classification problems, naïve Bayes models typically provide very strong baselines
- Simple model, easy to learn parameters

# Applied Text Mining in Python

## *Naïve Bayes Variations*

# Two Classic Naïve Bayes Variants for Text

- Multinomial Naïve Bayes
  - Data follows a multinomial distribution
  - Each feature value is a count (word occurrence counts, TF-IDF weighting, …)

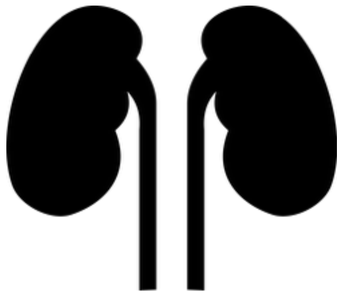
- Bernoulli Naïve Bayes
  - Data follows a multivariate Bernoulli distribution
  - Each feature is binary (word is present / absent)

# Applied Text Mining in Python

*Support Vector Machines*

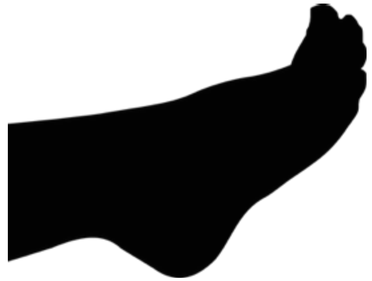
# Which medical speciality does this relate to?

TINEA PEDIS, or ATHLETE'S FOOT, is a very common fungal skin infection of the foot. It often first appears between the toes. It can be a one-time occurrence or it can be chronic. The fungus, known as Trichophyton, thrives under warm, damp conditions so people whose feet sweat a great deal are more susceptible. It is easily transmitted in showers and pool walkways. Those people with immunosuppressive conditions, such as diabetes mellitus, are also more susceptible to athlete's foot.



Nephrology

Neurology



Podiatry

# Case study: Sentiment analysis

**one word: wow.**

★★★★★★★★★★

**Author:** [redacted]

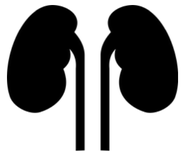
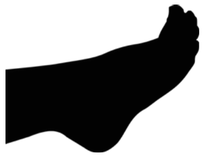
10 January 2005

This is a truly great movie. It is one that I can watch over and over, yet can never seem to get enough of! Kate Winslet is gorgeous, Emma Thomson is inspiring, and Hugh Grant shines in this unforgettable film. I have always loved a good movie; one i can sink into and fall in love with the characters. I feel that Sense and Sensibility presents all these things to the audience. There is love, heartbreak, humour and great music. (my applause to Kate Winslet for her unforgettable versions of "Weep You No More Sad Fountains" and "The Dream" ...so beautiful.) If you have not seen this movie, please go RIGHT now to rent it...or even ADD it to your video library! I must say it is well worth it! And if you have seen it. ....you know what i mean.

Brava! Brava! Bravo!

- Words that you might find in typical reviews
  - wow, great, Bravo!
  - boring, lame, worst

# Classifier = Function on input data

- f ( TINEA PEDIS, or ATHLETE'S FOOT, is a very common fungal skin infection of the foot. It often first appears between the toes. It can be a one-time occurrence or it can be chronic. The fungus, known as Trichophyton, thrives under warm, damp conditions so people whose feet sweat a great deal are more susceptible. It is easily transmitted in showers and pool walkways. Those people with immunosuppressive conditions, such as diabetes mellitus, are also more susceptible to athlete's foot. ) → {  , ,  }

- g( ) → { +1 , -1 }

one word: wow.

Author: [redacted]

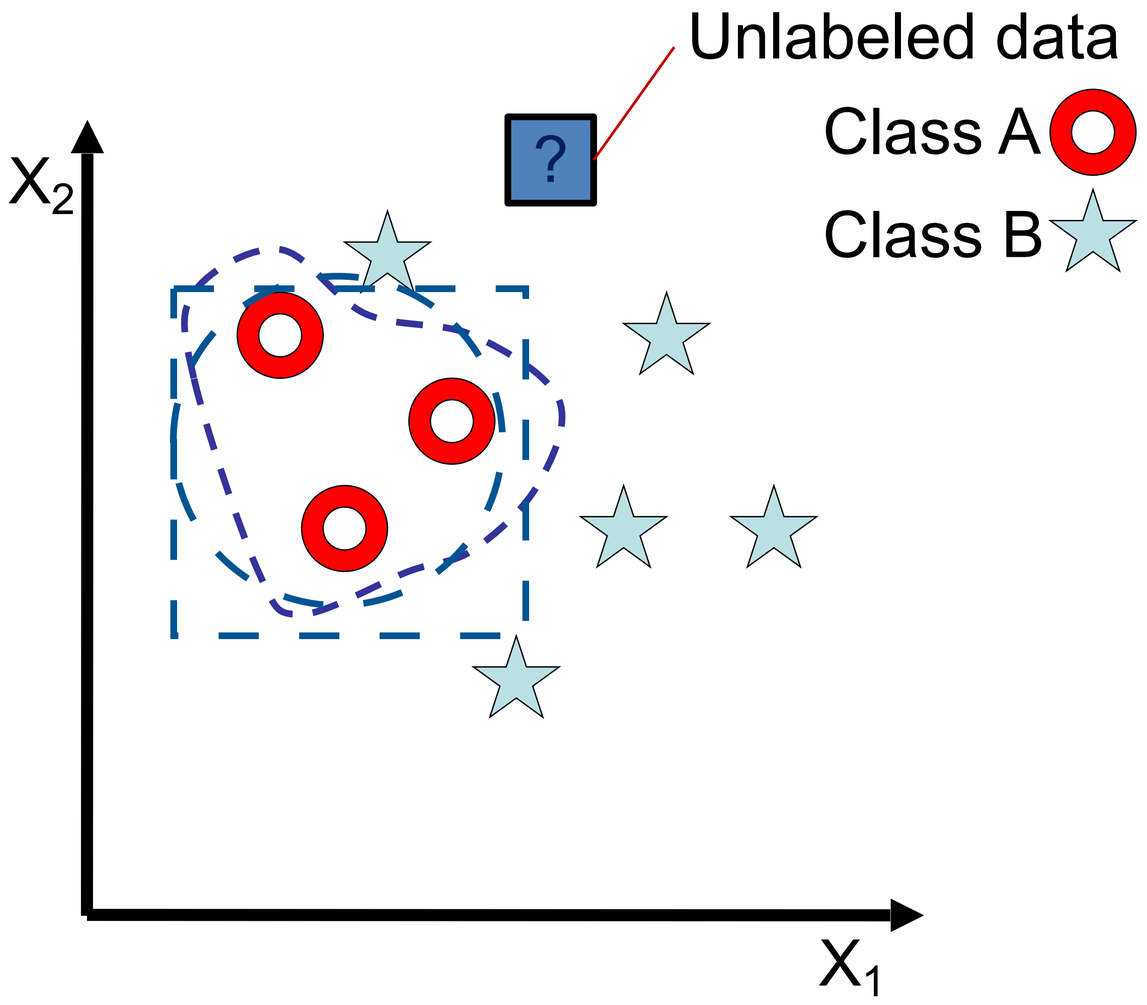
10 January 2005

This is a truly great movie. It is one that I can watch over and over, yet can never seem to get enough of! Kate Winslet is gorgeous, Emma Thomson is inspiring, and Hugh Grant shines in this unforgettable film. I have always loved a good movie; one I can sink into and fall in love with the characters. I feel that Sense and Sensibility presents all these things to the audience. There is love, heartbreak, humour and great music. (my applause to Kate Winslet for her unforgettable versions of "Weep You No More Sad Fountains" and "The Dream" ...so beautiful.) If you have not seen this movie, please go RIGHT now to rent it...or even ADD it to your video library! I must say it is well worth it! And if you have seen it. ...you know what I mean.

Brava! Brava! Bravo!

# Decision Boundaries

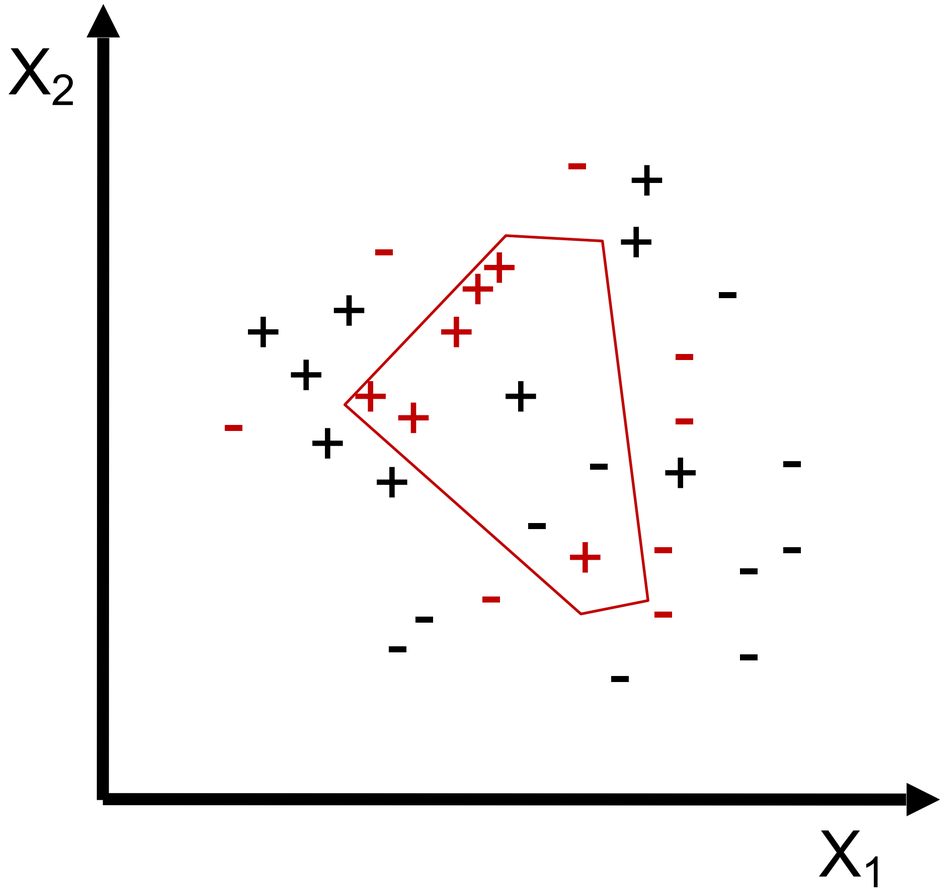
- Classification function is represented by decision surfaces
  - How do you find them?



# Choosing a Decision Boundary

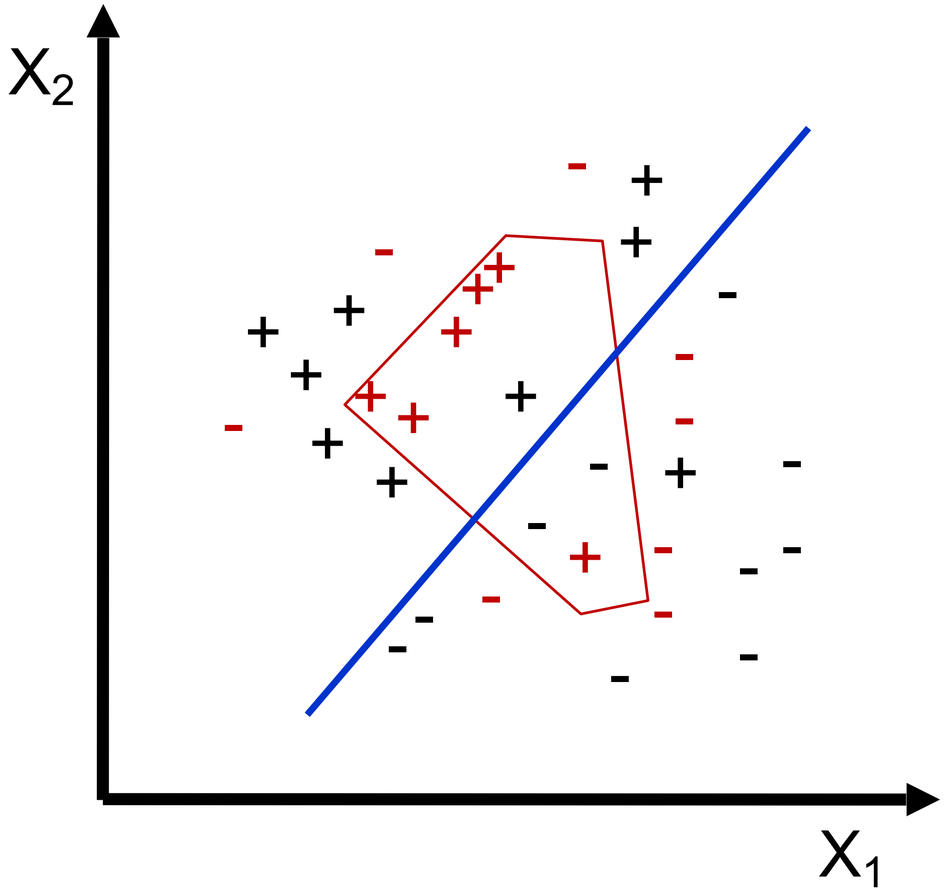
- Red +/- : Training data
- Black +/- : Test data

- Data overfitting: Decision boundary learned over training data doesn't generalize to test data



# Linear Boundaries

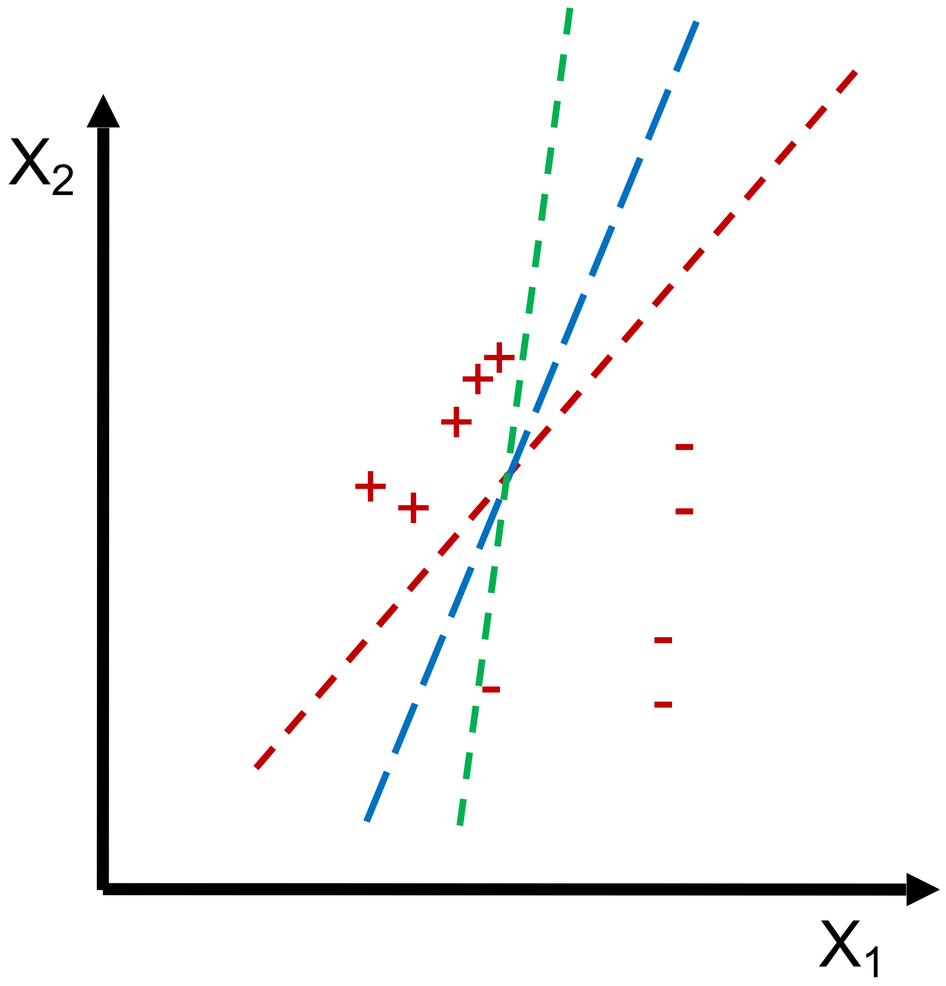
- Easy to find
- Easy to evaluate
- More generalizable: "Occam's razor"



# Finding a Linear Boundary

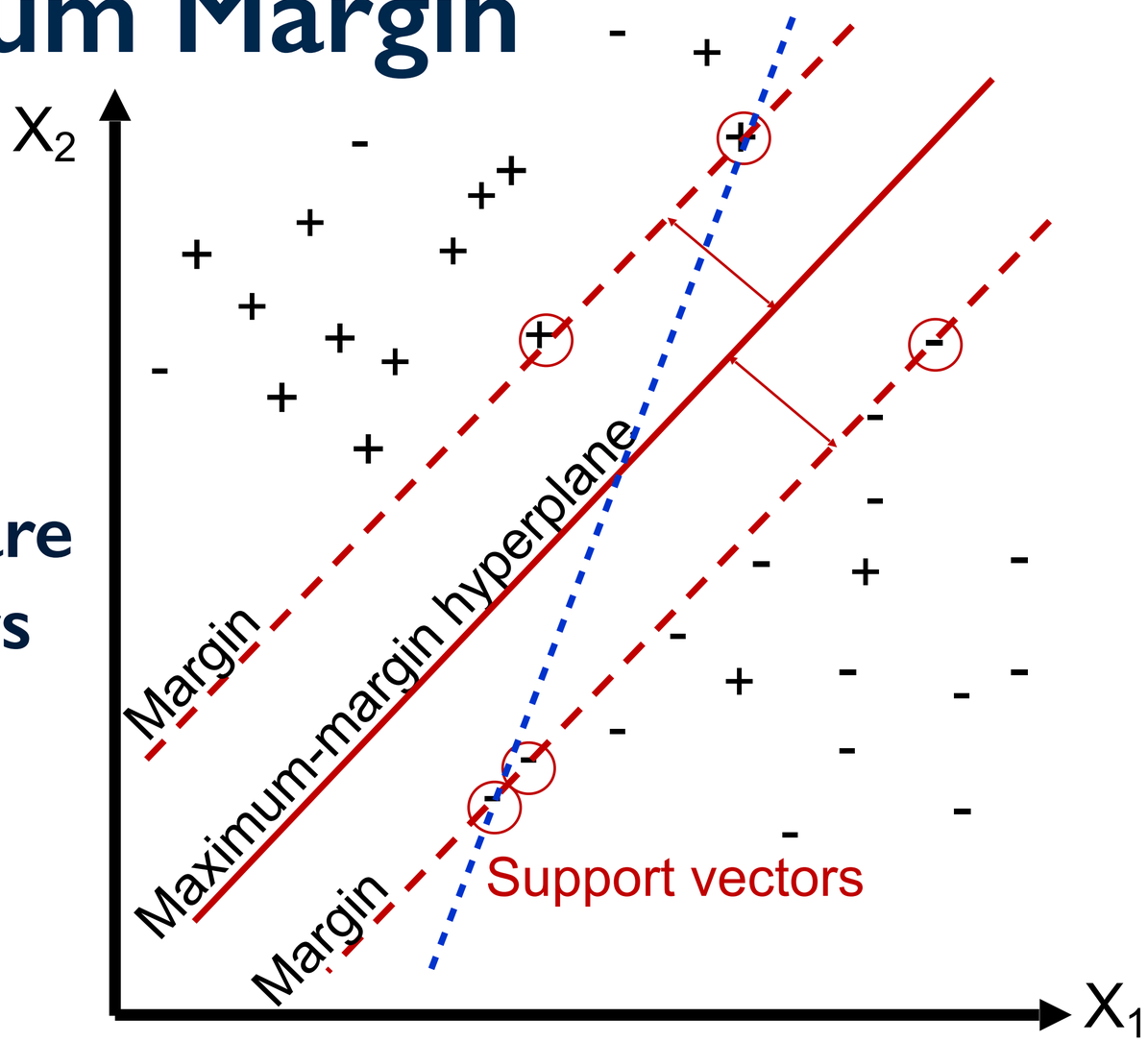
- Find the linear boundary = Find w
- Many methods
  - Perceptron
  - Linear Discriminative Analysis
  - Linear least squares
  - …

- Problem: If linearly separable, then infinite number of linear boundaries!



# Maximum Margin

- What is a reasonable boundary?
- Support Vector Machines are maximum-margin classifiers
- How do you find it?



# Support Vector Machines (SVM)

- SVMs are linear classifiers that find a hyperplane to separate two classes of data: positive and negative
- Given training data $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots$ ; where $\mathbf{x}_i = (x_1, x_2, \ldots, x_n)$ is instance vector and $y_i$ is one of {-1, +1}
- SVM finds a linear function w (weight vector)

$$f(\mathbf{x}_i) = \langle \mathbf{w} \,.\, \mathbf{x}_i \rangle + b$$

$$\text{if } f(\mathbf{x}_i) \geq 0, y_i = +1; \text{ else } y_i = -1$$

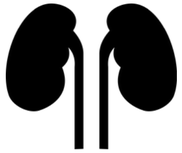
# SVM: Multi-class classification

- SVMs work only for binary classification problems

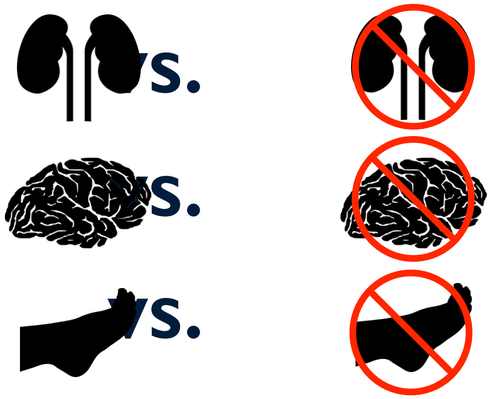
$$f(\mathbf{x}_i) = < \mathbf{w} \,.\, \mathbf{x}_i > + b$$

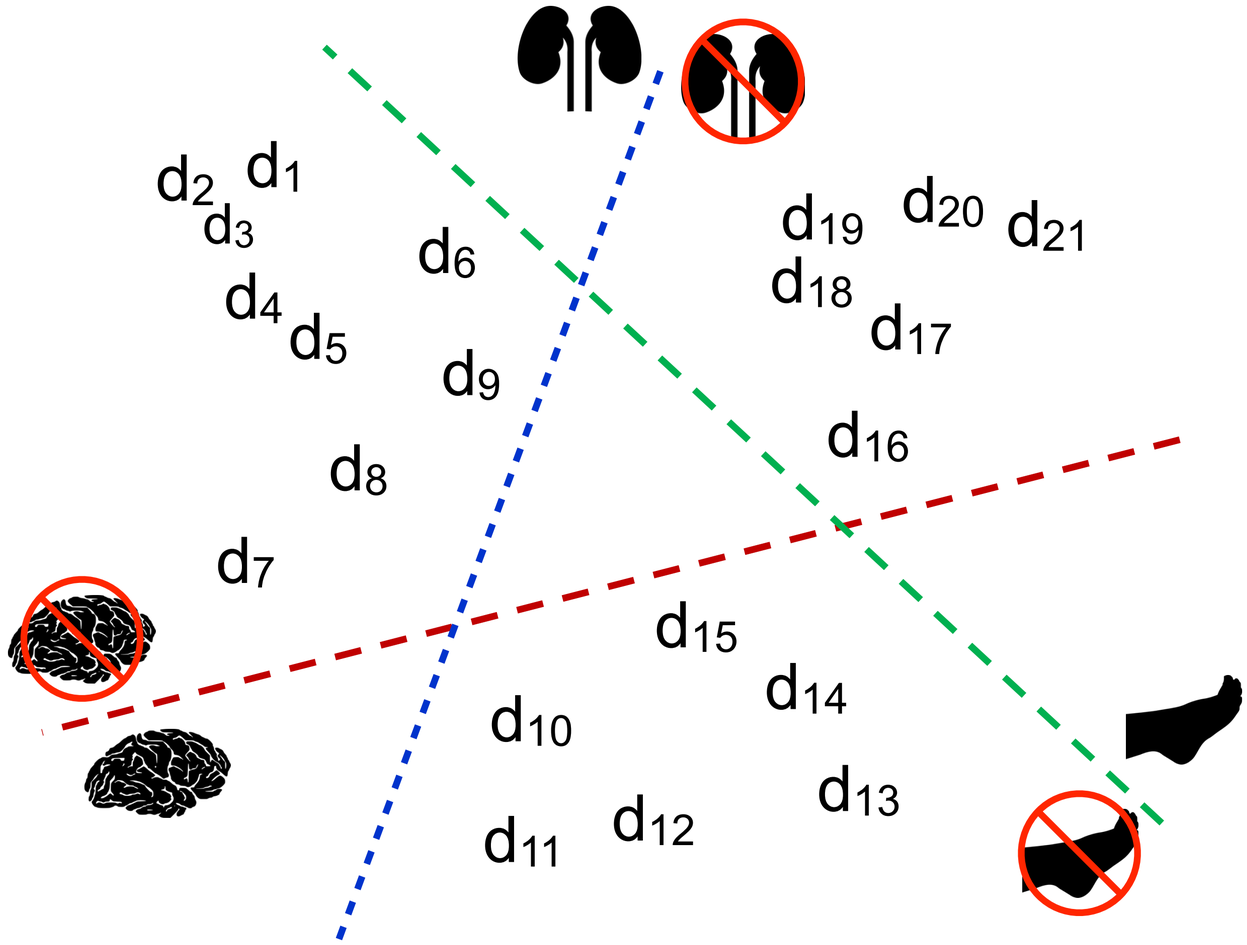
$$\text{if } f(\mathbf{x}_i) \geq 0, y_i = +1; \text{ else } y_i = -1$$

- What about three classes?



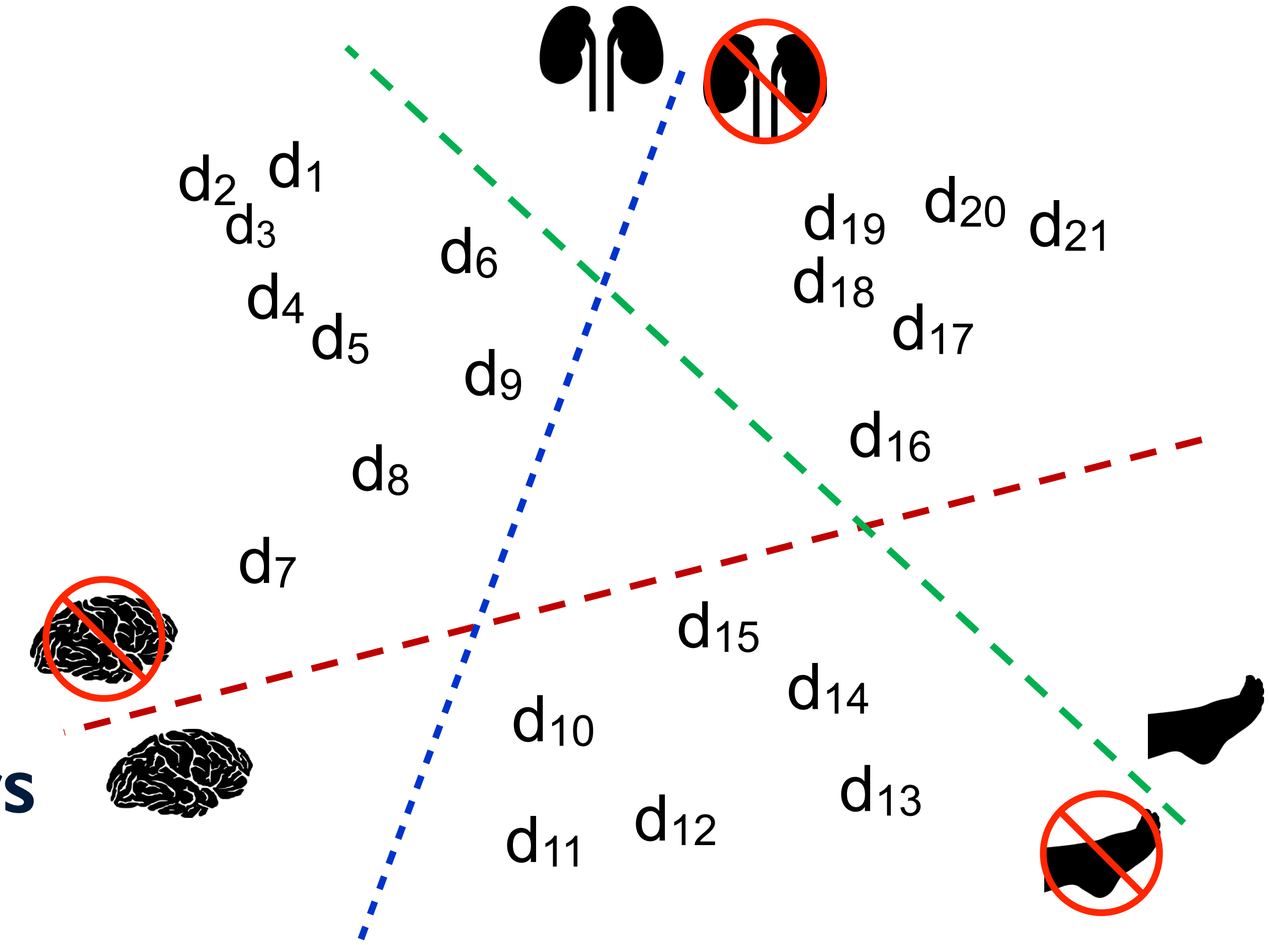
# SVM: Multi-class classification

- One vs Rest
  - 

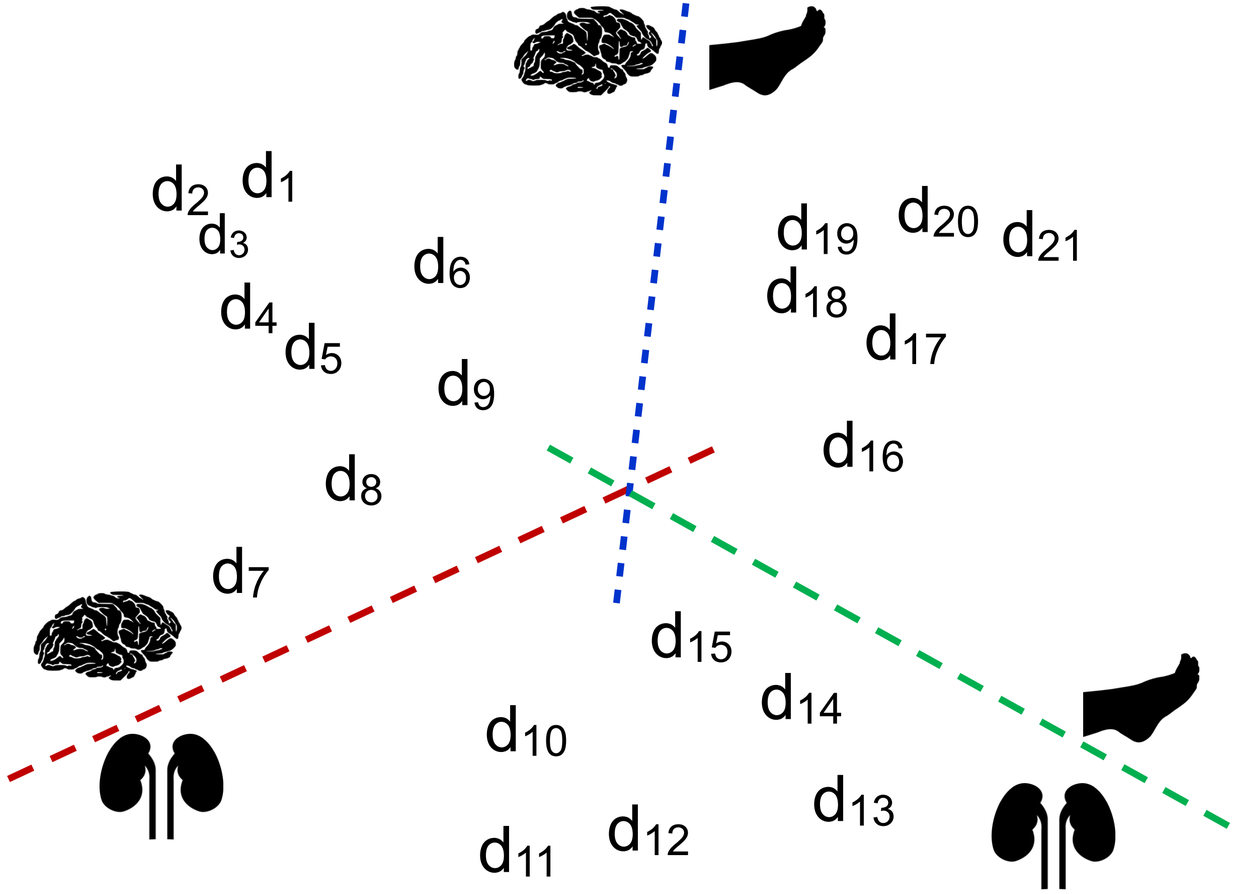
# SVM: Multi-class classification

- One vs Rest
  -
  -
  -
- n-class SVM has n classifiers



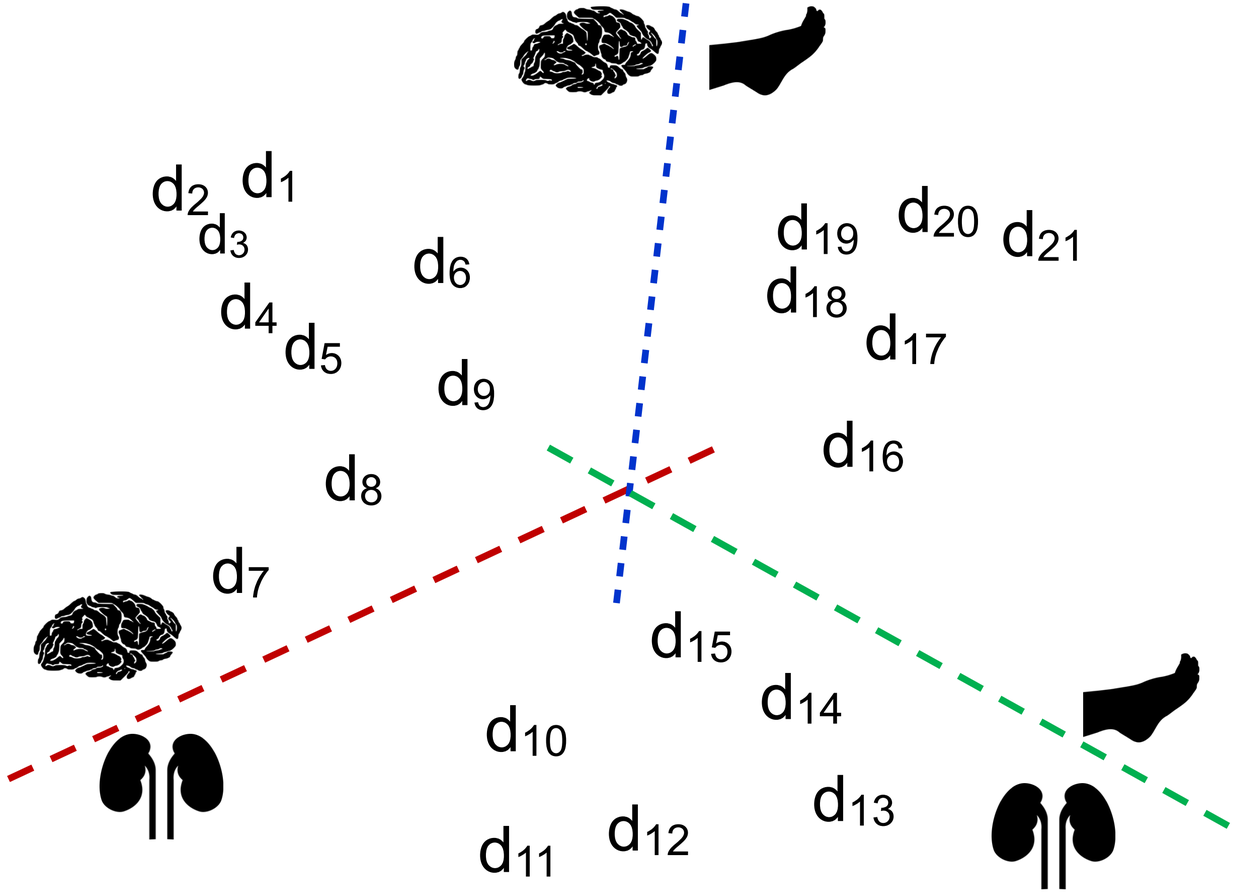
# SVM: Multi-class classification

- One vs One
  - vs.
  - vs.
  - vs.



# SVM: Multi-class classification

- One vs One
  - vs.
  - vs.
  - vs.
- n-class SVM has C(n,2) classifiers



# SVM Parameters (1): Parameter C

- Regularization: How much importance should you give individual data points as compared to better generalized model
- Regularization parameter c
  - Larger values of c = less regularization
    - Fit training data as well as possible, every data point important
  - Smaller values of c = more regularization
    - More tolerant to errors on individual data points

# SVM Parameters (2): Other params

- Linear kernels usually work best for text data
  - Other kernels include rbf, polynomial
- multi_class: ovr (one-vs-rest)
- class_weight: Different classes can get different weights

# Take Home Messages

- Support Vector Machines tend to be the most accurate classifiers, especially in high-dimensional data
- Strong theoretical foundation
- Handles only numeric features
  - Convert categorical features to numeric features
  - Normalization
- Hyperplane hard to interpret

# Applied Text Mining in Python

*Learning Text Classifiers in Python*

# Toolkits for Supervised Text Classification

- Scikit-learn
- NLTK
  - Interfaces with sklearn and other ML toolkits (like Weka)!

# Scikit-learn

- Open-source Machine Learning library
- Started as Google Summer of Code by Dave Cournapeau, 2007
- Has a more programmatic interface

# Using Sklearn's NaiveBayesClassifier

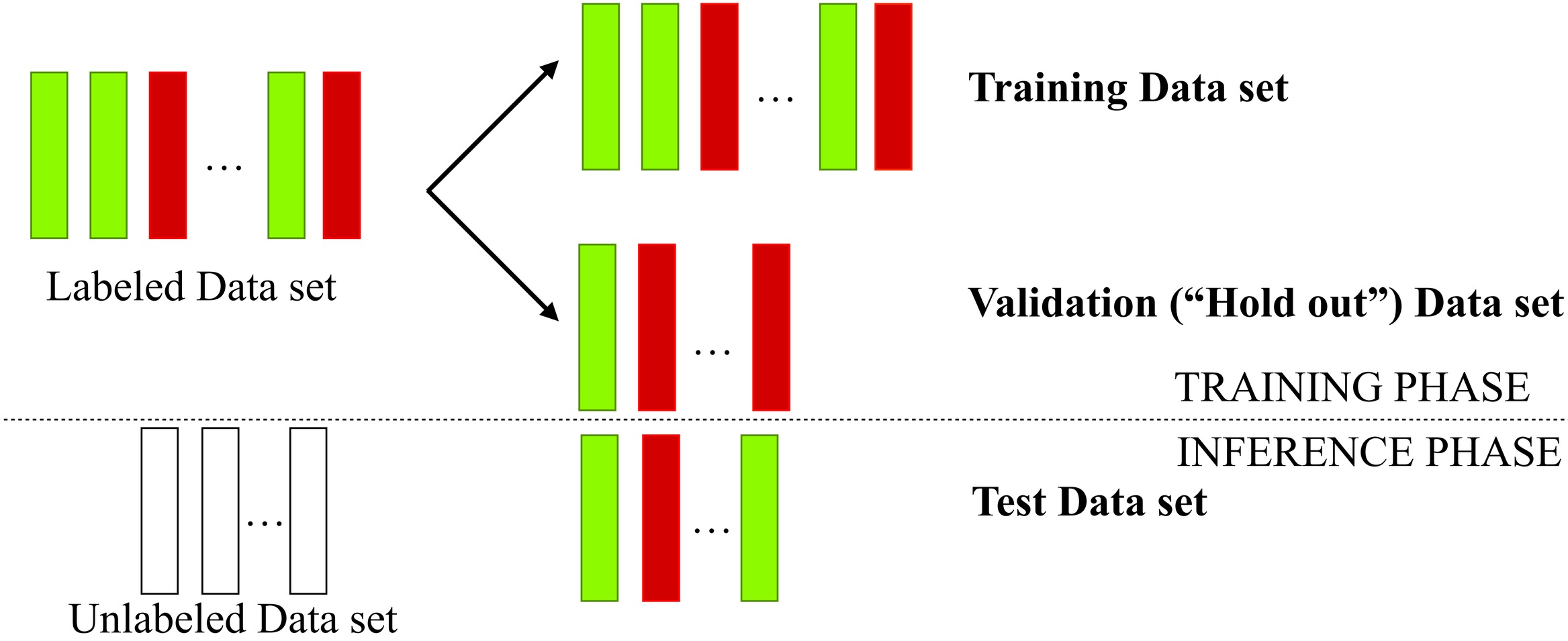
```
from sklearn import naive_bayes

clfrNB = naive_bayes.MultinomialNB()

clfrNB.fit(train_data, train_labels)

predicted_labels = clfrNB.predict(test_data)

metrics.f1_score(test_labels, predicted_labels, average='micro')
```

# Using Sklearn's SVM classifier

```
from sklearn import svm

clfrSVM = svm.SVC(kernel='linear', C=0.1)

clfrSVM.fit(train_data, train_labels)

predicted_labels = clfrSVM.predict(test_data)
```

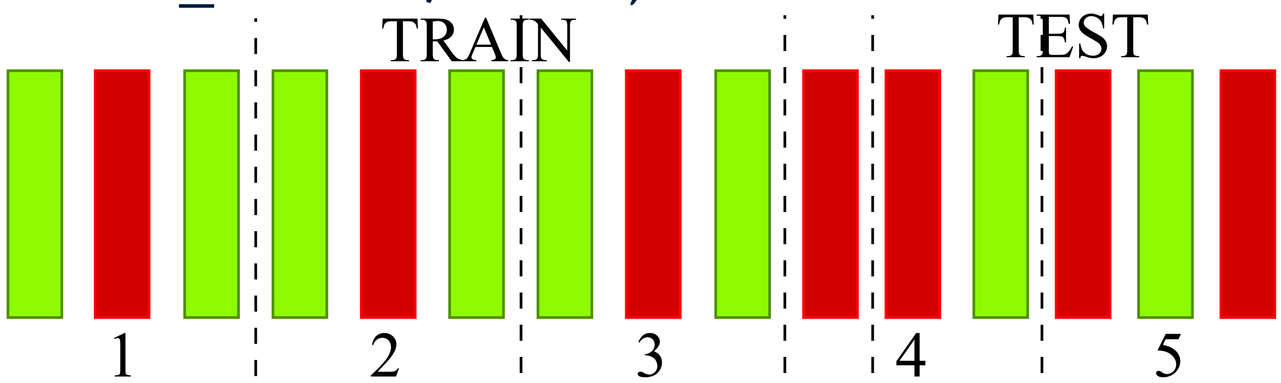
# Model Selection

- **Recall the discussion on multiple phases in a supervised learning task**



# Model selection in Scikit-learn

```
from sklearn import model_selection

X_train, X_test, y_train, y_test =
model_selection.train_test_split(train_data, train_labels,
test_size = 0.333, random_state = 0)

predicted_labels = model_selection.cross_val_predict(clfrSVM,
train_data, train_labels, cv=5)
```



# Supervised Text Classification in NLTK

- NLTK has some classification algorithms
  - NaiveBayesClassifier
  - DecisionTreeClassifier
  - ConditionalExponentialClassifier
  - MaxentClassifier
  - WekaClassifier
  - SklearnClassifier

# Using NLTK's NaiveBayesClassifier

```
from nltk.classify import NaiveBayesClassifier

classifier = NaiveBayesClassifier.train(train_set)

classifier.classify(unlabaled_instance)
classifier.classify_many(unlabeled_instances)

nltk.classify.util.accuracy(classifier, test_set)

classifier.labels()

classifier.show_most_informative_features()
```

# Using NLTK's SklearnClassifier

```
from nltk.classify import SklearnClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC

clfrNB = SklearnClassifier(MultinomialNB()).train(train_set)

clfrSVM =
SklearnClassifier(SVC(),kernel='linear').train(train_set)
```

# Take Home Concepts

- Scikit-learn most commonly used ML toolkit in Python
- NLTK has its own naïve Bayes implementation
- NLTK can also interface with Scikit-learn (and other ML toolkits like Weka)

# Applied Text Mining in Python

*Semantic Text Similarity*

# Which pair of words are most similar?

- deer , elk
- deer , giraffe
- deer , horse
- deer , mouse

# Which pair of words are most similar?

- deer , elk
- deer , horse
- deer , house
- deer , roof

- How can we quantify such similarity?

# Applications of Text Similarity

- Grouping similar words into semantic concepts
- As a building block in natural language understanding tasks
  - Textual entailment
  - Paraphrasing

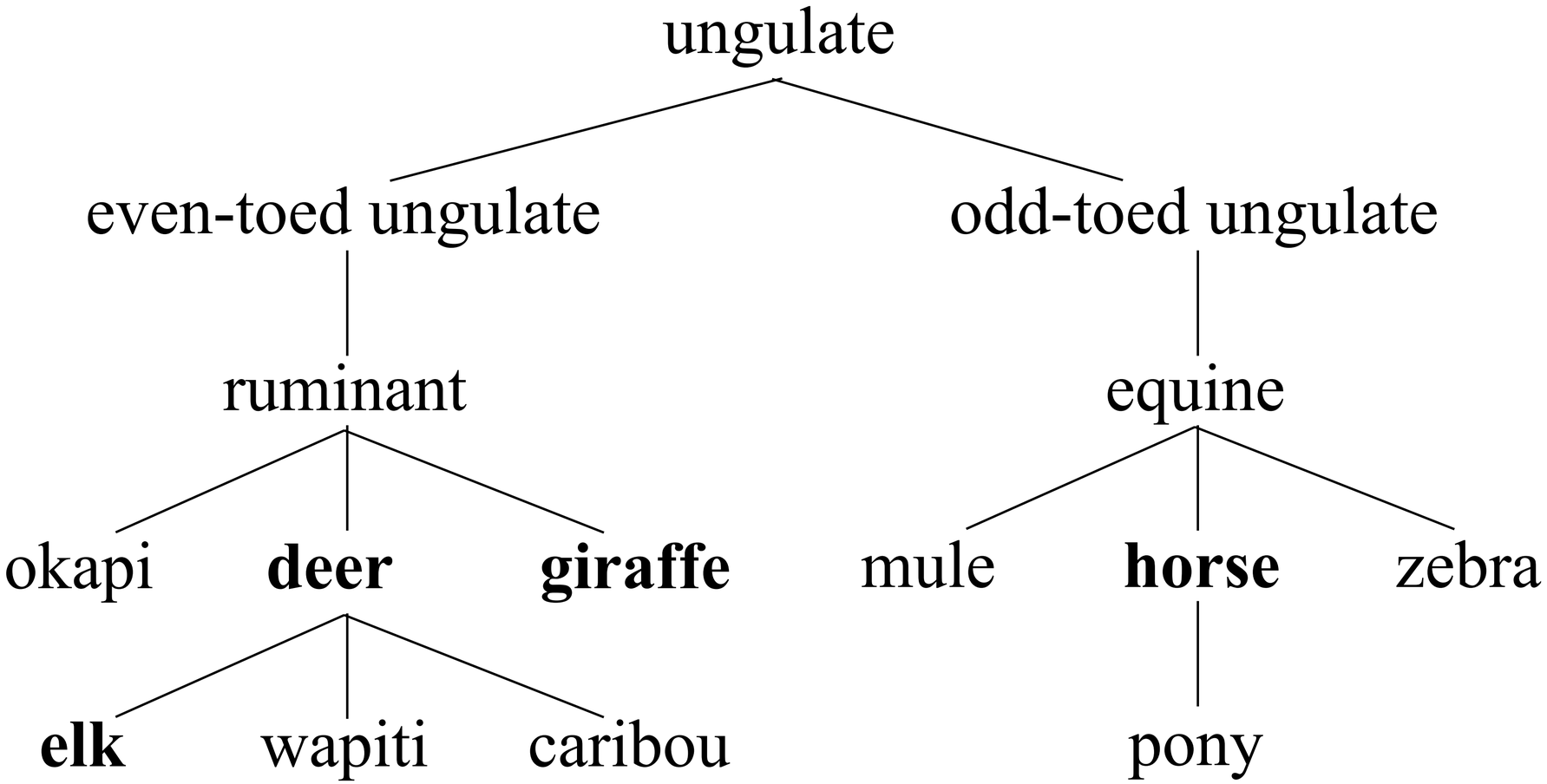
# WordNet

- Semantic dictionary of (mostly) English words, interlinked by semantic relations
- Includes rich linguistic information
  - part of speech, word senses, synonyms, hypernyms/ hyponyms, meronyms, distributional related forms, …
- Machine-readable, freely available

# Semantic Similarity Using WordNet

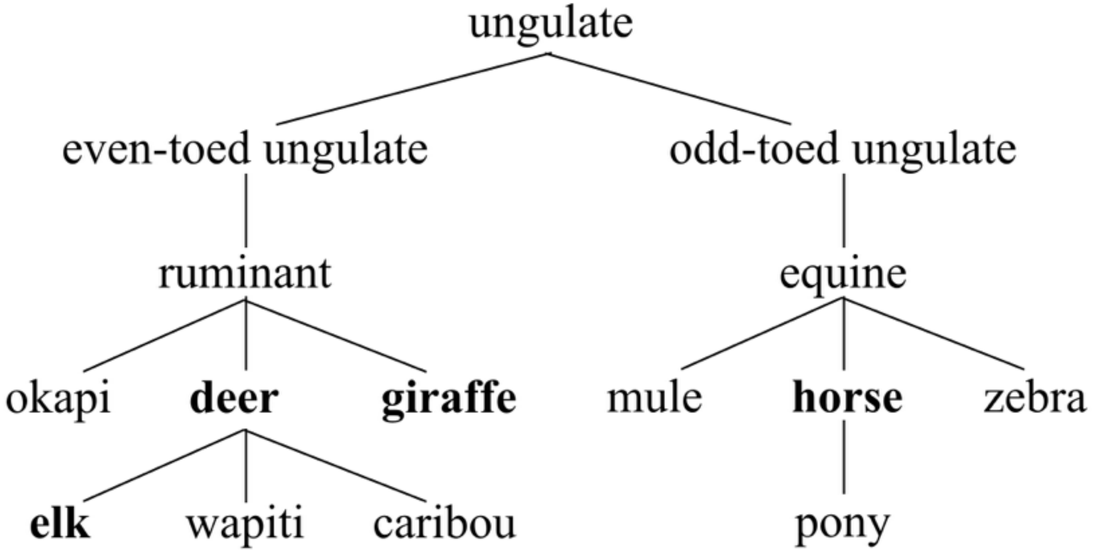
- WordNet organizes information in a hierarchy
- Many similarity measures use the hierarchy in some way
- Verbs, nouns, adjectives all have separate hierarchies

# Coming back to our deer example



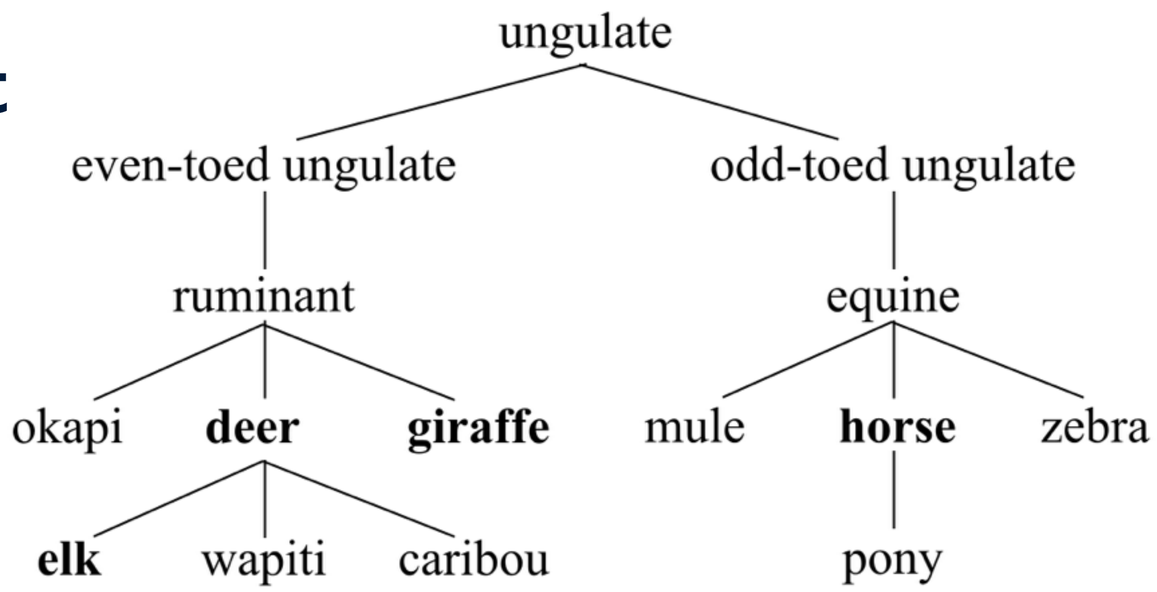
# Path Similarity

- Find the shortest path between the two concepts
- Similarity measure inversely related to path distance
  - PathSim(deer, elk) = 0.5
  - PathSim(deer, giraffe) = 0.33
  - PathSim(deer, horse) = 0.14



# Lowest Common Subsumer (LCS)

- Find the closest ancestor to both concepts
  - LCS(deer, elk) = deer
  - LCS(deer, giraffe) = ruminant
  - LCS(deer, horse) = ungulate



# Lin Similarity

- Similarity measure based on the information contained in the LCS of the two concepts
  - LinSim(u, v) = 2 x log P(LCS(u,v)) / (log P(u) + log P(v))
- P(u) is given by the information content learnt over a large corpus.

# How to do it in Python?

- **WordNet easily imported into Python through NLTK**

```
import nltk
from nltk.corpus import wordnet as wn
```

- **Find appropriate sense of the words**

```
deer = wn.synset('deer.n.01')
elk = wn.synset('elk.n.01')
…
```

# How to do it in Python? (2)

- **Find path similarity**

```
deer.path_similarity(elk)        0.5
deer.path_similarity(horse)      0.14285714285714285
```

- **Use an information criteria to find Lin similarity**

```
from nltk.corpus import wordnet_ic
brown_ic = wordnet_ic.ic('ic-brown.dat')

deer.lin_similarity(elk, brown_ic)        0.7726998936065773
deer.lin_similarity(horse, brown_ic)      0.8623778273893673
```

# Collocations and Distributional Similarity

- "You know a word by the company it keeps" [Firth, 1957]
- Two words that frequently appears in similar contexts are more likely to be semantically related
  - The friends met at a café.
  - Shyam met Ray at a pizzeria.
  - Let's meet up near the coffee shop.
  - The secret meeting at the restaurant soon became public.

# Distributional Similarity: Context

- Words before, after, within a small window
- Parts of speech of words before, after, in a small window
- Specific syntactic relation to the target word
- Words in the same sentence, same document, …

# Strength of association between words

- How frequent are these?
  - Not similar if two words don't occur together often
- Also important to see how frequent are individual words
  - 'the' is very frequent, so high chances it co-occurs often with every other word
- Pointwise Mutual Information $PMI(w,c) = \log [P(w,c) / P(w)P(c)]$

# How to do it in Python?

- **Use NLTK Collocations and Association measures**

```
import nltk
from nltk.collocations import *

bigram_measures = nltk.collocations.BigramAssocMeasures()

finder = BigramCollocationFinder.from_words(text)
finder.nbest(bigram_measures.pmi, 10)
```

- **finder also has other useful functions, such as frequency filter**

```
finder.apply_freq_filter(10)
```

# Take Home Concepts

- Finding similarity between words and text is non-trivial
- WordNet is a useful resource for semantic relationships between words
- Many similarity functions exist
- NLTK is a useful package for many such tasks

# Applied Text Mining in Python

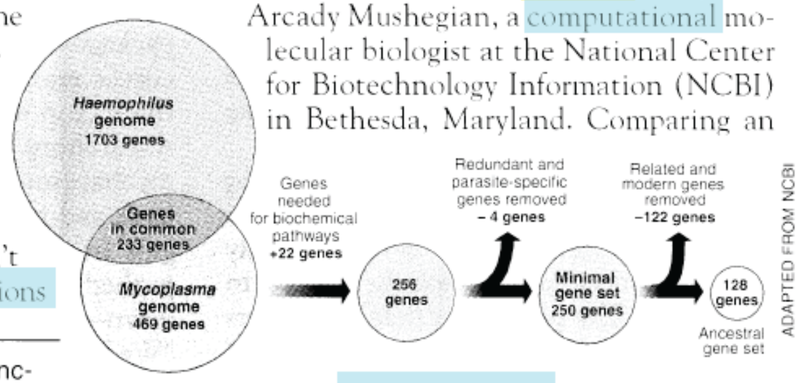
*Topic modeling*

# Documents Exhibit Multiple Topics

## Seeking Life's Bare (Genetic) Necessities

COLD SPRING HARBOR, NEW YORK—How many genes does an organism need to survive? Last week at the genome meeting here,* two genome researchers with radically different approaches presented complementary views of the basic genes needed for life. One research team, using computer analyses to compare known genomes, concluded that today's organisms can be sustained with just 250 genes, and that the earliest life forms required a mere 128 genes. The other researcher mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough.

Although the numbers don't match precisely, those predictions "are not all that far apart," especially in comparison to the 75,000 genes in the human genome, notes Siv Andersson of Uppsala University in Sweden, who arrived at the 800 number. But coming up with a consensus answer may be more than just a genetic numbers game, particularly as more and more genomes are completely mapped and sequenced. "It may be a way of organizing any newly sequenced genome," explains Arcady Mushegian, a computational molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing an




* Genome Mapping and Sequencing, Cold Spring Harbor, New York, May 8 to 12.

**Stripping down.** Computer analysis yields an estimate of the minimum modern and ancient genomes.

SCIENCE • VOL. 272 • 24 MAY 1996

(Figure courtesy Prof. David Blei)

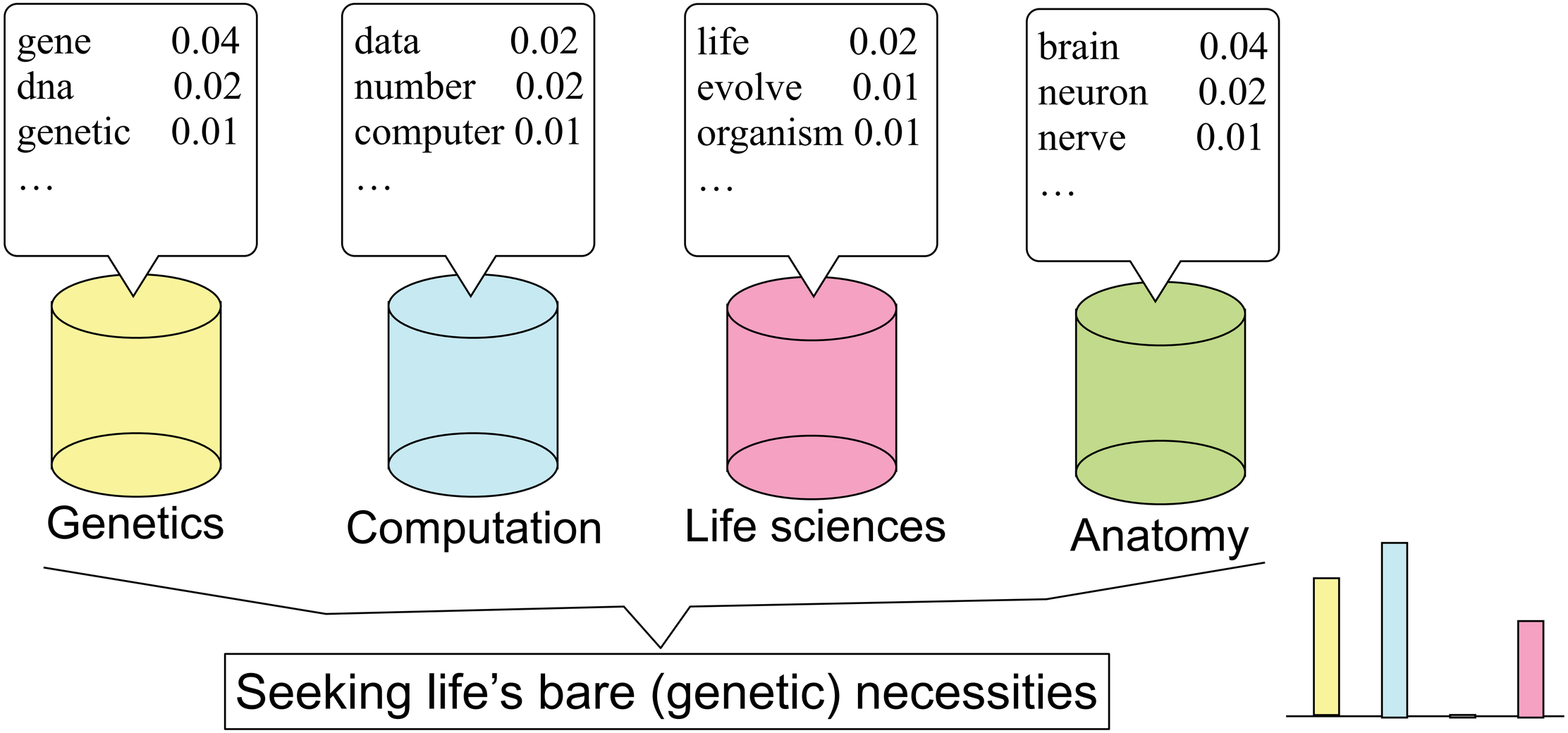
Latent Dirichlet Allocation
(Blei et al., '03)

**Topic 1: Genetics**
gene, sequence, genome, …

**Topic 2: Computation**
number, computer, analysis, …

**Topic 3: Life sciences**
life, survive, organism, …

# Intuition: Documents as a mixture of topics



# What is Topic Modeling?

- A course-level analysis of what's in a text collection
- Topic : the subject (theme) of a discourse
- Topics are represented as a word distribution
- A document is assumed to be a mixture of topics

# More examples of topics

| | | | |
|---|---|---|---|
| human | evolution | disease | computer |
| genome | evolutionary | host | models |
| dna | species | bacteria | information |
| genetic | organisms | diseases | data |
| genes | life | resistance | computers |
| sequence | origin | bacterial | system |
| gene | biology | new | network |
| molecular | groups | strains | systems |
| sequencing | phylogenetic | control | model |
| map | living | infectious | parallel |
| information | diversity | malaria | methods |
| genetics | group | parasite | networks |
| mapping | new | parasites | software |
| project | two | united | new |
| sequences | common | tuberculosis | simulations |

(Figure courtesy Prof. David Blei)

# What is Topic Modeling? (2)

- What's known:
  - The text collection or corpus
  - Number of topics
- What's not known:
  - The actual topics
  - Topic distribution for each document

# What is Topic Modeling? (3)

- Essentially, text clustering problem
  - Documents and words clustered simultaneously

- Different topic modeling approaches available
  - Probabilistic Latent Semantic Analysis (PLSA) [Hoffman '99]
  - Latent Dirichlet Allocation (LDA) [Blei, Ng, and Jordan, '03]

# Applied Text Mining in Python

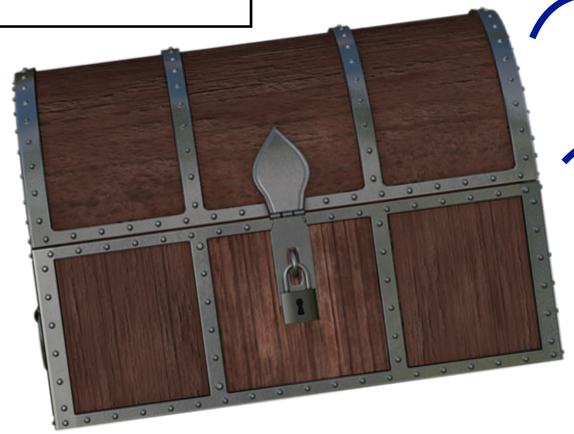
*Generative models and LDA*

# Generative Models for Text

Pr(text | model)

| | |
|---|---|
| **the** | 0.1 |
| **is** | 0.07 |
| **harry** | 0.05 |
| **potter** | 0.04 |
| **movie** | 0.04 |
| **plot** | 0.02 |
| **time** | 0.01 |
| **rowling** | 0.01 |

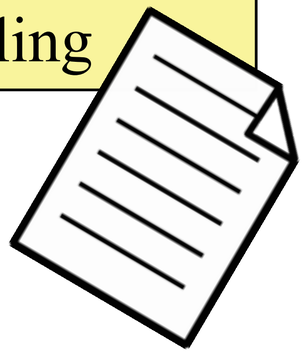
**Inference, Estimation**



the harry potter movie is harry is

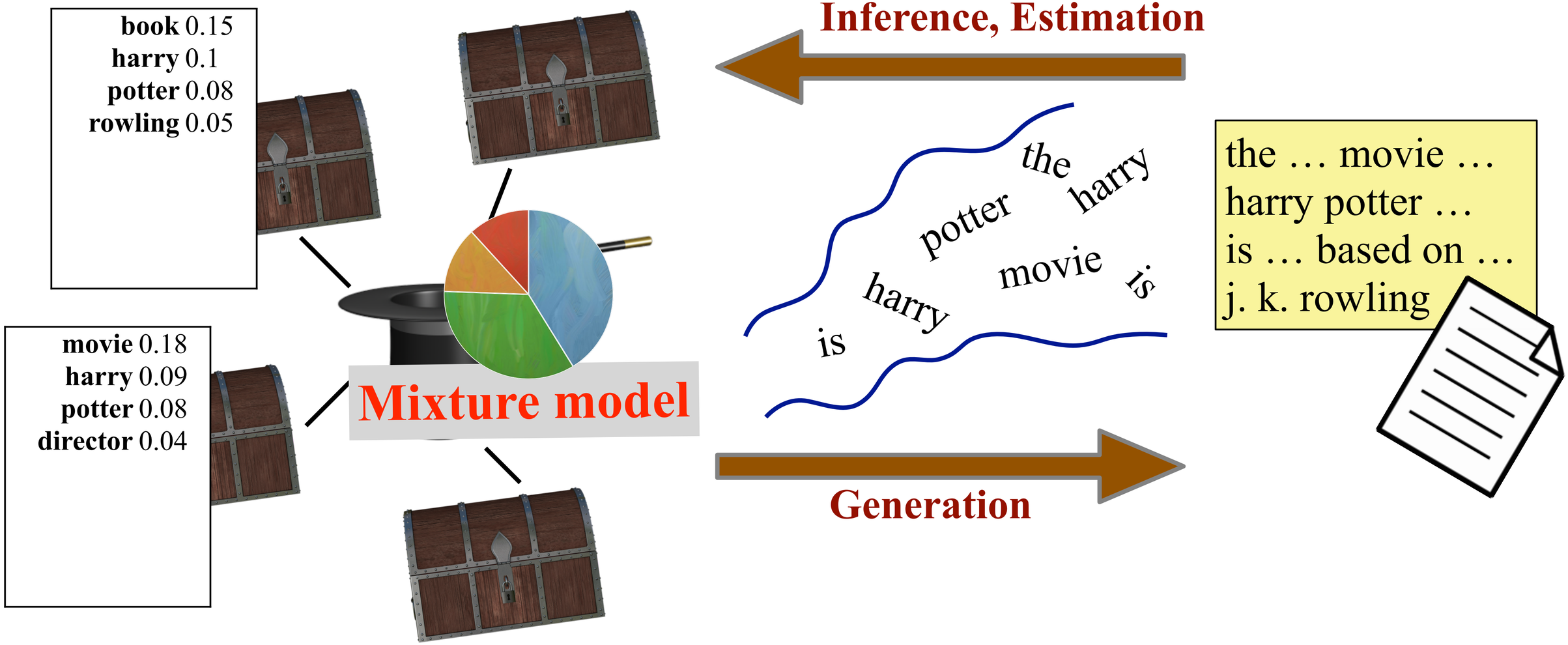
**Generation**

the … movie …
harry potter …
is … based on …
j. k. rowling



# Generative Models can be Complex

Pr(text | model)



# Latent Dirichlet Allocation (LDA)

- Generative model for a document d
  - Choose length of document d
  - Choose a mixture of topics for document d
  - Use a topic's multinomial distribution to output words to fill that topic's quota

# Topic Modeling in Practice

- How many topics?
  - Finding or even guessing the number of topics is hard
- Interpreting topics
  - Topics are just word distributions
  - Making sense of words / generating labels is subjective

# Topic Modeling: Summary

- Great tool for exploratory text analysis
  - What are the documents (tweets, reviews, news articles) about?
- Many tools available to do it effortlessly in Python

# Working with LDA in Python

- Many packages available, such as gensim, lda
- Pre-processing text
  - Tokenize, normalize (lowercase)
  - Stop word removal
  - Stemming
- Convert tokenized documents to a document - term matrix
- Build LDA models on the doc-term matrix

# Working with LDA in Python (2)

- **doc_set: set of pre-processed text documents**

```
import gensim
from gensim import corpora, models

dictionary = corpora.Dictionary(doc_set)
corpus = [dictionary.doc2bow(doc) for doc in doc_set]
ldamodel = gensim.models.ldamodel.LdaModel (corpus, num_topics=4,
id2word=dictionary, passes=50)
print(ldamodel.print_topics(num_topics=4, num_words=5))
```

- **ldamodel can also be used to find topic distribution of documents**

```
topic_dis = ldamodel[new_doc]
```

# Take Home Concepts

- Topic modeling is an exploratory tool frequently used for text mining
- Latent Dirichlet Allocation is a generative model used extensively for modeling large text corpora
- LDA can also be used as a feature selection technique for text classification and other tasks

# Applied Text Mining in Python

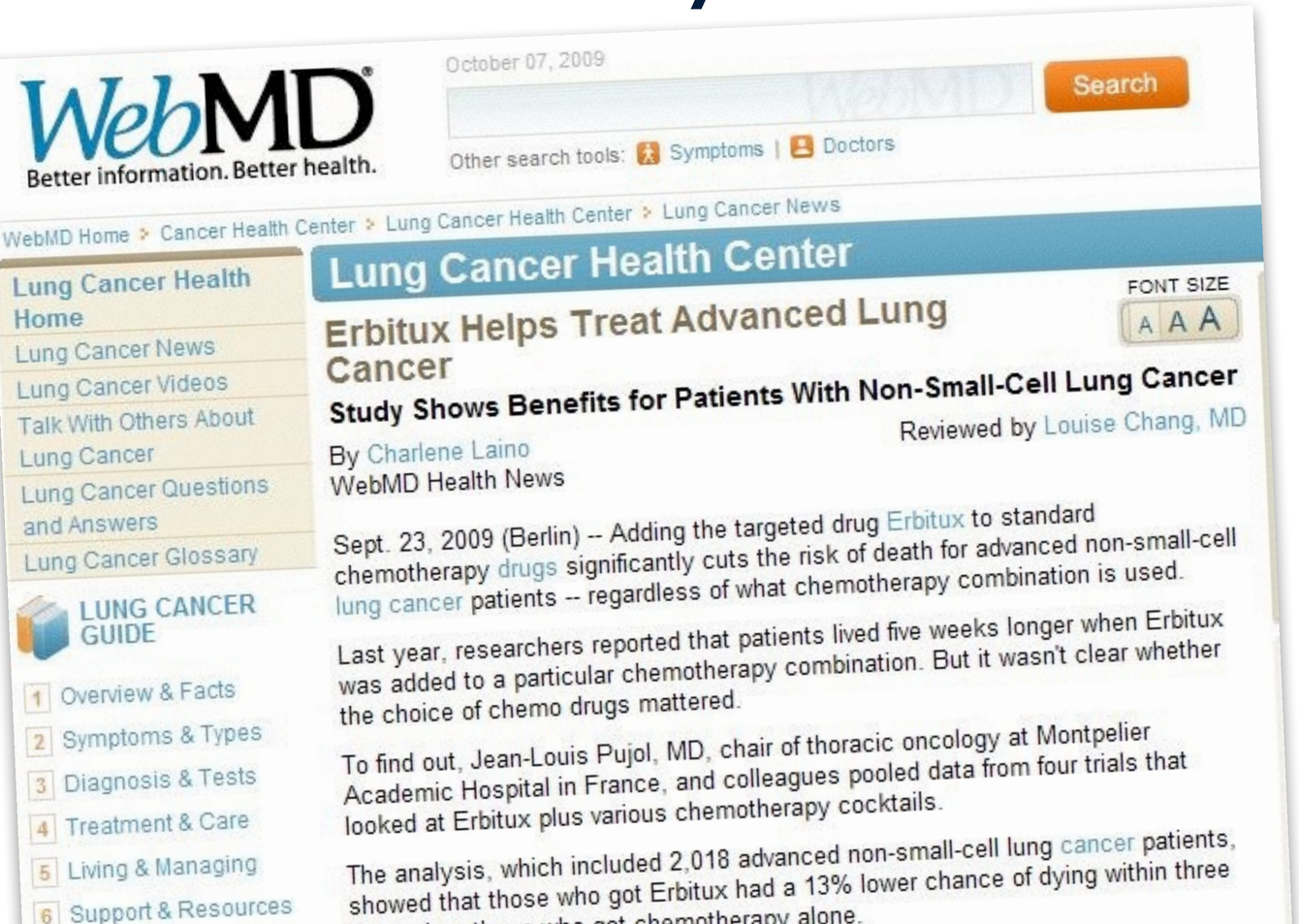
*Information Extraction*

# Information is hidden in free-text

- Most traditional transactional information is structured
- Abundance of unstructured, freeform text
- How to convert unstructured text to structured form?

# Information Extraction

- **Goal: Identify and extract fields of interest from free text**




Erbitux helps treat lung cancer

Author: Charlene Laino

Reviewer: Louise Chang, MD

Sept. 23, 2009

Berlin

…

# Fields of Interest

- Named entities
  - [NEWS] People, Places, Dates, …
  - [FINANCE] Money, Companies, …
  - [MEDICINE] Diseases, Drugs, Procedures, …
- Relations
  - What happened to who, when, where, …

# Named Entity Recognition

- **Named entities**: Noun phrases that are of specific type and refer to specific individuals, places, organizations, …
- **Named Entity Recognition**: Technique(s) to identify all mentions of pre-defined named entities in text
  - Identify the mention / phrase: *Boundary detection*
  - Identify the type: *Tagging / classification*

# Examples of Named Entity Recognition Tasks

The patient is a 63-year-old female with a three-year history of bilateral hand numbness and occasional weakness.

Within the past year, these symptoms have progressively gotten worse, to encompass also her feet.

She had a workup by her neurologist and an MRI revealed a C5-6 disc herniation with cord compression and a T2 signal change at that level.

# Approaches to identify named entities

- Depends on kinds of entities that need to be identified

- For well-formatted fields like date, phone numbers: Regular expressions (Recall Week 1)
- For other fields: Typically a machine learning approach

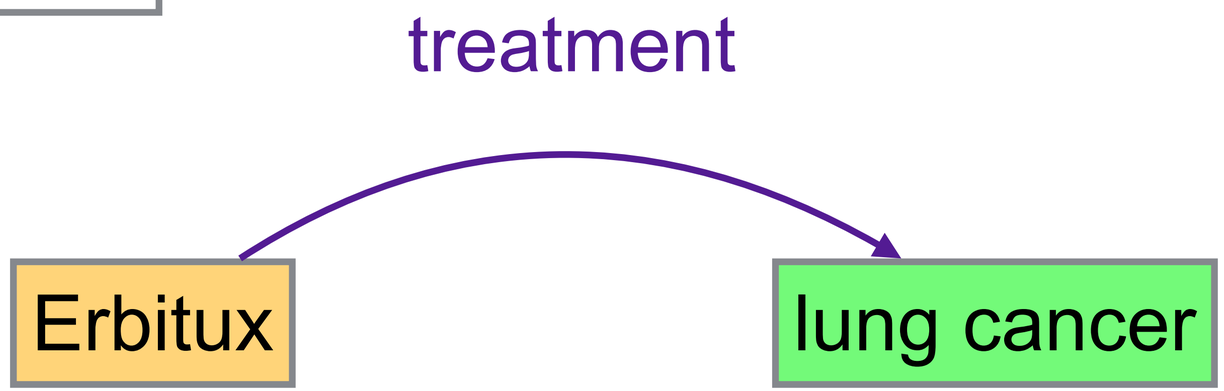
# Person, Organization, Location/GPE

- Standard NER task in NLP research community
- Typically a four-class model
  - PER
  - ORG
  - LOC / GPE
  - Other / Outside (any other class)

# Relation Extraction

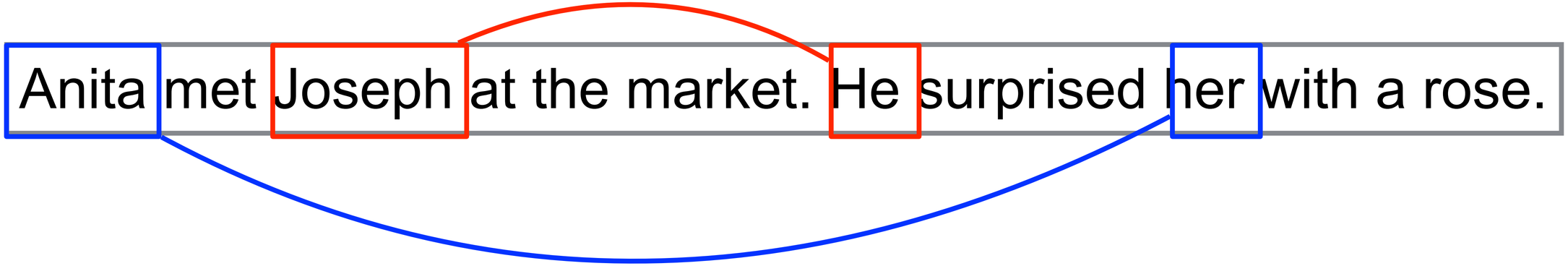
- **Identify relationships between named entities**

Erbitux helps treat lung cancer



# Co-reference Resolution

- **Disambiguate mentions and group mentions together**



# Question Answering

- Given a question, find the most appropriate answer from the text
  - What does Erbitux treat?
  - Who gave Anita the rose?
- Builds on named entity recognition, relation extraction, and co-reference resolution

# Take Home Concepts

- Information Extraction is important for natural language understanding and making sense of textual data
- Named Entity Recognition is a key building block to address many advanced NLP tasks
- Named Entity Recognition systems extensively deploy supervised machine learning and text mining techniques discussed in this course