Lecture 1: Administrata Language-Theoretic Python

Additional material

- Jurafsky & Martin: ***"Speech and Language Processing"***; library + http://www.cs.colorado.edu/~martin/slp.html
- Bird, Klein & Loper: ***"Natural Language Processing with Python"***; library + http://nltk.org/book/
- Manning & Schütze: ***"Foundations of Statistical Natural Language Processing"*** library + http://nlp.stanford.edu/fsnlp/
- ***Online NLP course from Stanford***: https://www.coursera.org/course/nlp

Jurafsky & Martin: "Speech and LanguageProcessing"; library + http://www.cs.colorado.edu/~martin/slp.html

Bird, Klein & Loper: "Natural Language Processing with Python"; library + http://nltk.org/book/

Manning & Schütze: "Foundations of Statistical Natural Language Processing"; library + http://nlp.stanford.edu/fsnlp/

Online NLP course from Stanford: https://www.coursera.org/course/nlp

Errors and Exceptions
http://docs.python.org/tutorial/errors.html

Python has a wide variety of already existing functions:
http://docs.python.org/library/functions.html

Official Unicode Webpage:
http://www.unicode.org/

argparse
https://docs.python.org/2/howto/argparse.html

Encoding
https://docs.python.org/2/library/codecs.html

English Parser:
http://nlp.stanford.edu:8080/parser/

German Parser:
http://kitt.cl.uzh.ch/kitt/parzu/

NLTK book parsing
ch. 8 “Analyzing Sentence Structure”,
ch. 9 “Building Feature Based Grammars”

NLTK list of tasks
http://www.nltk.org/howto/

Programming paradigms

Imperative programming:

- Program = list of statements that change the program state (variables, environment)
- Describe, **how to do something**
- Java, C++, ASM, PHP, Perl, VB, **Python**

Functional programming

- Program = functions, depending only on input (not on program state); order less important
- Describe, **what the program should accomplish**
- Haskell, Lisp, Scala, OCaml, Scala, **Python**

Python

- Imperative:

```
result = []
for num in [3, 4, -7, 0, 2]:
    if (num > 0):
        result += [num**2 - 1]
```

- Functional:

```
result = map(computeVal,
    filter(filterNegatives,
           [3, 4, -7, 0, 2]))
```

```
def computeVal(x):
    return x**2 - 1

def filterNegatives(x):
    return x > 0
```

Python

- Imperative:

```
result = []
for num in [3, 4, -7, 0, 2]:
    if (num > 0):
        result += [num**2 - 1]
```

- Functional:

```
result = map(lambda x: x**2 - 1,
    filter(lambda x: x > 0,
           [3, 4, -7, 0, 2]))
```

Exceptions

```
def average(numList):
    sum = 0
    try:
        for num in numList:
            try:
                sum = sum + num
            except TypeError:
                print 'skipping non-numeric list member'
    except TypeError:
        print 'non-list given'
    else:
        try:
            return sum / len(numList)
        except ZeroDivisionError:
            print 'list empty'
```

Exceptions

```
try:
    ...
    ...
except ExceptionClass:
    ...
except OtherExceptionClass as excVar:
    ... (raise excVar)
except:
    ...
else:
    ...
finally:
    ...
```

Exceptions

Universität Zürich

- You can raise them yourself:

```
if (... data is not good):
    raise Exception
```

or

```
if (... data is not good):
    raise Exception('The data has been very, very naughty')
```

Documentation

```
def retrieveArticle(url, asNltkText = False):
    """Import and clean an article text
    from the web, based on its URL

    @param url: the URL of the article
    @param asNltkText: if True, function returns
       an NLTK.Text object as a result;
       otherwise a list of sentences is returned
    """

    rawHtmlCode = unicode(urlopen(url).read(), "utf-8")
...
```

```
>>> import mymodule
>>> help(mymodule)
>>> help(mymodule.retrieveArticle)
```

Bonus: TextBlob

- like NLTK, only simpler

```
from textblob import TextBlob

text = "The titular threat of The Blob has always..."

blob = TextBlob(text)

blob.tags           # [('The', 'DT'), ('titular', 'JJ'),
                    #  ('threat', 'NN'), ('of', 'IN'), ...]

blob.noun_phrases   # WordList(['titular threat', 'blob',
                    #           'ultimate movie monster',
                    #           'amoeba-like mass', ...])

for sentence in blob.sentences:
    print(sentence.sentiment.polarity)
# 0.060
# -0.341

blob.translate(to="ru")  # 'Титульная угроза...'
```

```
#let the user enter two strings
a = raw_input()
b = raw_input()
#initialize a matrix for computing the Levenshtein distance
m = []
#go through every cell of the matrix starting with the 1st string
for i in range(0, len(a) + 1):
    #add an empty row to the matrix
    m += [[]]
    #go through the 2nd string
    for j in range(0, len(b) + 1):
        #add an empty cell to the current row
        m[i] += [0]
        if (i > 0 and j > 0):
            #min of: above+1, left+1, above_left+{0,1}
            m[i][j] = min(m[i - 1][j] + 1, m[i][j - 1] + 1,
                          m[i - 1][j - 1] +
                          (0 if (a[i - 1] == b[j - 1]) else 1))
#get the total Levenshtein distance from the bottom-right cell
print m[len(a)][len(b)]
```

Lecture 2: Classes, Modules, Namespaces

Functions

```
tokenListA = ["Process", "me", ",", "PLEASE", "!"]
tokenListB = ["ME", "TOO", ",", "please", "do", "me"]

def lcaseList(tokenList):        function definition
    result = []                  function body

    for tok in tokenList:
        result.append(tok.lower())

    return result

lcaseListA = lcaseList(tokenListA)
lcaseListB = lcaseList(tokenListB)

print " ".join(lcaseListA)
print " ".join(lcaseListB)
```

function name

arguments/argument list

function calls

Functions - arguments:

Also possible: optional/default function arguments

Function Arguments:

```
def yetanotherFunction(argx=1, argy=[], argz=None):
    ...
```

If the function is called without arguments, the arguments get their default values.

Function Type

```
def applySomething(inputList, function):
    return function(inputList)
# functions can be used as arguments to other functions
# pass built-in / standard library / own defined function
ourInputList = [1, 1, 2, 3, 5, 8]
print applySomething(ourInputList, sum)  # 20

# pass nameless function / lambda expression
print applySomething(ourInputList,
    lambda inputList: [str(elem) + "x" for elem in inputList])
# ['1x', '1x', '2x', '3x', '5x', '8x',]

# python functions using other functions as input:
print map(str, ourInputList)  # ['1', '1', '2',...]
```

Composite Types

- NLTK corpora:
  - list of sentences + list of words + list of tags + …
- Storing or returning them:
  - list or tuple:

```
def text2corpus(rawText):
    …
    return (sentenceList, wordList, tagList)

# bad! have to remember order and content:
corpus = text2corpus(rawInputText)
print corpus[1] # list of words
```

Composite Types

- NLTK corpora:
  - list of sentences + list of words + list of tags + …
- Storing or returning them:
  - dict:

```
def text2corpus(rawText):
    …
    return { 'sentences': sentenceList,
             'words': wordList,
             'tags': tagList }

# better:
corpus = text2corpus(rawInputText)
print corpus["words"] # list of words

# but can do much better
```

```
class TmpClass3 (object):
    # i=5: creates 1 class variable!
    # until: self.i=number or x.i=number is called
    # then an additional instance variable is created!
    i = 5
    # to change instance and class variables
    def create(self,v, w):
        self.i = v
        TmpClass3.i = w
        TmpClass.i=99
#print i #NameError 5
x = TmpClass3()
print "1: ", x.i, TmpClass3.i #1:  5 5
y = TmpClass3()
print "2: ",y.i, TmpClass3.i #2:  5 5
```

```
y.create(2,3) # change instance variable of y and class variables
print "3: ",y.i, TmpClass3.i, TmpClass.i #3:  2 3 99
print "4: ", x.i, TmpClass3.i, TmpClass.i #4:  3 3 99!!!
x.i = 10 # change instance variable of x
print "5: ",y.i, TmpClass3.i #5:  2 3
print "6: ", x.i, TmpClass3.i #6:  10 3
TmpClass3.i = 11 # change class variable
print "7: ",y.i, TmpClass3.i #7:  2 11
print "8: ", x.i, TmpClass3.i #8:  10 11
y.i = 7
print "9: ",y.i, TmpClass3.i #9:  7 11
print "10: ", x.i, TmpClass3.i #10:  10 11
y.create(2,3) # change instance variable of y and class variables
print "11: ",y.i, TmpClass3.i, TmpClass.i #11:  2 3 99
print "12: ", x.i, TmpClass3.i, TmpClass.i #12:  10 3 99!!!
```

Classes

```
import nltk, nltk.stem

lem = nltk.stem.WordNetLemmatizer()

sentence1 = "here are some tokens to process ."

class SentenceInfo(object):
   tokList = []
   posList = []
   lemmaList = []

info1 = SentenceInfo()

info1.tokList = sentence1.split()
info1.posList = nltk.pos_tag(info1.tokList)
info1.lemmaList = [lem.lemmatize(tok)  for tok in info1.tokList]
# dotted syntax to refer to class attributes

print info1.lemmaList
```

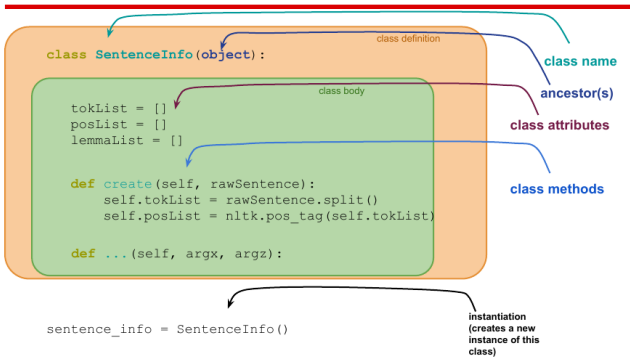
Classes: class methods

```
import nltk, nltk.stem

lem = nltk.stem.WordNetLemmatizer()

class SentenceInfo(object):
   tokList = []
   posList = []
   lemmaList = []

def createSentenceInfo(rawSentence):
   result = SentenceInfo()

   result.tokList = rawSentence.split()
   result.posList = nltk.pos_tag(result.tokList)
   result.lemmaList = [lem.lemmatize(tok)  for tok in result.tokList]

   return result

info1 = createSentenceInfo("here are some tokens to process .")
print info1.lemmaList[3]
```

Classes: class methods

```
import nltk, nltk.stem

lem = nltk.stem.WordNetLemmatizer()

class SentenceInfo(object):
   tokList = []
   posList = []
   lemmaList = []

   def create(self, rawSentence):
      self.tokList = rawSentence.split()
      self.posList = nltk.pos_tag(self.tokList)
      self.lemmaList = [lem.lemmatize(tok)  for tok in self.tokList]
   # NB! needs to be self.tokList
   def getToken(self, tokIndex):
      return (self.tokList[tokIndex], self.posList[tokIndex],
                  self.lemmaList[tokIndex])

info = SentenceInfo()
info.create("here are some tokens to process .")
print info.getToken(3)
```

Classes: syntax



Classes: initialization method

- the *__init__* ("initialization") method
  - is called on (= immediately after) instantiation
  - acts like a "constructor method" (but strictly speaking is not)
  - never returns a value
  - can initialize whatever necessary (e.g. starting values, additional behavior, etc.):

```
class TmpClass(object):
    i = 5

    def __init__(self, startingI):
        self.i = startingI

x = TmpClass(10)
y = TmpClass(5)
```

Classes: initialization method

```
import nltk, nltk.stem

lem = nltk.stem.WordNetLemmatizer()

class SentenceInfo:
   tokList = []
   posList = []
   lemmaList = []

   def __init__(self, rawSentence):
      self.tokList = rawSentence.split()
      self.posList = nltk.pos_tag(self.tokList)
      self.lemmaList = [lem.lemmatize(tok)  for tok in self.tokList]

info1 = SentenceInfo("here are some tokens to process .")
info2 = SentenceInfo("they are already lower-cased and tokenized .")
```

```
import nltk, nltk.stem
lem = nltk.stem.WordNetLemmatizer()
class SentenceInfo (object):
    #class variables SentenceInfo.var (not used)
    tokList = [], posList = [], lemmaList = []
    def create(self, rawSentence):
        # instance/object variable
        self.tokList = rawSentence.split()
        self.posList = nltk.pos_tag(self.tokList)
        self.lemmaList =
                [lem.lemmatize(tok) for tok in self.tokList]
        # class/static variable
        SentenceInfo.tokList=[["hi"],["there"]]
    # NB! needs to be self.tokList
    def getToken(self, tokIndex):
        return (self.tokList[tokIndex], self.posList[tokIndex],
                 self.lemmaList[tokIndex])
info = SentenceInfo()
info.create( "here are some tokens to process ." )
print info.getToken(3) #('tokens', ('tokens', 'NNS'), u'token')
print info.tokList #['here', 'are', 'some', 'tokens', 'to', ...]
print SentenceInfo.tokList #[['hi'], ['there']]
```

Classes: inheritance

```
class SentenceInfo(object):
    tokList = []
    posList = []
    lemmaList = []

    def getToken(self):
        return "something"

class MoreSentenceInfo(SentenceInfo):
    newInfo = "info"

x = MoreSentenceInfo()
print x.tokList      # []
print x.newInfo      # "info"
print x.getToken()   # "something"

print MoreSentenceInfo.__bases__   # get information about parent
```

ancestor(s)

Classes: inheritance, overriding

```
class SentenceInfo:
    someVar = 3.14
    info = "old info"

    def getToken(self):
        return self.info


class MoreSentenceInfo(SentenceInfo):
    info = "new piece of info"


x = MoreSentenceInfo()
print x.someVar      # 3.14
print x.getToken()   # "new piece of info" →
    #the getToken method of the parent class
    #is applied to the instance of the child class,
    #so the overridden variable value is taken from
    #the child class instance
```

Classes: inheritance, initialization method

```
import nltk, nltk.stem

lem = nltk.stem.WordNetLemmatizer()

class SentenceInfo:
    tokList = []
    posList = []
    lemmaList = []

    def __init__(self, rawSentence):
        self.tokList = rawSentence.split()
        self.posList = nltk.pos_tag(self.tokList)
        self.lemmaList = [lem.lemmatize(tok) for tok in self.tokList]


class MoreSentenceInfo(SentenceInfo):

    def __init__(self, raw_sentence, new_info):
      SentenceInfo.__init__(self, raw_sentence)
      self.new_info = new_info
```

- if overriding `__init__` , itialization method of super-class has to be called explicitly!

Modules

Module:

A python file whose code can be imported and used by another python file.

- bigger programs are typically split into several files
- the executed file is called the script
- the additional files are called modules

- by default only Python's built-in types, functions, classes, etc. are available
- several modules are pre-installed (standard library)
- modules can be used via an explicit `import` command

Modules

```
from mymodule import FreqDist

corp = ["this", "sentence", "is", "a", "sample", "sentence"]


freqDist = FreqDist(corp)
print freqDist.get("sentence")     # 0.333


########### mymodule.py:
from collections import defaultdict

class FreqDist(defaultdict):
    def __init__(self, corpus):
    # ...
```

Modules: long/short Names

```
import nltk
nltk.pos_tag(...)
```

vs

```
from nltk import pos_tag
pos_tag()
```

```
import nltk, nltk.stem
lem = nltk.stem.WordNetLemmatizer()
#parent class
class SentenceInfo(object):
    # class variables SentenceInfo.var
    tokList_ = [], posList_ = [], lemmaList_ = []
    # constructor (some kind of)
    def __init__(self, rawSentence):
        self.tokList = rawSentence.split()
        self.posList = nltk.pos_tag(self.tokList)
        self.lemmaList = [lem.lemmatize(tok) for tok in
                          self.tokList]
#child class
class MoreSentenceInfo(SentenceInfo):
    def __init__(self, raw_sentence, new_info):
        #call super class constructor
        SentenceInfo.__init__(self, raw_sentence)
        self.new_info = new_info
```

```
#calls
x = MoreSentenceInfo("here are some tokens to process .",
                     "they are already lower-cased and
                      tokenized .")
SentenceInfo.tokList_="a v e"
#print
print SentenceInfo.tokList_ #a v e
print x.tokList #['here', 'are', 'some', 'tokens', 'to',...]
print x.posList #[('here', 'RB'), ('are', 'VBP'),..]
print x.lemmaList #['here', 'are', 'some', u'token',...]
print x.new_info #they are already lower-cased and tokenized
```

```python
class TmpClass(object):
    i = 5
    def add(self, arg):
        return self.i + arg

x = TmpClass()
y = TmpClass()

#print i # NameError
print x.i # 5

x.i = 10
print TmpClass.add(x, 2) # 12 = 10 + 2
print TmpClass.add(x, y.i) # 15 = 10 + 5
print y.add(2)# 7 = 5 + 2
print y.add(x.i) # 15 = 5 + 10
```

```python
TmpClass.i=99 #affects calc
print y.add(2)# 101 = 99 + 2
print y.add(x.i) # 109 = 99 + 10
print TmpClass.i #99

y.i=20 #affects calc
print y.add(2)# 22 = 20 + 2
print y.add(x.i) # 30 = 20 + 10
print TmpClass.i #99

TmpClass.i=50 #no effect anymore!!!
print y.add(2)# 22 = 20 + 2
print y.add(x.i) # 30 = 99 + 10
print y.i #20
```

```python
def f(x):
    print(locals())
x = 20
print (globals()) # {'x': 20, 'f': <function ...>, ...}
f(10) # {'x': 10}
print (globals()) # {'x': 20, 'f': <function ...>, ...}
print x #20
print locals()['x'] #20
locals()['z'] = 3 #z=3
print z # 3
```

Existing Modules

- Interface to OS Functionality
- HTTP, FTP, E-mail
- Serializing, databases (sqlite3, pickle)
- Parallel computing
- XML
- Specific data structures
- Assisting functions
- Special theme modules
  - NLTK

Modules: file handling

`os`

- deletion
- renaming

`os.path`

- decomposition
- existence checking

Modules: `pip install`

pip

- Package management system for Python
  - used to install and manage packages
    - "package" = a directory of python modules
  - downloads from PyPI ("Python Package Index"): official third-party software repository for Python
  - Python 2.7.9 and later and Python 3.4 and later include pip (pip3 for Python 3) by default
  - Link to Documentation: https://pip.pypa.io/en/stable/

- On command line:
  - `pip install some-package-name`
  - `pip uninstall some-package-name`
  - `pip freeze`
  - `pip list`

Namespaces

Functions, classes, modules keep their own variables, functions, classes by using namespaces

Motivation

- how to solve variable/function name conflicts?
- solution: let names be valid only in a limited scope

Namespaces

- an abstract container for grouping names
- mapping from names to objects
- names in different namespaces are independent
- analogy: folders and files

Namespaces in classes

```
class TmpClass(object):
    i = 5

    def add(self, arg):
        return self.i + arg

x = TmpClass()
y = TmpClass()

print i            # NameError
print x.i          # 5
x.i = 10

print TmpClass.add(x, 2)  # 12 = 10 + 2
print TmpClass.add(x, y.i)# 15 = 10 + 5
print y.add(2)            # 7 = 5 + 2
```

Namespaces in modules

```
import nltk
nltk.pos_tag(...)
```

- nltk module imported for usage

```
from nltk import pos_tag
pos_tag()
```

- the pos_tag function imported into local namespace

Namespaces

```
def f(x):
    print(locals())

x = 20
print (globals()) # {'x': 20, 'f': <function ...>, ...}
f(10)             # {'x': 10}
print (globals()) # {'x': 20, 'f': <function ...>, ...}
```

- Python implements namespaces with dictionaries
- Namespaces are not explicitly declared
- Every module, function, class have their own namespace
- Loops and list comprehensions **do not** have their own namespace

Namespaces: scopes

- **Scope :** piece of Python code where a certain namespace can be reached directly
- In any piece of code 3 namespaces can be reached directly:
  - local scope: namespace with local names. Specific to function/class
  - global scope: namespace of the module/script
  - outer scope: namespace with built-in functions
- name search goes from local to outer

- new declarations belong to the local scope
  - (unless declared with `global`)

Namespaces: global names

```
def f(z):
    global x
    y = z
    x = z
    print(locals())

y = 20
x = 20
print(locals())   # {'x': 20, 'y': 20, 'f': <function...>}
f(10)             # {'z': 10, 'y': 10}
print(locals())   # {'x': 10, 'y': 20, 'f': <function...>}
```

Namespaces

```
class MyClass:
    def __init__(self, i):
        self.i = i

x = MyClass(3)
print(x.i) # 3
```

- what is the difference between `self.i` and `i`?
  - `i` is in the function namespace
  - `self.i` is the namespace of `self` = object namespace
- `x` and `self` point to the same object
- `self` only valid within the function scope

Lecture 3: Input/output Encoding

Unicode

- "Meta-encoding" or an encoding standard
  (not *Zeichenkodierung*, but *Zeichenkodierungsstandard*)
  - describes how characters are presented by code points
    - code point: hexadecimal integer value
      - one code point per character
      - represents characters in an abstract way
        - e.g. U+2746 (U + hexadecimal number)
      - range U+0000..U+10FFFF

- Unicode code points can be encoded according to defined format
  - UTF = "Unicode Transformation Format"
  - mapping from unicode code points to unique byte sequence

UTF-32

- Unicode Transformation Format, 32 bits
- Fixed-length encoding
- 4 bytes for every character
  - 2^{32} = 4 billion
- Direct presentation of the numerical value of a code point
- Problem: space-inefficient
  - compared to ASCII, required memory/space is 4 times bigger
  - so are the files and strings encoded in UTF-32
  - many (or most) characters beyond ASCII are rarely used
- Answer: variable-length encoding

UTF-16

- Each *code point* represented with one or two 16-bit *code units*
- Variable-length encoding
- 2 bytes for more common characters
  - "BMP" = basic multilingual plane (up to 65k characters)
- 4 bytes for the rest
  - Supplementary planes (millions more characters)
- Still at least 2 times bigger memory/space requirements compared to ASCII
- Backwards compatible to UCS-2

UTF-8

- Dominant Character Encoding for the WWW since 2007
- Variable-length encoding
- Each character encoded with 1 to 6 bytes
- **Backward-compatible with ASCII**
  - i.e. characters that can be represented with ASCII are represented in UTF-8 in the same way as in ASCII
- How?
  - some bits used for encoding characters
  - some bits used to indicate whether this byte is the only one or last one or etc.

UTF-8

Bits of code point	First code point	Last code point	Bytes in sequence	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
7	U+0000	U+007F	1	0xxxxxxx					
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx				
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx			
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
26	U+200000	U+3FFFFFF	5	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
31	U+4000000	U+7FFFFFFF	6	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

- 1 byte (128 values) covers ASCII characters
- 2 bytes (1920 values) cover all latin-based alphabets + Greek, Cyrillic, Arabic, etc.
- 3 bytes (63k values) cover the BMP of UTF-16
- the rest covers the supplementary planes

Python file i/o

- Handled by the `file` object
- Opening (loading from disc into memory):

```
f = open('/home/lmascarell/file.txt', 'r')
```

  - *open(filePath, accessMode)*
  - *accessMode*:
    - `'r'`: reading
    - `'w'`: writing (deletes the current content)
    - `'a'`: appending (adds new content to old)
    - `'r+'/'w+'/'a+'`: reading and writing
- Closing (clearing memory/saving to the disk):

```
f.close()
```

Python file input

- `f.read()`: read the entire file into a string
- `f.read(N)`: read at most N bytes
- `f.readline()`: read one line
  - can be repeated to read file one line at a time
- `f.readlines()`: read the entire file into a list of lines as strings
- `for line in f`: will iterate over the file, line by line

Python file input

Example:

```
path = '/home/lmascarell/file.txt'

f = open(path, 'r')
for l in f:
   print l,
f.close()

f = open(path, 'r')
print "".join(f.readlines()),
f.close()
```

the 2 code snippets will print the same output

Python file output

- `f.write(str)`: write string to file
- `f.writelines(strList)`: write a list of strings to file

```
f = open('/home/lmascarell/file.txt', 'w')
f.write("hello\n")
f.writelines(["one\n", "two\n", "three\n"])
f.close()
```

- only strings:

```
val = ['g', 2, "hoho", 5]
f.write(val)      # TypeError
f.write(str(val)) # ok!
```

Python file reading position

Universität Zürich

Reading/writing updates the current position in file

- `f.tell()`: position from file beginning in bytes
- `f.seek(N)`: change position to `N` bytes from file beginning
- `f.seek(N, 1)`: change position to `N` bytes from current position
- `f.seek(N, 2)`: change position to `N` bytes from file end

```
f.seek(-3, 2) # go to 3 bytes before the end
```

Python folder handling

Universität Zürich

- Folder = list of files/folders that it includes
- module for handling: `os`
- `os.listdir(path)`: return a list of files/folders in given path
- `os.path.isdir(path)`: checks if path is a directory/folder or not
- `os.path.isfile`, `os.path.isabs`, ...

```
import os

for subPath in os.listdir('.'):
   print subPath,
   if os.path.isdir(subPath):
      print '/'
```

```
import os
path='./' #current dir
for subPath in os.listdir(path):
    print subPath,
    if os.path.isdir(path+subPath):
        print '/'
path='/' #root dir
for subPath in os.listdir(path):
    print subPath,
    if os.path.isdir(path+subPath):
        print '/'
path='/home/benzro/' #user home dir
for subPath in os.listdir(path):
    print subPath,
    if os.path.isdir(path+subPath):
        print '/'
```

Python argument handling

Define input and output through arguments:

```
$ python test.py arg1 arg2 arg3
```

argument list: `['test.py', 'arg1', 'arg2', 'arg3']`

```
import sys

var1 = sys.argv[1]
var2 = sys.argv[2]
var3 = sys.argv[3]
```

Python argument handling

- `argparse`
  - Module for argument and option handling
  - https://docs.python.org/2/howto/argparse.html
- Help message, arguments information
- Better user experience
- Class: `ArgumentParser`
- Class Methods:
  - `add_argument()`
  - `parse_args()`

```
import argparse
import sys

#Command line Parser
def parse_command_line():
    parser = argparse.ArgumentParser(
        description=" Prints the first 'num_lines' "
                    " of the file 'src' ")
    parser.add_argument('src',
        type=argparse.FileType('r'), #read file object
        metavar='FILE',
        help='source file');
    parser.add_argument('num_lines',
        help="number of lines to print",
        type=int) #integer object
    parser.add_argument('-o', '--out',
        type=argparse.FileType('w'), #write file object
        default=sys.stdout, #print console object
        metavar='FILE',
        help='out file');
    return parser.parse_args()
```

```
#Main
def main(args):
    #argument 2
    for i in range(args.num_lines):
        #argument 3 (args.out) and 1 (args.src)
        args.out.write(args.src.readline())
if __name__ == '__main__':
    #parse arguments
    args = parse_command_line()
    #call main
    main(args)

# Print three lines from one file into another
#$ python V3.2.py
#         '/home/benzro/TextFile.txt'
#         3
#        -o  '/home/benzro/TextFile1.txt'

# Print three lines from file onto console
#$ python V3.2.py
#         '/home/benzro/TextFile.txt'
#         3
```

Unicode in Python 2.x

- **str** type vs **unicode** type

```
a_umlaut = 'ä'
type(a_umlaut)
<type 'str'>


a_umlaut_unicode = u'\xe4'
type(a_umlaut_unicode)
<type 'unicode'>
```

Unicode in Python 2.x

- **bytes** vs **unicode**

```
a_umlaut = 'ä'
print a_umlaut
'\xc3\xa4'
```

UTF-8 Byte representation (hexadecimal)

```
a_umlaut_unicode = u'\xe4'
print a_umlaut_unicode
ä
```

Unicode Code point

Unicode in Python 2.x

Conversion

- from **bytes** to **unicode**
  - map bytes to unicode code points

```
'\xc3\xa4'.decode('utf-8') → u'\xe4'
unicode('ä', 'utf-8')  → u'\xe4'
unicode('\xc3\xa4', 'utf-8')
```

- from **unicode** to **bytes**
  - map unicode code points to bytes

```
u'\xe4'.encode('utf-8') → '\xc3\xa4'
```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
a_umlaut = 'ä'
print type(a_umlaut) #<type 'str'>
a_umlaut_unicode = u'\xe4'
print type(a_umlaut_unicode) #<type 'unicode'>
#UTF-8 Byte representation (hexadecimal)
a_umlaut = 'ä'
a_umlaut #'\xc3\xa4'
#Unicode Code point
a_umlaut_unicode = u'\xe4'
a_umlaut_unicode #ä
#map bytes to unicode code points
'\xc3\xa4'.decode('utf-8') # u'\xe4'
unicode('ä', 'utf-8') # u'\xe4'
unicode('\xc3\xa4', 'utf-8') # u'\xe4'
#map unicode code points to bytes
u'\xe4'.encode('utf-8') # '\xc3\xa4'
```

Encoding Error Handling in Python

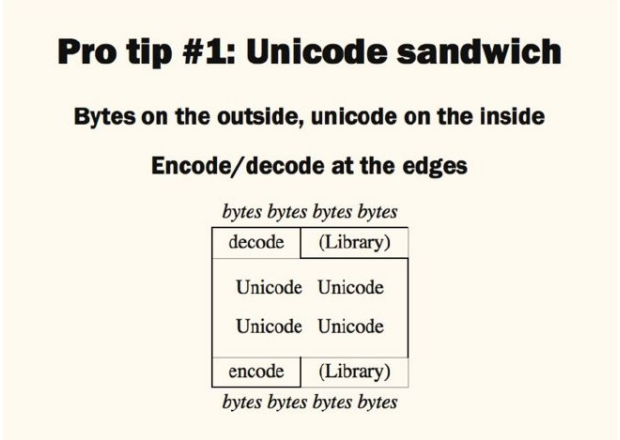
- **Replace:**

```
unicode_str = u'Zürich'
unicode_str.encode('ascii', 'replace')
'Z?rich'
```

- **Ignore:**

```
unicode_str = u'Zürich'
i = unicode_str.encode('ascii', 'ignore')
'Zrich'
```

The Unicode Sandwich



Python and unicode

- Python 2.x:

```
>>> "ö"
'\xc3\xb6'

>>> len("ö")
2

>>> u"ö"
u'\xf6'

>>> len(u"ö")
1

>>> re.findall("ö", u"Höhentraining")
[]
```

File i/o with unicode: codecs

```
import codecs

f = open('song.txt', 'r')
lines = f.readlines()
print lines
# ['Song\n', '\tWell here we are again\n', ...]

g = codecs.open('song.txt', 'r', 'UTF-8')
print g.readlines()
# [u'Song\n', u'\tWell here we are again\n', ...]
```

https://docs.python.org/2/library/codecs.html

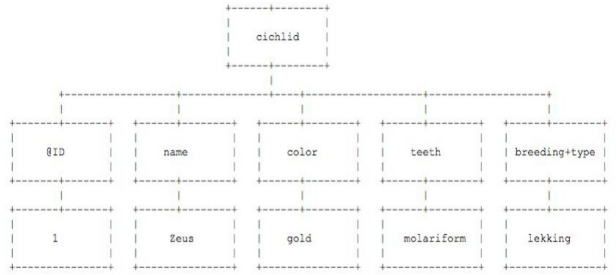
```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import codecs

# open as UTF-8
f1 = open('/home/benzro/TextFile.txt', 'r')
lines1 = f1.readlines()
print "1: ", [lines1[0]] #\xc3\xa4\xc3\xb6\xc3\xbc\n
print "2: ", lines1[0] #äöü
line1 = "äöü\n"
print "3: ", [line1] #\xc3\xa4\xc3\xb6\xc3\xbc\n
print "4: ", line1 #äöü
if lines1[0] == line1:
    print "5: ", "true\n\n" #true
```

```
# open as Unicode (sandwich)
f2 = codecs.open('/home/benzro/TextFile.txt', 'r', 'UTF-8')
lines2 = f2.readlines()
print "1: ", [lines2[0]] #u'\xe4\xf6\xfc\n'
print "2: ", lines2[0] #äöü
line2 = u'äöü\n' #u'\xe4\xf6\xfc\n'
print "3: ", [line2]
print "4: ", line2 #äöü
if lines2[0] == line2:
    print "5: ", "true\n\n"

# print as UTF-8
print "6: ", [codecs.encode(lines2[0],'UTF-8')]
#\xc3\xa4\xc3\xb6\xc3\xbc\n
print "7: ", codecs.encode(lines2[0],'UTF-8') #äöü
```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import codecs
g_in = codecs.open('/home/benzro/TextFile.txt', 'r', 'UTF-8')
g_lines = g_in.readlines()
print g_lines[0],[g_lines[0]] #äöü, [u'\xe4\xf6\xfc\n']
g_o = codecs.open('/home/benzro/TextFile1.txt', 'w', 'UTF-8')
g_o.write(' '.join(g_lines))
#UnicodeDecodeError: 'ascii' codec can't decode
# byte 0xc3 in position 0: ordinal not in range(128)
g_o.write(' '.join(codecs.encode(g_lines[0],'UTF-8')))#Error
g_o.close()
```

Lecture 4: Handling XML

What do you mean by tree structure?




```
<?xml version="1.0" encoding="UTF-8" ?>
<cichlids>
   <cichlid ID="c1">
      <name>Zeus</name>
      <color>gold</color>
      <teeth>molariform</teeth>
      <breeding-type>lekking</breeding-type>
   </cichlid>
</cichlids>
```

What is XML?

- XML vocabulary is not predefined
  - author can define the content and document structure
- XML describes data
  - no information on how data will be presented
- XML documents form a tree structure
  - root element (= parent of all other elements)
  - branches
  - leaves/child nodes

XML concepts

<?xml version="1.0" encoding="UTF-8" ?>

- XML declaration/ XML prolog
  - optional
  - **must** be first line of the document
  - Specify encoding used:
    - `<?xml version="1.0" encoding="UTF-8"?>`
    - UTF-8 is the default character encoding

XML concepts

<?xml version="1.0" encoding="UTF-8" ?>
<node attr1="value" attr2="value2" >

- XML declaration
- Start tag/Opening tag
- Attribute(s)
  - Attribute values must be quoted (single quotes or double quotes)

XML concepts

<?xml version="1.0" encoding="UTF-8" ?>
<node attr1="value" attr2="value2" >
  <child>...</child>
</node>

- XML declaration
- Start tag/Opening Tag
- Attribute(s)
- Child nodes (hierarchy)
- End tag/Closing Tag
  - All nodes must have a closing tag
  - tags are case sensitive

Comments in XML

<!-- This is a comment -->

- Allowed in a comment: everything except --
- Do not put data inside comments

Characters with special meaning in XML

- An XML document can contain any character, except & and <
- Why?

Wrong:

<formula> x < y & x != 2 </formula>

Right:

<formula> x < y & x != 2 </formula>

Checking XML documents

- Well-formedness (“syntactic correctness”)
- Validity (“semantic correctness”)

Well-formedness of XML

Most aspects of well-formedness:

- XML declaration must be on the first line of the document
- Every start tag must have a corresponding end tag
- Tags names must begin with a letter or underscore
- Elements must be properly nested
- There must be a single root element
- Attribute values must be quoted
- An element must not contain two attributes with the same name
- Comments and processing instructions within tags are not allowed

Validity of XML

- Optional, many XML documents are never validated
- Schema languages:
  - DTD (old!)
  - XML Schema
  - Relax NG
  - Schematron
  - (...)
- Content of schema: rules
- (demo)

What is XPath?

- To **navigate** XML documents and select content from them
- Embedded into many XML technologies, as a sublanguage
  - XSLT
  - XQuery
  - XLink / XPointer
  - any popular XML library, in many programming languages

- XPath navigates the XML tree structure
- Important concepts: expression, step, axis, predicate

What is XSLT?

- A language that **transforms** XML documents
- (external slides)
- (Demo)

XSLT processors:

- Saxon (best one, free version)
- MSXSL
- Exselt
- (online tools)
- Libraries, e.g. libxsl

XML parser

- Checks for well-formedness, possibly for validity
- XML parser in most browsers, e.g. to check for well-formedness
- Unlike a well-formedness error, a validation error is not necessarily fatal

Types of XML parsers

- Event based (read as you go → streaming)
  - Advantage: speed, efficiency → does not store everything at once
  - Disadvantage: less flexible
  - e.g. Simple API for XML (SAX)

- Tree based (entire document as a tree at a time)
  - Advantage: entire document at any time, portions of a Document can easily be moved back and forth
  - Disadvantages: larger memory footprint, slow
  - e.g. Document Object Model (DOM) API

XML parsing in Python

- `xml.etree.ElementTree`
- `lxml.etree`

Parsing library: nice-to-haves

ElementTree

- Fast processing OK
- Low Memory consumption OK
- Complete document tree OK
- Optional partial parsing OK
- XPath support Only very limited

Parsing library: nice-to-haves

lxml.etree

- Fast processing OK
- Low Memory consumption OK
- Complete document tree OK
- Optional partial parsing OK
- XPath support OK

Parsing elements

XML node

```
<node attr1="value1" attr2="value2">
    some text
    <child /><child />
</node>some more text
```

Node object

- `n.tag`
- `n.attrib`
- `n.text`
- `n[0], … n[len(n)-1]`
- `n.tail` <type 'str'>

Methods on elements

Element attribute

Like dictionaries:

- `n.get(key)`: *attribute access*
- `n.set(key,value)`: *attribute set*
- `n.items()`: *attributes as a list of feature-value pairs*
- `n.keys()`: *list of attribute key*

Methods on elements

Child nodes

Like lists:

- `n.append(),n.extend(),n.insert(),n.remove()`
- `list(n)`: *list all child nodes*
- `n.find(), n.findall()`:
  search child nodes (Supports XPath paths)

Text+Berg: important elements

```
<?xml version="1.0" encoding="UTF-8"?>
<book id="1901_mul">
    <article n="55">
        <tocEntry title="Alpine Journal" author="Redaktion" lang="de" category="Kleinere Mitteilungen"/>
        <div>
            <s n="55-1" lang="de">
                <w n="55-1-1" pos="ADJA" lemma="alpin">Alpine</w>
                <w n="55-1-2" pos="NN" lemma="Journal">Journal</w>
                <w n="55-1-3" pos="." lemma=".">.</w>
            </s>
        </div>
    </article>
</book>
```

- Book: book
- Article: article
- Sections: div
- Sentences: s
- Words: w
- Entry from Table of Contents: tocEntry

lxml.etree and ElementTree

Example 1

- ElementTree:
  `from xml.etree import cElementTree as ET`
- lxml.etree:
  `import lxml.etree as ET`

```
doc = ET.parse("SAC-Jahrbuch_2008_de.xml")
doc.getroot().tag
words = doc.findall("//w")
len(words)
```

All `w` elements, no matter where they occur

(c)ElementTree: `.//w`

lxml.etree and ElementTree

Example 2

Extract all French articles:

```
for article in doc.findall("//article"):
    tocEntry = article.find("tocEntry")
    if tocEntry != None and tocEntry.get("lang") == "fr":
        for sentence in article.findall(".//s"):
            words = [word.text for word in sentence.findall
            ("w") if word.text]
            print(" ".join(words))
```

Why if tocEntry != None?

Lecture 5: Probabilities

Probabilities
Language Model

Universität Zürich

- Language Model is the distribution of sequence of words
- Applications
  - Handwriting recognition
  - Tagging
    *p(*Adj Noun Verb Det Verb | "fruit flies like a banana"*)*
  - speech recognition
    p("wreck a nice beach" |  *)*
  - Machine Translation
    p("All your base are belong to us." | "君達の基地は、全てCATSがいただいた。"*)*
  - Language Identification
    - Based on "Character-N-Grams" [Dunning, 1994]

Conditional Probabilities
Maximum Likelihood Estimate (MLE)

- What probability estimates should we use for estimating the next word?
  - Absolute frequency: $f(x)$
  - **Relative frequency: $f(x)/N$**
- Maximum Likelihood Estimate (MLE)

$P_{MLE}(w_1...w_n) = C(w_1...w_n)/N$

$P_{MLE}(w_n|w_1...w_{n-1}) = C(w_1...w_n)/C(w_1...w_{n-1})$

Conditional Probabilities
Maximum Likelihood Estimate (MLE)

der Wagen	10
der schnelle	2
der Mensch	3
schnelle Wagen	5

Q: What is the probability that *Wagen* follows *der*?
p(Wagen|der) = ?

P(Wagen|der) = f(der, Wagen)/f(der) = 10/15

Conditional Probabilities in Python:
Nested dictionaries

```
f = {"der": {"Wagen": 10, "schnelle": 2, "Mensch": 3},
  "schnelle": {"Wagen": 5, "Mensch": 2}}

total = sum(f["der"].values())
f["der"]["Wagen"]/float(total)
```

Conditional Probabilities in Python:
nltk.ConditionalFreqDist

```
import nltk

b_a = [("der", "Wagen"), ("der", "Wagen"), ("der", "Wagen"),
("der", "Wagen"), ("der", "Wagen"), ("der", "Wagen"),
("der", "Wagen"), ("der", "Wagen"), ("der", "Wagen"),
("der", "Wagen"), ("der", "schnelle"), ("der", "schnelle"),
("der", "Mensch"), ("der", "Mensch"), ("der", "Mensch"),
("schnelle", "Wagen"), ("schnelle", "Wagen"), ("schnelle", "Wagen"),
("schnelle", "Wagen"), ("schnelle", "Wagen"), ("schnelle", "Mensch"),
("schnelle", "Mensch")]

cfd = nltk.ConditionalFreqDist(b_a)
```

Conditional Probabilities in Python:
nltk.ConditionalFreqDist

```
print cfd["der"]
print cfd["schnelle"]
print cfd["der"]["Wagen"]
print sum(cfd["der"].values())
print cfd["der"]["Wagen"]/float(sum(cfd["der"].values()))

cfd.tabulate()
cfd.plot()
```

```
import nltk
b_a=[("der","Wagen"),("der","Wagen"),("der","Wagen"),
     ("der","Wagen"),("der","Wagen"),("der","Wagen"),
     ("der","Wagen"),("der","Wagen"),("der","Wagen"),
     ("der","Wagen"),("der","schnelle"),("der","schnelle"),
     ("der", "Mensch"),("der", "Mensch"),("der", "Mensch"),
     ("schnelle", "Wagen"),("schnelle", "Wagen"),
     ("schnelle", "Wagen"),("schnelle", "Wagen"),
     ("schnelle", "Wagen")]
cfd=nltk.ConditionalFreqDist(b_a)
print cfd
print cfd["der"]
print cfd["der"]["Wagen"]
print sum(cfd["der"].values())
print cfd["der"]["Wagen"]/float(sum(cfd["der"].values()))
cfd.tabulate()
"""
<ConditionalFreqDist with 2 conditions>
<FreqDist with 3 samples and 15 outcomes>
10
15
0.666666666667
          Mensch Wagen schnelle
     der      3   10      2
schnelle      0    5      0
"""
cfd.plot()
```

Chain Rule

Chain rule for Texts:

$$P(w_1,\ldots,w_l) = \prod_{n=1}^{l} P(w_n|w_1,\ldots,w_{n-1})$$

Problem: Data sparseness ⟶ Zero probabilities

Solution: only N-Grams - Markov assumption

Markov Model

Markov Assumption

- A word w_n depends only on k previous words w_{n-k}, ..., w_{n-1}, where $0 <= k < n$

Markov Models

Unigram	(k = 0) Normal word frequency
Bigram	(k = 1) w_n depends on w_{n-1}
Trigram	(k = 2) w_n depends on w_{n-2}, w_{n-1}

Markov Model
Example: Unigram

Was sich seit Monaten angedeutet hatte , was nach dem letzten , ernüchternden Vorbereitungstreffen der Unterhändler **in** Barcelona keine Überraschung mehr ist , es reift nun zur sicheren Prognose : Das **in** diversen Konferenzen beschworene Ziel , **in** diesem Jahr noch den soliden Rahmen für ein globales Klimaschutz-Abkommen **in** der Nachfolge des Kyoto-Protokolls zu zimmern , ist illusorisch geworden . Daran ist nicht mehr zu zweifeln , wenn nicht einmal der dänische Ministerpräsident Lars Løkke Rasmussen als Repräsentant des traditionell optimistisch auftretenden Gastgebers noch an einen Erfolg glaubt . Kopenhagen wird eine Station von vielen werden auf dem schwierigen Weg der Staatengemeinschaft , die Treibhausgase **in** diesem Jahrhundert gerade noch auf ein halbwegs erträgliches Ausmaß zu begrenzen .

$$P(\text{in}) = \text{count(in)} / \text{count(all words)}$$
$$P(\text{in}) = 5 / 117 \approx 0.042$$

Markov Model
Example: Bigram

Was sich seit Monaten angedeutet hatte , was nach dem letzten , ernüchternden Vorbereitungstreffen der Unterhändler in Barcelona keine Überraschung mehr ist , es reift nun zur sicheren Prognose : Das in diversen Konferenzen beschworene Ziel , **in diesem** Jahr noch den soliden Rahmen für ein globales Klimaschutz-Abkommen in der Nachfolge des Kyoto-Protokolls zu zimmern , ist illusorisch geworden . Daran ist nicht mehr zu zweifeln , wenn nicht einmal der dänische Ministerpräsident Lars Løkke Rasmussen als Repräsentant des traditionell optimistisch auftretenden Gastgebers noch an einen Erfolg glaubt . Kopenhagen wird eine Station von vielen werden auf dem schwierigen Weg der Staatengemeinschaft , die Treibhausgase **in diesem** Jahrhundert gerade noch auf ein halbwegs erträgliches Ausmaß zu begrenzen .

$$P(\text{diesem}|\text{in}) = P(\text{in, diesem})/P(\text{in})$$

$$P(\text{diesem}|\text{in}) = \frac{\frac{2}{116}}{\frac{5}{117}} \approx 0.411$$

Markov Model
Example: Sentence probability

Universität Zürich

Sentence: This sentence has 5 tokens

➔ Bigram model?

➔ P(This sentence has 5 tokens)

P(This|..) x P(sentence|this) x P(has|sentence) x P(5|has) x P(tokens|5)

Markov Model
Python - defaultdict

```
def count1(text):
    absfreqs = {}
    for word in text.split():
        absfreqs[word] = absfreqs.setdefault(word, 0) + 1
    return absfreqs
```

or using defaultdict:

```
from collections import defaultdict

def count2(text):
  freqs = defaultdict(int)
  for word in text.split():
    freqs[word] += 1
  return freqs
```

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-
from collections import defaultdict
def main():
    text = "This is a sentence . Each sentence has a number of
                „words . The number of sentences is 3 ."
    #unigram
    absfreqs=count1_unigram(text)
    freqs=count2_unigram(text)
    # bigram
    f=count_bigrams(text)
    print absfreqs, "\n\n", freqs, "\n\n", f, "\n\n"
    print freqs["a"], "\n\n", f["a"], "\n\n", f["a"]["number"]
def count1_unigram(text):
    absfreqs={}
    for word in text.split():
        absfreqs[word]=absfreqs.setdefault(word,0)+1
    return absfreqs
def count2_unigram(text):
    freqs=defaultdict(int)
    for word in text.split():
        freqs[word]+=1
    return freqs
def count_bigrams(text):
    f=defaultdict(lambda: defaultdict(int))
    hist=None
    for word in text.split():
        f[hist][word]+=1
        hist=word
    return f
if __name__ == "__main__":
    main()
```

Features
Feature extraction

Universität Zürich[UZH]

Problem: filter out spam

NON-SPAM

From: "Patrick Meyer " <p.meyer@ifi.uzh.ch>
Subject: **Submission of project note**
Yes, let's talk about the overview after these holidays [...]

SPAM

From: ObyhYtatiz@bk.ru
Subject: **Money**
Hi Let me introduce Myself my name Loretta and I know the Secret of making money $4844 if you WAnt to know the Secret follow this [...]

F_1	F_2	…	F_N	Class (or label)
V_1	V_2	…	V_N	NON-SPAM
V_1	V_2	…	V_N	SPAM
[...]				

Features
Types

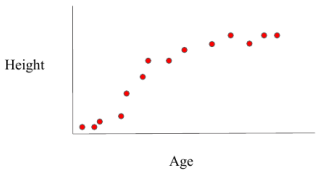
- Numeric
  - Discrete
  - Continuous
- Nominal
  - binary (2 values)
  - N values
- Hybrid

Machine Learning Types

Univer Zürich

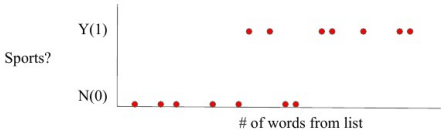
- Supervised learning - Labeled training data
- Unsupervised learning - Unlabeled training data

Machine Learning Types
Supervised Learning



Regression - continuous valued output

Machine Learning Types
Supervised Learning

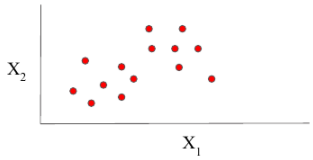


Classification - discrete valued output

- Regression - predicts continuous valued output
- Classification - predicts discrete valued output
  - Spam filtering (spam / non-spam)
  - OCR (A / B / a / 0 / 1 / % / …)
  - Word sense disambiguation ($bank_{river\_edge}$/$bank_{institution}$/…)
  - Document Classification (news/technical/law/…)

Machine Learning Types
Unsupervised Learning

- Unsupervised learning - Unlabeled training data
  - Clustering - K-means algorithm



Learning Algorithms

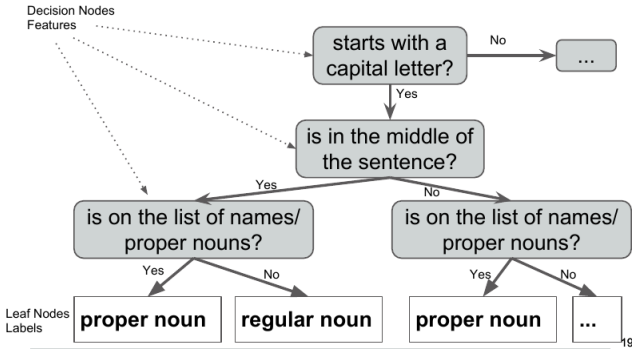
Learning Algorithms look for the hypothesis that generalizes best over a training set.

Learning Algorithms
Naive Bayes Classifier

- It looks for the label that maximizes P(*T* | *features*)
  1. Compute prior probability to each label p(*T'*)
  2. Combine the contribution of each feature with the prior probability:

     $T = \text{argmax}_{T'} p(T') \times p(f_1 \mid T') \times p(f_2 \mid T') \times \ldots$

Learning Algorithms
Decision Trees



Learning Algorithms
Decision Trees

- Learning: select feature that reduces total entropy the most
  - Entropy of a set:

    $H = -\sum_{output} p(output) \cdot \log p(output)$
  - Little variety – entropy is low
  - Mixed output values – entropy is high
- Learning type: iterative

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-
import nltk
import math
def entropy(labels):
    # frequency of labels
    freqdist = nltk.FreqDist(labels)
    # plot frequency of labels
    print "freq. distribution:"
    freqdist.tabulate()
    # probability of labels
    #probs = [freqdist.freq(l) for l in nltk.FreqDist(labels)]
    probs=[]
    for label in nltk.FreqDist(labels):
        print "label: ", label
        prob=freqdist.freq(label)#probability
        probs.append(prob)#list of probabilities
        print "probabilities: ", probs
    #entropy
    #entropy = -sum([p * math.log(p, 2) for p in probs])
    entropy=0
    for prob in probs:
        log = math.log(prob, 2)
        entropy_i = -prob * log
        entropy+=entropy_i
        print "prob, log, entropy", prob, log, entropy_i
    #return
    return entropy
```

```
print "no variety: low entropy (high discriminatory power)"
print entropy(['male', 'male', 'male', 'male'])#0.0
print
print "little variaty: medium entropy"
print entropy(['male', 'female', 'male', 'male'])#0.811
print
print "max variety: high entropy (no discriminatory power)"
print entropy(['female', 'male', 'female', 'male'])#1.0
print
print "little variaty: medium entropy"
print entropy(['female', 'female', 'male', 'female'])#0.811
print
print "no variety: low entropy (high discriminatory power)"
print entropy(['female', 'female', 'female', 'female'])#0.0
```

Evaluation
Split Data

Data splitting

- Training data set ≈ 80%
- Development (or tuning) data set ≈ 10%
- Testing (or evaluation) data set ≈ 10%

Cross validation

- Divide corpus into *N* subsets (folds)
  - 10-fold cross validation

Evaluation

Accuracy

- Percentage of things right.
- Be careful with unbalanced corpus

Precison and Recall

System	Actual target	Actual ¬target
selected	*tp*	*fp*
¬selected	*fn*	*tn*

precision = tp/(tp+fp)   recall = tp/(tp+fn)

Evaluation
Confusion Matrix

Each cell [*i*,*j*] indicates how often label *j* was predicted when the correct label was *i*

	NN	IN	AT	JJ	.	NNS	,	VB	NP
NN	<11.8%>	0.0%	.	0.2%	.	0.0%	.	0.3%	0.0%
IN	0.0%	<9.0%>	.	.	.	0.0%	.	.	.
AT	.	.	<8.6%>	.	.	.	.	.	.
JJ	1.7%	.	.	<3.9%>	.	.	.	0.0%	0.0%
.	.	.	.	.	<4.8%>	.	.	.	.
NNS	1.5%	.	.	.	.	<3.2%>	.	.	0.0%
,	.	.	.	.	.	.	<4.4%>	.	.
VB	0.9%	.	.	0.0%	.	.	.	<2.4%>	.
NP	1.0%	.	.	0.0%	.	.	.	.	<1.8%>

(row = reference; col = test)

[NLTK book, chapter 6]

ML in Python
Name genders

```
#data set
from nltk.corpus import names
import random
raw_male_names = names.words('male.txt') # ['Aamir', 'Aaron',
raw_female_names = names.words('female.txt') # ['Abagael', 'Abagail',
labeled_male_names = [(name, 'male') for name in raw_male_names]
labeled_female_names = [(name, 'female') for name in raw_female_names]

name_set = labeled_male_names + labeled_female_names
random.shuffle(name_set)
#[('Nissy', 'female'), ('Warren', 'male'), ('Olivie', 'female'),...
```

ML in Python
Name genders

```
def generate_features(name):
  """function to generate features from an input name"""
    return {'last_letter': name[-1], 'len': len(name) }
#extracting features
feature_set = [(gender_features(n), g) for (n,g) in name_set]
#training set, test set
train_set, test_set = feature_set[500:], feature_set[:500]
#train the classifier
classifier = nltk.NaiveBayesClassifier.train(train_set)

#apply the classifier
classifier.classify(gender_features('Neo'))
#'male'
classifier.classify(gender_features('Trinity'))
#'female'
#classifier accuracy on the test set
print nltk.classify.accuracy(classifier, test_set)   #0.758
```

References

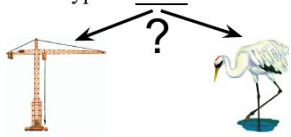
➔ In Python

- nltk.NaiveBayesClassifier
  - NLTK book, sections 6.1, 6.5
- nltk.classify.decisiontree.DecisionTreeClassifier
  - NLTK book, section 6.4
- nltk.classify.maxent.MaxentClassifier
  - NLTK book, chapter 6

Lecture 7: Sequence Tagging

Tagging

- Part-of-speech (POS) tagging
  - Fruit$_{\text{noun}}$ flies$_{\text{noun}}$ like$_{\text{verb}}$ a$_{\text{determiner}}$ banana$_{\text{noun}}$
  - Time$_{\text{noun}}$ flies$_{\text{verb}}$ like$_{\text{preposition}}$ an$_{\text{determiner}}$ arrow$_{\text{noun}}$
- Named entity recognition
  - Prof. ~~people~~Volk gives a presentation in ~~train~~Zug
  - Prof. Volk$_{\text{NAME}}$ hält einen Vortrag in Zug$_{\text{PLACE}}$
- Word sense disambiguation
  - A duck is smaller than a typical crane



POS Tagging Taggers

NLTK built-in:

```
import nltk

tokenList = nltk.word_tokenize( "Fed raises interest rates 0.5 percent" )
# ['Fed', 'raises', 'interest', 'rates', '0.5', 'percent']

posResult = nltk.pos_tag(tokenList)
# [('Fed', 'NNP'), ('raises', 'VBZ'), ('interest', 'NN'), ('rates', 'NNS'),
('0.5', 'CD'), ('percent', 'NN')]
# posResult[0] = ('Fed', 'NNP')
# posResult[0][1] = 'NNP'
```

```
import nltk
tokenList = \
    nltk.word_tokenize( "Fed raises interest rates 0.5 percent" )
print tokenList
# ['Fed', 'raises', 'interest', 'rates', '0.5', 'percent']
posResult = nltk.pos_tag(tokenList)
print posResult
# [('Fed', 'NNP'), ('raises', 'VBZ'), ('interest', 'NN'),
# ('rates', 'NNS'), ('0.5', 'CD'), ('percent', 'NN')]
print posResult[0], posResult[0][1]
# posResult[0] = ('Fed', 'NNP')
# posResult[0][1] = 'NNP'
```

POS Tagging Tagged Data

- The Brown corpus:
  - `nltk.corpus.brown.raw()`
  - `nltk.corpus.brown.words()`
  - `nltk.corpus.brown.sents()`
  - `nltk.corpus.brown.tagged_words()`
    `[(u'The', u'AT'), (u'Fulton', u'NP-TL'), …]`
  - `nltk.corpus.brown.tagged_sents()`

POS Tagging NLTK simplified tag set

Tag	Meaning	English Examples
ADJ	adjective	*new, good, high, special, big, local*
ADP	adposition	*on, of, at, with, by, into, under*
ADV	adverb	*really, already, still, early, now*
CONJ	conjunction	*and, or, but, if, while, although*
DET	determiner, article	*the, a, some, most, every, no, which*
NOUN	noun	*year, home, costs, time, Africa*
NUM	numeral	*twenty-four, fourth, 1991, 14:24*
PRT	particle	*at, on, out, over per, that, up, with*
PRON	pronoun	*he, their, her, its, my, I, us*
VERB	verb	*is, say, told, given, playing, would*
.	punctuation marks	*. , ; !*
X	other	*ersatz, esprit, dunno, gr8, univeristy*

POS Tagging How? - Introduction

There are essentially two sources of information:

1. Syntagmatic Information
   Look at the tags of other words in the context
   e.g. a new *play*. NN or VBP?
   AT JJ NN vs. AT JJ VBP

2. Lexical Information
   Consider only the word involved
   → assign the most common tag.

Markov Model Taggers Bigram tagger

```
import nltk
from nltk.tag import *

bigram_tagger = nltk.BigramTagger(train_sents)
print bigram_tagger.tag(untag(sents[2007]))
print bigram_tagger.tag(untag(sents[4203]))
# [('The', 'AT'), ('population', 'NN'), ('of', 'IN'), ('the', 'AT'),
('Congo', 'NP'), ('is', 'BEZ'), ('13.5', None), ('million', None), (',',
None), ('divided', None), ('into', None), …
```

- problem: `'13.5'` is OOV

```
import nltk
from nltk.tag import *
train_sents=nltk.corpus.brown.tagged_sents()[:2000]
sents=nltk.corpus.brown.tagged_sents()[2000:]

bigram_tagger = nltk.BigramTagger(train_sents)

print bigram_tagger.tag(untag(sents[4203]))
```

```
[(u'The', u'AT'), (u'population', u'NN'), (u'of', u'IN'), (u'the', u'AT'), (u'Congo', None), (u'is', None), (u'13.5', None), (u'million', None), (u',', None), (u'divided', None), (u'into', None), (u'at', None), (u'least', None), (u'seven', None), (u'major', None), (u'``', None), (u'culture', None), (u'clusters', None), (u"''", None), (u'and', None), (u'innumerable', None), (u'tribes', None), (u'speaking', None), (u'400', None), (u'separate', None), (u'dialects', None), (u'.', None)]
```

Markov Model Taggers
Backing off

- longer context (bigger k) means more probability that a particular n-gram has not been seen = "sparse data" effect
- solution: back-off to smaller k:
  - try to find $p(t_i \mid t_{i-1}, t_{i-2})$
  - if "$t_{i-2}t_{i-1}t_i$" has not been seen, try $p(t_i \mid t_{i-1})$
  - etc.
- Unigram Tagger: only based on $p(w_i \mid t_i)$
  - will assign the most probable tag per word
  - no matter the context
- Default tagger: will assign the same tag to all words

```
import nltk
from nltk.tag import *
train_sents=nltk.corpus.brown.tagged_sents()[:2000]
sents=nltk.corpus.brown.tagged_sents()[2000:]

default_tagger = nltk.DefaultTagger("NN")
unigram_tagger = nltk.UnigramTagger(train_sents,
                                    backoff=default_tagger )
bigram_tagger = nltk.BigramTagger(train_sents,
                                  backoff=unigram_tagger )

print bigram_tagger.evaluate(sents[2000:2100])

print bigram_tagger.tag(untag(sents[4203]))
```

```
0.7959
```

```
[(u'The', u'AT'), (u'population', 'NN'), (u'of', u'IN'), (u'the', u'AT'), (u'Congo', 'NN'), (u'is', u'BEZ'), (u'13.5', 'NN'), (u'million', u'CD'), (u',', u','), (u'divided', u'VBN'), (u'into', u'IN'), (u'at', u'IN'), (u'least', u'AP'), (u'seven', u'CD'), (u'major', u'JJ'), (u'``', u'``'), (u'culture', 'NN'), (u'clusters', u'NNS'), (u"''", u"''"), (u'and', u'CC'), (u'innumerable', 'NN'), (u'tribes', 'NN'), (u'speaking', u'VBG'), (u'400', u'CD'), (u'separate', u'JJ'), (u'dialects', 'NN'), (u'.', u'.')]
```

Markov Model Taggers
My walk was awesome

$p(t \mid$ "my"$) = \{$ 'pron': *0.99, ... }*
$p(t \mid$ "walk"$) = \{$ 'verb': *0.8,* 'noun': *0.19, ... }*
$p(t \mid$ "was"$) = \{$ 'verb': *0.92, ... }*
$p(t \mid$ "awesome"$) = \{$ 'adj': *0.99, ... }*

$p(t_i \mid t_{i-1} =$ '<s>'$) = \{$ 'noun': *0.35,* 'pron': *0.3 ... }*
$p(t_i \mid t_{i-1} =$ 'pron'$) = \{$ 'verb': *0.3,* 'noun': *0.35,* 'adj': *0.3 ... }*
$p(t_i \mid t_{i-1} =$ 'verb'$) = \{$ 'adj': *0.2,* 'noun': *0.15,* 'verb': *0.01, ... }*
$p(t_i \mid t_{i-1} =$ 'noun'$) = \{$ 'verb': *0.3,* 'noun': *0.2, ... }*

Best tag for 'walk':
$p($'verb' | 'walk'$) \cdot p($'verb' | 'pron'$) = 0.8 \cdot 0.3 = 0.24$
$p($'noun' | 'walk'$) \cdot p($'noun' | 'pron'$) = 0.19 \cdot 0.35 = 0.0665$

But 'walk' as 'verb' brings down the likelihood of the whole sequence:
$p($'pron **verb** verb adj' | 'my walk was awesome'$) =$
$p($'pron' | 'my'$) \cdot p($**'verb' | 'walk'**$) \cdot p($'verb' | 'was'$) \cdot p($'adj' | 'awesome'$) \cdot$
$p($'pron' | '<s>'$) \cdot p($'verb' | 'pron'$) \cdot p($**'verb' | 'verb'**$) \cdot p($'adj' | 'verb'$) =$
$0.99 \cdot \mathbf{0.8} \cdot 0.92 \cdot 0.99 \cdot 0.3 \cdot 0.3 \cdot \mathbf{0.01} \cdot 0.2 = \mathbf{0.00013..}$

$p($'pron **noun** verb adj' | 'my walk was awesome'$) =$
$p($'pron' | 'my'$) \cdot p($**'noun' | 'walk'**$) \cdot p($'verb' | 'was'$) \cdot p($'adj' | 'awesome'$) \cdot$
$p($'pron' | '<s>'$) \cdot p($'verb' | 'pron'$) \cdot p($**'noun' | 'verb'**$) \cdot p($'adj' | 'verb'$) =$
$0.99 \cdot \mathbf{0.2} \cdot 0.92 \cdot 0.99 \cdot 0.3 \cdot 0.3 \cdot \mathbf{0.15} \cdot 0.2 = 0.00049..$

Viterbi Algorithm

- Instead of
  $t = \text{argmax}_{t'}\ p(w_i \mid t') \cdot p(t' \mid t_{i-1})$ for each i
- Viterbi = optimization over the whole sequence:
  $\boldsymbol{t} = \text{argmax}_{\boldsymbol{t}'}\ p(\boldsymbol{t}' \mid \boldsymbol{w})$
- Using dynamic programming
  - to be explained in lecture #11

Hidden Markov Model

- No tagged training data
- In NLTK:

```
from nltk.tag.hmm import HiddenMarkovModelTagger
hmm_tagger = HiddenMarkovModelTagger.train(train_sents)
```

```
import nltk
from nltk.tag import *
from nltk.tag.hmm import HiddenMarkovModelTagger

train_sents=nltk.corpus.brown.tagged_sents()[:2000]
sents=nltk.corpus.brown.tagged_sents()

hmm_tagger = HiddenMarkovModelTagger.train(train_sents)

print hmm_tagger.tag(untag(sents[4203]))
```

```
[(u'The', u'AT'), (u'population', u'NN'), (u'of', u'IN'), (u'the', u'AT'), (u'Congo', u'NN'), (u'is', u'BEZ'), (u'13.5', u'RB'), (u'million', u'CD'), (u',', u','), (u'divided', u'VBN'), (u'into', u'IN'), (u'at', u'IN'), (u'least', u'AP'), (u'seven', u'CD'), (u'major', u'JJ'), (u'``', u'``'), (u'culture', u'JJ'), (u'clusters', u'NNS'), (u"''", u"''"), (u'and', u'CC'), (u'innumerable', u'VBD'), (u'tribes', u'.'), (u'speaking', u'.'), (u'400', u'.'), (u'separate', u'.'), (u'dialects', u'.'), (u'.', u'.')]
```

RegExp tagger

- List of regular expressions and corresponding POS tags

```
>>> regexp_tagger = nltk.RegexpTagger(
...     [(r'^-?[0-9]+(.[0-9]+)?$', 'CD'),   # cardinal numbers
...      (r'(The|the|A|a|An|an)$', 'AT'),   # articles
...      (r'.*able$', 'JJ'),                # adjectives
...      (r'.*ness$', 'NN'),                # nouns formed from adjectives
...      (r'.*ly$', 'RB'),                  # adverbs
...      (r'.*s$', 'NNS'),                  # plural nouns
...      (r'.*ing$', 'VBG'),                # gerunds
...      (r'.*ed$', 'VBD'),                 # past tense verbs
...      (r'.*', 'NN')                      # nouns (default)
... ])
>>> regexp_tagger.tag(test_sent)
[('The', 'AT'), ('Fulton', 'NN'), ('County', 'NN'), ('Grand', 'NN'), ('Jury', 'NN'),
('said', 'NN'), ('Friday', 'NN'), ('an', 'AT'), ('investigation', 'NN'), ('of', 'NN'),
("Atlanta's", 'NNS'), ('recent', 'NN'), ('primary', 'NN'), ('election', 'NN'),
('produced', 'VBD'), ('``', 'NN'), ('no', 'NN'), ('evidence', 'NN'), ("''", 'NN'),
('that', 'NN'), ('any', 'NN'), ('irregularities', 'NNS'), ('took', 'NN'),
('place', 'NN'), ('.', 'NN')]
```

```
import nltk
from nltk.tag import *
regexp_tagger = nltk.RegexpTagger(
 [(r'^-?[0-9]+(.[0-9]+)?$', 'CD'),# cardinal numbers
 (r'(The|the|A|a|An|an)$', 'AT'),# articles
 (r'.*able$', 'JJ'),# adjectives
 (r'.*ness$', 'NN'),# nouns formed from adjectives
 (r'.*ly$', 'RB'),# adverbs
 (r'.*s$', 'NNS'),# plural nouns
 (r'.*ing$', 'VBG'),# gerunds
 (r'.*ed$', 'VBD'),# past tense verbs
 (r'.*', 'NN')])# nouns (default)
sents=nltk.corpus.brown.tagged_sents()
print regexp_tagger.tag(untag(sents[4203]))
```

[(u'The', 'AT'), (u'population', 'NN'), (u'of', 'NN'), (u'the', 'AT'), (u'Congo', 'NN'), (u'is', 'NNS'), (u'13.5', 'CD'), (u'million', 'NN'), (u',', 'NN'), (u'divided', 'VBD'), (u'into', 'NN'), (u'at', 'NN'), (u'least', 'NN'), (u'seven', 'NN'), (u'major', 'NN'), (u'``', 'NN'), (u'culture', 'NN'), (u'clusters', 'NNS'), (u'"', 'NN'), (u'and', 'NN'), (u'innumerable', 'JJ'), (u'tribes', 'NNS'), (u'speaking', 'VBG'), (u'400', 'CD'), (u'separate', 'NN'), (u'dialects', 'NNS'), (u'.', 'NN')]

Accuracy in NLTK

```
import nltk
from nltk.tag import *

default_tagger = nltk.DefaultTagger("NN")
unigram_tagger = nltk.UnigramTagger(train_sents, backoff=default_tagger)
bigram_tagger = nltk.BigramTagger(train_sents, backoff=unigram_tagger)

print bigram_tagger.evaluate(test_sents)
```

Confusion matrix in NLTK

```
def tagList(sents):
    '''remove tokens and leave only tags'''
    return [tag for sent in sents for word, tag in sent]

def applyTagger(tagger, corpus):
    '''apply a tagger to a corpus'''
    return [tagger.tag(nltk.tag.untag(sent)) for sent in corpus]

goldTags = tagList(test_sents)
testTags = tagList(applyTagger(bigram_tagger, test_sents))

cm = nltk.ConfusionMatrix(goldTags, testTags)
print cm.pp(sort_by_count=True, show_percents=True,
    truncate=9)
```

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-
import nltk
from nltk.tag import *
from nltk.tag.hmm import HiddenMarkovModelTagger
def tagList(sents):
    '''remove tokens and leave only tags'''
    return [tag for sent in sents for word, tag in sent]
def applyTagger(tagger, corpus):
    '''apply a tagger to a corpus'''
    return [tagger.tag(untag(sent)) for sent in corpus]

train_sents=nltk.corpus.brown.tagged_sents()[:2000]
test_sents=nltk.corpus.brown.tagged_sents()[2000:2100]

hmm_tagger = HiddenMarkovModelTagger.train(train_sents)

goldTags = tagList(test_sents)
testTags = tagList(applyTagger(hmm_tagger, test_sents))

cm = nltk.ConfusionMatrix(goldTags, testTags)
print cm.pretty_format(sort_by_count= True,
                       show_percents= True, truncate=9)
```

```
from nltk.metrics import ConfusionMatrix
ref  = 'DET NN VB DET JJ NN NN IN DET NN'.split()
tagged = 'DET VB VB DET NN NN NN IN DET NN'.split()
cm = ConfusionMatrix(ref, tagged)
print cm.pretty_format(show_percents= True, truncate=9)
```

```
    |      D                                  |
    |      E      I      J      N      V      |
    |      T      N      J      N      B      |
----+-----------------------------------------+
DET | <30.0%>      .      .      .      .      |
 IN |      . <10.0%>     .      .      .      |
 JJ |      .      .     <.> 10.0%     .      |
 NN |      .      .      . <30.0%> 10.0%      |
 VB |      .      .      .      . <10.0%>      |
----+-----------------------------------------+
(row = reference; col = test)
```

Lecture 8: External Programs, NLP Pipeline

External Programs
The subprocess Module

```
>>> import subprocess
>>> subprocess.call(["touch", "file.txt"]) #prints 0
```

- the function `call` executes a command with arguments and returns its exit status
- command and arguments given as a list of strings
- command functionality executed
- meant mostly for simple commands

- **output** of the command **lost**

```
import subprocess
# Update the access and modification times
# of each FILE to the current time.
subprocess.call(["touch",
                 "/home/benzro/PycharmProjects/Lectures_PCL2/"
                 "TextFile.txt"])
```

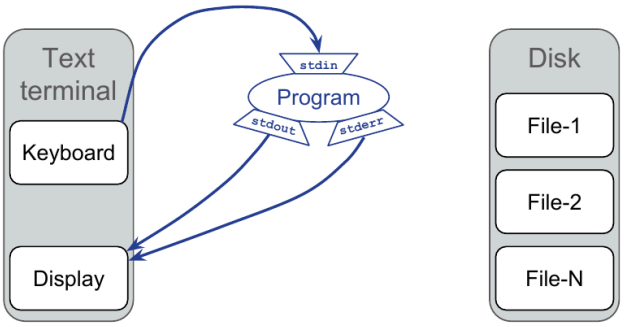
External Programs
The subprocess Module

```
>>> import subprocess
>>> subprocess.check_output(["ls", "-l"])
#prints a list of files with mod. dates, sizes, etc.
```

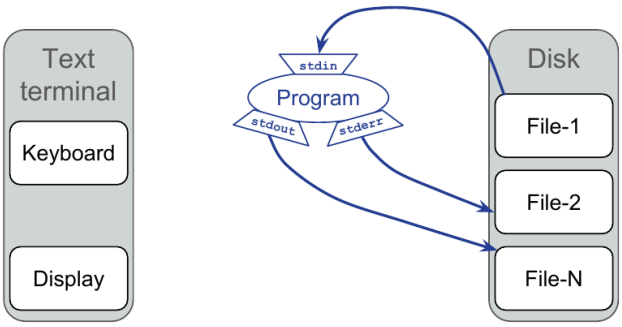
- the function `check_output` executes a command with arguments and returns its output as string
- exit status not lost -- if not 0, `CalledProcessError` thrown
- still meant mostly for simple commands

```
import subprocess
# prints a list of files with mod. dates, sizes,etc.
out=subprocess.check_output(["ls", "-l", "/home/benzro/"])
print out
```

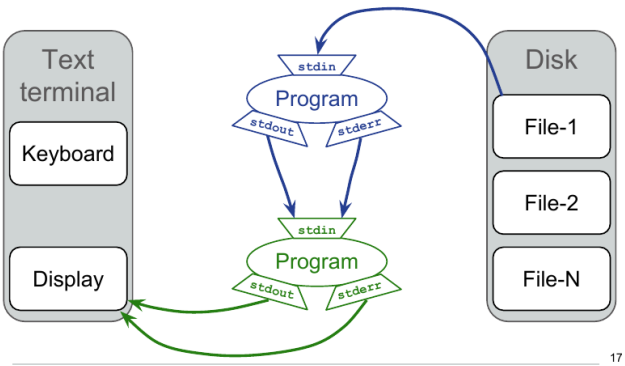
External Programs
Redirection



External Programs
Redirection



External Programs
Redirection



External Programs
Redirection in Unix

- redirection between programs is done with pipes: |

```
command1 | command2
```

- redirects `stdout` of `command1` into `stdin` of `command2`
- >, <, 2> and | can be combined

  e.g. to redirect `stderr` of `command1` into `stdin` of `command2`

```
ls -l 2>ls-err.log | grep '\.txt' | cut -d . -f 1 >& log
```

External Programs
Redirection in Python

- subprocess.Popen() enables more detailed control over the process and its standard streams:

```
inputFile = open("/etc/wgetrc", "r")

proc = subprocess.Popen(['./test.py', 'arg'],
       stdin=inputFile,
#optional
       stdout=open("/tmp/test-out.txt", "w"))
#optional
#process executed in background

proc.wait()
#wait for its completion
```

External Programs
Redirection in Python

- pipe redirection:

```
proc = subprocess.Popen(['./test.py', 'arg'],
       stdin=subprocess.PIPE,
       stdout=subprocess.PIPE)

proc.stdin.write("hello\n")
proc.stdin.write("hi again\n")
proc.stdin.close() #must close for the process to
continue

for outputLine in proc.stdout:
   print('OUT: ' + outputLine),

proc.wait() #must wait -- why?
```

External Programs
Example

```
import subprocess

proc = subprocess.Popen(['tree-tagger-german'],
    stdin=subprocess.PIPE,
    stdout=subprocess.PIPE,
    stderr=subprocess.PIPE)

inputData = "\n".join(["Der", "Hund", "bellt", "laut",
"."])

(out, err) = proc.communicate(inputData)

print out,
```

External Programs
Process communication

using `communicate`:

- Pro's: simple, no deadlocks
- Con's: slower since all data is copied in memory

manually, via `read/write/...`:

- Pro's: fast
- Con's: complicated and deadlocks possible

Pipeline

- pipes transfer data from the input (pump) through the filters to the output (sink)
- filters process the data, they are fully independent of each other

- Pro's: modular architecture, easy to understand in pieces
- Con's: errors on one step passed on to next steps

Lecture 9: Code and project management

Unit Testing

```
def levenshtein(str1, str2):
    ...

if __name__ == "__main__":
    TESTS = [
        #string1, string2, expected lev. distance
        ("", "abccba", 6),
        ("abcd", "abce", 1),
        ("abcd", "abdc", 2),
        ("apple", "orange", 5)]

    #apply every test
    for (s1, s2, expectedVal) in TESTS:
        assert levenshtein(s1, s2) == expectedVal
```

Counting hapax legomena

```
for token in tokenList:
    tokenCount = 0

    for token2 in tokenList:
        if token2 == token:
            tokenCount += 1

    if tokenCount == 1:
        hapaxCount += 1
```

- python hapax.py data.txt 300: 0.186s
- python hapax.py data.txt 400: 0.316s
- python hapax.py data.txt 500: 0.545s
- python hapax.py data.txt 600: 0.816s

Counting hapax legomena

```
frequencies = defaultdict(int)

for token in tokenList:
   frequencies[token] += 1

for token in frequencies:
   if frequencies[token] == 1:
      hapaxCount += 1
```

- python fhapax.py data.txt 1000: 0.033s
- python fhapax.py data.txt 2000: 0.039s
- python fhapax.py data.txt 4000: 0.049s
- python fhapax.py data.txt 8000: 0.077s

Profiling

→ **Monitoring and "debugging" complexity**

- insert time measurements into the code
  - `time.clock()` / `time.time()` / `time.localtime()` / `time.gmtime()`
- report time or intervals for different parts of code
- understand why the code is running slow
- compare different approaches/methods
- Similar to tracing (but different purpose)

`timeit` module

```
import timeit
code = """for token in tokenList:
    tokenCount = 0
    ..."""

timeit.timeit(code, number=100)
```

- run code a given number of times
- report the average time of execution

cProfile

```
python -m cProfile hapax.py data.txt 400
```

```
  ncalls  tottime  percall  cumtime  percall filename:lineno(function)
       1    0.000    0.000    0.005    0.005 hapax.py:10(loadData)
    3625    0.001    0.000    0.321    0.000 hapax.py:19(countToken)
       1    0.001    0.001    0.327    0.327 hapax.py:29(main)
  [...]
```

```
python -m cProfile fhapax.py data.txt 400
```

```
  ncalls  tottime  percall  cumtime  percall filename:lineno(function)
       1    0.000    0.000    0.004    0.004 fhapax.py:10(loadData)
       1    0.001    0.001    0.005    0.005 fhapax.py:19(main)
  [...]
```

Source control

- allows code and project management
- tracks all changes to a document/project
- developers can work concurrently

- **git**: free and open source distributed version control system
- **gitHub**: web-based git repository
- **bitbucket**
- **gitlab**

- *git* main commands:
  - `clone`
  - `pull`
  - `add`
  - `commit`
  - `push`
  - `status`
  - `branch`

Lecture 10: Complexity, Dynamic Programming

Time complexity

Universität Zürich

- **Hapax legomena**: words (word forms) that occur in a text only once
- Useful why?
  - hapax legomena usually form around half of the vocabulary
    - 44.2% of the Brown corpus vocabulary
  - hapax legomena usually constitute a small portion of the running tokens
    - 1.9% of the Brown corpus
  - a good model for unknown words

Counting hapax legomena

```
hapaxCount = 0

for token in tokenList:
    tokenCount = 0

    for token2 in tokenList:
        if token2 == token:
            tokenCount += 1

    if tokenCount == 1:
        hapaxCount += 1
```

```
frequencies = defaultdict(int)

for token in tokenList:
    frequencies[token] += 1

for token in frequencies:
    if frequencies[token] == 1:
        hapaxCount += 1
```

Tagging

- Sentence with n words given
- One of m tags can be assigned to each word
- Task: find the most likely tag sequence
- Naive solution:
  - generate all possible tag sequences
  - select the one with the highest probability
- Complexity:
  - there is m^n possible tag sequences
  - $\in O(m^n)$, exponential on sentence length

Dynamic programming

A way of optimizing complex tasks and avoiding bad complexity

Applications:

- Syntax parsing
- Sentence alignment
- Tagging

Fibonacci series Up-down, Memoization

Memoization (NOT Memorization)

```
def fib(a):
    memory = {}           # using a dict as a memory

    if a in memory:       # if already computed
        return memory[a] # then retrieve solution from memory
    if (a == 1 or a == 2):
        return 1
    else:
        memory[a] = fib(a-1) + fib(a-2)  # new sub-solution into memory
        return memory[a]

print(fib(40))
```

Fibonacci series Bottom-up

```
def fib(a):
    memory = []
    for i in range(a + 1):
        if i < 3:
            memory.append(1)
        else:
            memory.append(memory[-1] + memory[-2])

    return memory[-1]

print(fib(40))
```

LCS Up-down, recursive

```
#Recursive
#Simplification: Length


def lcs_len(x, y, i, j):



      if (x and y): #for both non-empty strings
         if x[i] == y[j]: #equal last characters
            return lcs_len(x, y, i-1, j-1) + 1
         else:                #different last characters
            return max(lcs_len(x, y, i-1, j), lcs_len(x, y, i, j-1))

       else:           #for strings, one of which is empty
            return 0   #if one of the strings is empty, the LCS


print lcs_len("jumper", "jumps", 6, 5)  # LCS length = 4
```

LCS
Up-down, recursive

```
#Recursive, Memoization (NOT Memorization)
#Simplification: Length

from collections import defaultdict

def lcs_len(x, y, i, j):
   global matrix

   if not matrix[i][j]: #check if solution is already memorized
      if (x and y): #for non-empty strings
         if x[i] == y[j]: #equal last characters
            matrix[i][j] = lcs_len(x, y, i-1, j-1) + 1
         else:                #different last characters
            matrix[i][j] = max(lcs_len(x, y, i-1, j), lcs_len(x, y, i, j-1))


   return matrix[i][j]

matrix = defaultdict(lambda: defaultdict(list)) #initialize memory
print lcs_len("jumper", "jumps", 6, 5)  # LCS length = 4
```

LCS("fear", "fair")
bottom-up

	ø	f	fe	fea	fear
ø	0	0	0	0	0
f	0	f=f 1	e≠f 1	a≠f 1	r≠f 1
fa	0	f≠a 1	e≠a 1	a=a 2	r≠a 2
fai	0	1	1	2	2
fair	0	1	1	2	3

f a r

Levenshtein distance
Bottom-up

	ø	P	L	A	N	E	T
ø	0	1	2	3	4	5	6
P	1	0	1	2	3	4	5
A	2	1	1	1	2	3	4
P	3	2	2	2	2	3	4
E	4	3	3	3	3	2	3
R	5	4	4	4	4	3	3

= min(3 + 1, 3 + 1, 2 + 1), 2 + 1 because 'T' ≠ 'R'

Levenshtein distance
optimal path

	ø	P	L	A	N	E	T
ø	**0**	1	2	3	4	5	6
P	1	**0**	**1**	2	3	4	5
A	2	1	1	**1**	2	3	4
P	3	2	2	2	**2**	3	4
E	4	3	3	3	3	**2**	3
R	5	4	4	4	4	3	**3**

Levenshtein distance

- Why a fail?
  - locally motivated (greedy) optimization runs a danger of local optima
  - every intermediate step has an influence on later steps
  - its influence cannot be assessed locally
- Which is why
  - the global solution is to fill the whole matrix without fixing any decisions
  - at the end it will be clear, which path is optimal overall

Bottom-up

```
from collections import defaultdict

def lev(xs, ys):
   matrix = defaultdict(lambda: defaultdict(int))

   for i, x in enumerate([None] + list(xs)):
      for j, y in enumerate([None] + list(ys)):
         if i > 0 and j > 0:
            replCost = (0 if x == y else 1) + matrix[i-1][j-1]
            insCost = matrix[i-1][j] + 1
            delCost = matrix[i][j-1] + 1
            matrix[i][j] = min(replCost, insCost, delCost)
         elif i > 0:
            matrix[i][j] = i
         else:
            matrix[i][j] = j
   return matrix[i][j]

print lev("planet", "paper")
```

complexity: $O(n \cdot m)$

Top-down + Memoisation

```
def lev(xs, ys):
   global matrix

   if (not matrix[xs][ys]): # recompute if needed
      if (xs and ys):
         replCost = lev(xs[:-1], ys[:-1]) + (0  if xs[-1] == ys[-1] else 1)
         delCost = lev(xs[:-1], ys) + 1
         insCost = lev(xs, ys[:-1]) + 1

         matrix[xs][ys] = min(replCost, insCost, delCost)
      elif (xs):
         matrix[xs][ys] = len(xs)
      else:
         matrix[xs][ys] = len(ys)

   return matrix[xs][ys]

from collections import defaultdict
matrix = defaultdict(lambda: defaultdict(int))

print lev("planet", "paper") # 3
```

complexity: $O(n \cdot m)$

Viterbi Algorithm

$$\boldsymbol{t} = \text{argmax}_{\boldsymbol{t'}}\ p(\boldsymbol{w}|\boldsymbol{t'}) = \text{argmax}_{\boldsymbol{t'}}\ \Pi_i\, p(w_i|t_i') \cdot p(t_i'|t_{i-1})$$

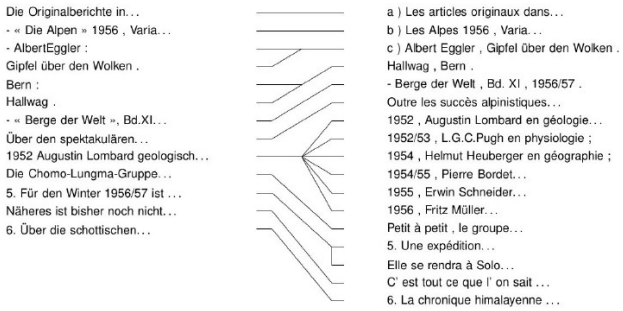
- For calculating $P(t_i|\ t_{i-1})$ only have to go over 1 previous tag
- Dynamic programming:
  - sub-tasks: probabilities for tag sequences from *1* to *i-1*
  - approach: for each candidate of the current tag find optimal predecessor
    - in other words -- let us not decide on a tag, let us evaluate all the options and decide in the end

Viterbi Algorithm

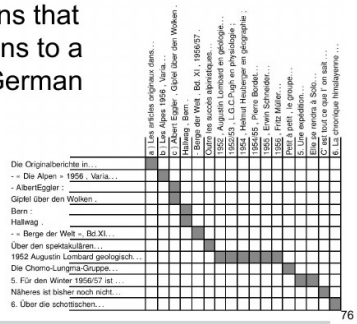
	The	complex	houses	soldiers
Det	p(The | Det) × p(Det | <s>)	p(complex | Det) × p(Det | Verb) × prev	p(houses | Det) × p(Det | Verb) × prev	p(soldiers | Det) × p(Det | Verb) × prev
Adj	p(The | Adj) × p(Adj | <s>)	p(complex | Adj) × p(Adj | Det) × prev	p(houses | Adj) × p(Adj | Det) × prev	p(soldiers | Adj) × p(Adj | Adj) × prev
Verb	p(The | Verb) × p(Verb | <s>)	p(complex | Verb) × p(Verb | Noun) × prev	p(houses | Verb) × p(Verb | Noun) × prev	p(soldiers | Verb) × p(Verb | Noun) × prev
Noun	p(The | Noun) × p(Noun | <s>)	p(complex | Noun) × p(Noun | Det) × prev	p(houses | Noun) × p(Noun | Adj) × prev	p(soldiers | Noun) × p(Noun | Verb) × prev

Sentence alignment

- Given:
  - text (list of sentences) in one (source) language
  - its translation into another (target) language
- Find:
  - for every source text sentence (sentences) find the target text sentence (sentences) that correspond to it



- Alignment shown on the matrix:
- *3 -- (3,4)* here means that French snt. #3 aligns to a concatenation of German snts. #3 & #4



Sentence alignment via Dynamic Programming

Alignment likelihood can depend on

- sentence length (Gale & Church 1993: **Vanilla aligner**)
  https://www.youtube.com/watch?v=_4InyoC3mtQ
- lexical information (Chen 1993)
- both (Moore 2002; Varga et al. 2005: **HunAlign**)
- advanced (Sennrich & Volk 2010: **BleuAlign**)

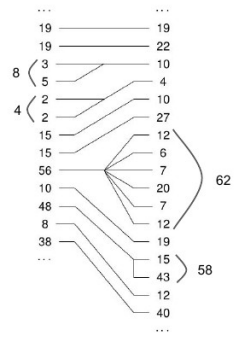
Vanilla aligner

- Principle:
  - longer sentences in one language should correspond to longer sentences in another language
- Likelihood/weight per alignment pair
  - depends on the length ratio of the pair
  - is multiplied by fixed "penalty" values for 1-0/1-1/...

We see...

Die Originalberichte in...	19	19	a) Les articles originaux dans...
- « Die Alpen » 1956 , Varia...	19	22	b) Les Alpes 1956 , Varia...
- AlbertEggler :	3	10	c) Albert Eggler , Gipfel über den Wolken .
Gipfel über den Wolken .	5	4	Hallwag , Bern .
Bern :	2	10	- Berge der Welt , Bd. XI , 1956/57 .
Hallwag .	2	27	Outre les succès alpinistiques...
- « Berge der Welt », Bd.XI...	15	12	1952 , Augustin Lombard en géologie...
Über den spektakulären...	15	6	1952/53 , L.G.C.Pugh en physiologie ;
1952 Augustin Lombard geologisch...	56	7	1954 , Helmut Heuberger en géographie ;
Die Chomo-Lungma-Gruppe...	10	20	1954/55 , Pierre Bordet...
5. Für den Winter 1956/57 ist ...	48	7	1955 , Erwin Schneider...
Näheres ist bisher noch nicht...	8	12	1956 , Fritz Müller...
6. Über die schottischen...	38	19	Petit à petit , le groupe...
		15	5. Une expédition...
		43	Elle se rendra à Solo...
		12	C' est tout ce que l' on sait ...
		40	6. La chronique himalayenne ...

Vanilla aligner aligns:



Syntax for statistical machine translation: pre-reordering

- Input: Ich werde Ihnen die entsprechenden Anmerkungen aushändigen, damit Sie das eventuell bei der Abstimmung übernehmen können.
- Gloss: I will to you the corresponding comments pass on, so that you them perhaps in the vote adopt can.

- **Reordered** input: Ich werde aushändigen Ihnen die entsprechenden Anmerkungen, damit Sie können übernehmen das eventuell bei der Abstimmung.
- Gloss: I will pass on to you the corresponding comments, so that you can adopt them perhaps in the vote.

Language properties

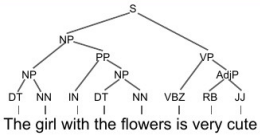
- A sentence has no limited length:
  - Mark gave a lecture
  - The students realized that Mark gave a lecture
  - Despite all Mark's efforts the students realized that Mark gave a lecture
  - According to the local news, despite all Mark's efforts the students realized that Mark gave a lecture
  - Laura was happy to announce that according to the local news, despite all Mark's efforts the students realized that Mark gave a lecture
- Potentially an infinite set of sentences
- that we want to handle using a finite model

- Ambiguity:
  - (The viking) killed (the pirate) (with a sword) vs
  - (The viking) killed (the pirate (with a sword))

- Can we ignore structure and handle it on the word sequence level?

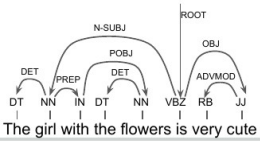
  - e.g. “The girl with **the flowers is** cute”
  - A 3-gram-based spellchecker will likely suggest: “did you mean ‘**the flowers are**’ ”
  - e.g. translating “**Legen** sie noch etwas **zu**”:
    - Legen = lay? lie? put?...
    - Zulegen = add

Constituency vs. dependency



(((The $_{DT}$ girl $_{NN}$) $_{NP}$ (with $_{IN}$ (the $_{DT}$ flowers $_{NN}$) $_{NP}$) $_{PP}$) $_{NP}$ (is $_{VBZ}$ (very $_{RB}$ cute $_{JJ}$) $_{AdjP}$) $_{VP}$) $_{S}$

(((The$_{DET}$ girl$_{NN}$) (with$_{IN}$ (the$_{DET}$ flowers$_{NN}$)$_{POBJ}$)$_{PREP}$)$_{N\text{-}SUBJ}$ is$_{VBZ}$ (very$_{ADVMOD}$ cute$_{JJ}$)$_{OBJ}$)$_{ROOT}$



Grammar

- A set of rules describing which sentences are acceptable for a particular language
- Context-free grammars (CFG): classical approach to consistency syntax
  - even though dependency syntax can also be described with the same kind of grammar
- Key point: describe the permitted structures one sub-tree at a time

Context-free grammar

Example:

- S → NP VP
- NP → N
- NP → JJ NP
- VP → V NP
- VP → V
- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep

Sentences:

- dogs chase cats
- cats eat
- big cats eat big dogs
- big black dogs sleep

But also

- black dogs sleep big cats
- black black big black dogs eat

Grammar:

- a set of terminal symbols (dogs, stuff, big, ...)
- a set of non-terminal symbols (N, JJ, NP, ...)
- a set of rules to turn non-terminals into terminals and other non-terminals
- a (set of) starting symbol(s)

Parsing

Task: given a grammar and a sentence:

- is the sentence part of the given grammar?
- what is its structure/rule sequence that would generate it starting from S?

Parsing

- Is the sentence parseable with the given grammar, a naive solution:
  - generate all trees that the grammar can generate
  - check if given sentence is among them
- Why naive?
  - the set of sentences that the grammar accepts is in general exponential
  - due to recursion it can also be infinite

Chart parsing

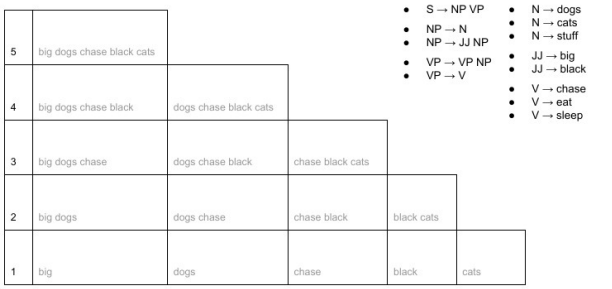
- A solution: dynamic programming
- Find partial trees only once, store in a chart and reuse
- Cocke-Younger-Kasami (CYK) algorithm:
  - also called CKY algorithm
  - bottom-up parsing
  - list of chart entries per word subsequence

CYK parsing

- Requirement: the grammar must be in the "Chomsky normal form" (CNF), which allows only 2 kinds of rules:
  - A → a (a non-terminal into a terminal)
  - A → BC (a non-terminal into 2 non-terminals)
- Any grammar can be put into CNF, e.g.
  - **original rule:** **CNF rules:**

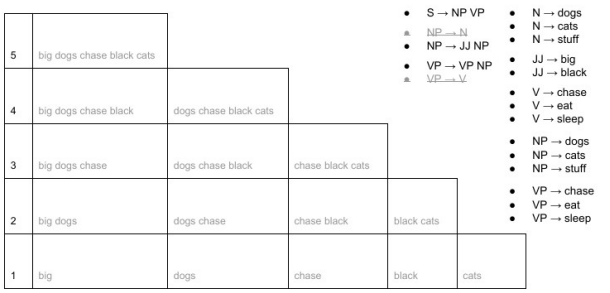
VP → V NP PP	VP → V VP_1
	VP_1 → NP PP
NP → little N	NP → ADJ_1 N
	ADJ_1 → little

CYK parsing



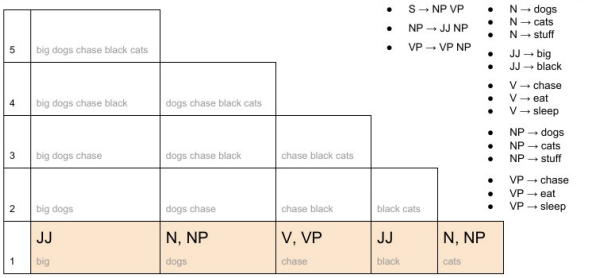
- A cell for every subsequence (total num: len^2)

CYK parsing



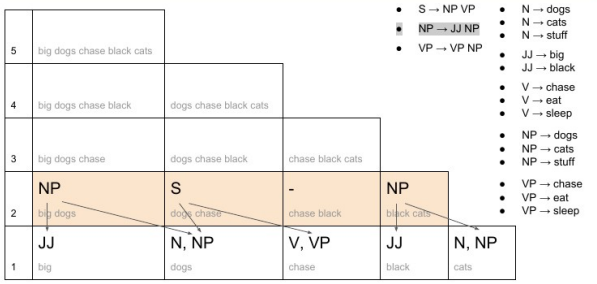
- A cell for every subsequence (total num: len^2)

CYK parsing: step 1



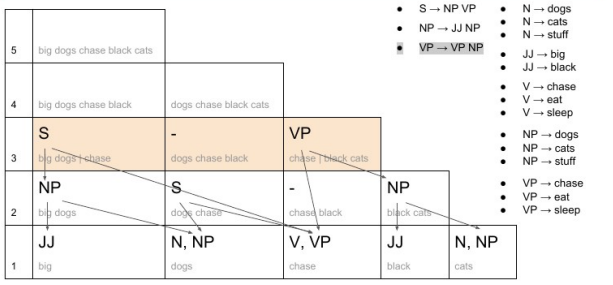
- Find a non-terminal$_{\text{(PoS-tag)}}$ for each terminal$_{\text{(token)}}$

CYK parsing: step 2



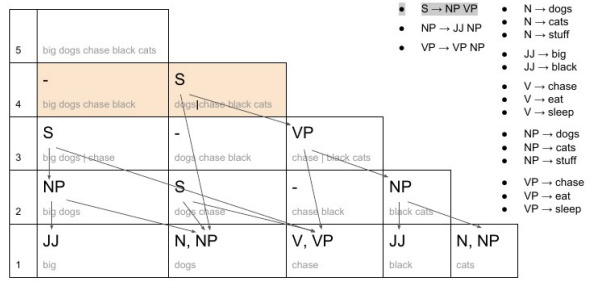
- `for i in 2..N:` try splitting each i-word sequence into two sub-sequences (i-1 different ways)
- check if both sub-sequences have at least 1 parse available in the table

CYK parsing: step 2



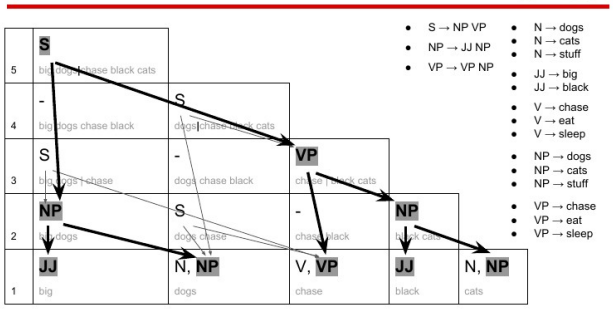
- `for i in 2..N:` try splitting each i-word sequence into two sub-sequences (i-1 different ways)
- check if both sub-sequences have at least 1 parse available in the table

CYK parsing: step 2



- `for i in 2..N:` try splitting each i-word sequence into two sub-sequences (i-1 different ways)
- check if both sub-sequences have at least 1 parse available in the table

CYK parsing: ready!



- back-track the tree(s) starting from the top cell

Parsing Pseudocode

```
def parse(grammar, sentence):
    cnfGrammar = getCNF(grammar)

    memory = dict(subseq) of lists of subtrees
        # subtree: (
        #   NonTerminal,
        #   ( LeftSubSequence, LeftNonTerminal),
        #   ( RightSubSequence, RightNonTerminal) )

    for token in sentence:
        memory[token] = list of possible PoS tags for token
            # or PoS-tagging with 1 tag per token
```

Parsing Pseudocode

```
def parse(grammar, sentence):
    …(continued)…

    for sequenceLen in (2, …, len(sentence)):
        for subSequence in (subSequences of sentence with length sequenceLen):
            for splitPoint in (1, …, sequenceLen - 1):
                (leftSubSeq, rightSubSeq) = split subSequence on splitPoint
                for LeftNT in memory[leftSubSeq] NonTerminals:
                    for RightNT in memory[rightSubSeq] NonTerminals:
                        if cnfGrammar has a rule X → LeftNT RightNT:
                            add it to the list under memory[subSequence]

    the result(s) are in memory[sentence]
```

Complexity? $O(n^3)$

Parsing in NLTK

```
import nltk

grammar1 = nltk.CFG.fromstring("""
  S -> NP VP
  VP -> V NP | V NP PP
  PP -> P NP
  V -> "saw" | "ate" | "walked"
  NP -> "John" | "Mary" | "Bob" | Det N | Det N PP
  Det -> "a" | "an" | "the" | "my"
  N -> "man" | "dog" | "cat" | "telescope" | "park"
  P -> "in" | "on" | "by" | "with"
  """)

sent = "Mary saw Bob".split()
parser = nltk.parse.chart.BottomUpChartParser(grammar1)
for tree in parser.parse(sent):
   print tree
#(S (NP Mary) (VP (V saw) (NP Bob)))
```

Ambiguity

- How to decide between several possible parses?
- One solution: probabilistic CFG (PCFG)

CFG:

- S → NP VP
- NP → N
- NP → JJ NP
- VP → VP NP
- VP → V
- N → dogs
- NP → dogs

PCFG:

- S → NP VP *(p = 1)*
- NP → N *(p = 0.6)*
- NP → JJ NP *(p = 0.4)*
- VP → VP NP *(p = 0.32)*
- VP → V *(p = 0.68)*
- N → dogs *(p = 0.17)*
- NP → dogs *(p = 0.17)*

PCFG Parsing

Algorithm same as CKY, but with a Viterbi twist

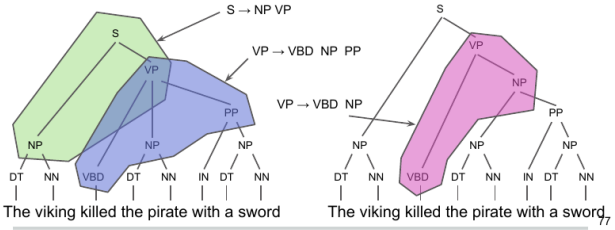
- each possible parse 1 step down is kept with its probability
- probabilities are multiplied
- link to the likeliest predecessor is kept
- back-tracking follows the most likeliest predecessor links

PCFG Parsing in NLTK

```
import nltk

grammar2 = nltk.PCFG.fromstring("""
  S -> NP VP              [1.0]
  VP -> V NP              [0.217]
  VP -> V NP PP           [0.53]
  ...
  """)

sent = "Mary saw Bob".split()

viterbi_parser = nltk.ViterbiParser(grammar2)
for tree in viterbi_parser.parse(sent):
   print tree
```

Grammar Learning

- Q: where do we get a grammar from?
- A: we learn it!
  - from a treebank = syntactically annotated corpus
  - subtree = production



Grammar Learning

```
from nltk.corpus import treebank

treebank.sents()[314]
#['The', 'offer', 'is', 'being', 'launched', …]

treebank.parsed_sents()[314]
#Tree('S', [Tree('NP-SBJ-45', [Tree('DT', ['The']), Tree
('NN', ['offer'])]), Tree('VP', [Tree('VBZ', ['is']), Tree
('VP', [Tree('VBG', ['being']), …)

treebank.parsed_sents()[314].productions()[4]
#VP -> VBZ VP

treebank.parsed_sents()[314].productions()[0].lhs()
#VP

treebank.parsed_sents()[0].productions()[0].rhs()
#(VBZ, VP)
```

```
import nltk
grammar1 = nltk.CFG.fromstring("""
S -> NP VP
VP -> V NP | V NP PP
PP -> P NP
V -> "saw" | "ate" | "walked"
NP -> "John" | "Mary" | "Bob" | Det N | Det N PP
Det -> "a" | "an" | "the" | "my"
N -> "man" | "dog" | "cat" | "telescope" | "park"
P -> "in" | "on" | "by" | "with"
""")
sent = "Mary saw Bob".split()
parser = nltk.parse.chart.BottomUpChartParser(grammar1)
for tree in parser.parse(sent):
    print tree #(S (NP Mary) (VP (V saw) (NP Bob)))


from nltk.corpus import treebank
from nltk.grammar import CFG, Nonterminal
tbank_productions = set(production for sent in
                    treebank.parsed_sents() for production in
                    sent.productions())
tbank_grammar = CFG(Nonterminal('S'), list(tbank_productions))
print tbank_grammar
```

Weighed Grammar Learning

Learning the weights/probabilities:

- $p(A \rightarrow \gamma) = p(\gamma \mid A)$
  (must sum to 1 over all γ for any A)
- $p(A \rightarrow \gamma) = \text{count}(A \rightarrow \gamma) / \text{count}(A)$

(γ: any sequence of terminals/non-terminals)

Weighed Grammar Learning

```
import nltk
from nltk.corpus import treebank
from nltk.grammar import Nonterminal

tbank_production_list = [production for sent in
       treebank.parsed_sents() for production in
       sent.productions()]

tbank_prob_grammar = nltk.induce_pcfg(Nonterminal('S'),
       tbank_production_list)
```

```
import nltk
from nltk.corpus import treebank
from nltk.grammar import Nonterminal
tbank_production_list = [production for sent in
            treebank.parsed_sents() for production in
            sent.productions()]
tbank_prob_grammar = nltk.induce_pcfg(Nonterminal('S'),
                                      tbank_production_list)
print tbank_prob_grammar
```

```
Grammar with 21763 productions (start state = S)
  VBZ -> 'cites' [0.00141176]
  VBD -> 'spurned' [0.000328623]
  PRN -> , ADVP-TMP , [0.00668896]
  NNP -> 'ACCOUNT' [0.00010627]
  JJ -> '36-day' [0.000171409]
  NP-SBJ-2 -> NN [0.00606061]
  JJ -> 'unpublished' [0.000342818]
  NP-SBJ-1 -> NNP [0.0389484]
  JJ -> 'elusive' [0.000171409]
  NNS -> 'Lids' [0.000165371]
……
```

Computational Semantics

Propositional logic

- Basic statements are represented with *propositional symbols*:
  - "John is chasing Mary" = P
  - "Mary is running" = Q
- They can be combined:
  - John is chasing Mary and she (Mary) is running: P & Q
- Using the logical connectives (boolean operators)
  - not (-), and (&), or (|),
    therefore (→), equivalence (↔)

Logical Connectives

- Negation (not) -
  -X = True if X is False (=the reverse value of
- Conjunction (and) &
  X & Y = True only if both X and Y are True
- Disjunction (or) |
  X | Y = True if at least one of X and Y is True
- Implication (if …, then ...) →
  X → Y = False only when X is True and Y is False
- Equivalence (if and only if) ↔
  X ↔ Y = True if X and Y are both True or False

Logical Connectives

Conjunction (X & Y):

	X = False	X = True
Y = False	False	False
Y = True	False	True

Disjunction (X | Y):

	X = False	X = True
Y = False	False	True
Y = True	True	True

Implication (X → Y):

	X = False	X = True
Y = False	True	False
Y = True	True	True

Equivalence (X ↔ Y):

	X = False	X = True
Y = False	True	False
Y = True	False	True

Propositional Logic @ NLTK:

```
# NLTK can process logical expressions into subclasses of Expression object

>>> from nltk.sem.logic import LogicParser
>>> lp = LogicParser()
>>> lp.parse('-(P & Q)')
<NegatedExpression -(P & Q)>

>>> lp.parse('P & Q')
<AndExpression (P & Q)>

>>> lp.parse('P | (R -> Q)')
<OrExpression (P | (R -> Q))>

>>> lp.parse('P <-> -- P')
<IffExpression (P <-> --P)>
```

Valuation

- Assigning values:

```
>>> val = nltk.Valuation([('P', True), ('Q', True), ('R', False)])
>>> val['P']
True
```

- Evaluating expressions:

```
>>> dom = set([])
>>> g = nltk.Assignment(dom)
>>> m = nltk.Model(dom, val)

>>> print m.evaluate('(P & Q)', g)
True
>>> print m.evaluate('-(P & Q)', g)
False
>>> print m.evaluate('(P | R)', g)
True
```

Arguments

- Starting with some assumptions
  - propositional symbol expressions, e.g.
    - *P* (fish live in water)
    - *Q* (fish can fly)
    - $P \rightarrow -Q$
- Can we reach a conclusion?
  - e.g. $-Q$
- Argument: $[A_1, ..., A_n] / C$, where $A_1, ..., A_n$ are **assumptions**

An argument is **valid** if there is no possible situation in which its premises are all true and its conclusion is not true.

First-order logic

- Propositions are analyzed into predicates and arguments
  - read(John)
- combined with boolean operators
  - read(John, book) & falling_asleep(John)

Quantifiers

- Used with variables
- Existence quantifier: ∃
  - $\exists x$ means: for some *x* it is true that...
  - e.g. $\exists x.$(person(x) ^ like(*x*, waffles))
  - meaning: there are some that like waffles
  - NLTK:

```
'exists x.(person(x) & like(x, waffles))'
```

Quantifiers

- Used with variables
- Universal quantifier: ∀
  - $\forall x$ means: for all *x*…
  - e.g. $\forall x.(person(x) \rightarrow$ lie(x))
  - meaning: everybody lies
  - NLTK: `'all x.(person(x) → lie(x))'`

Lexical semantics

Words meanings and their relations

- word meanings/senses
  - bank → financial institution, river shore,…
  - crane → construction machine, bird,…
  - …
- word relations
  - synonyms/antonyms, hyponyms/hyperonyms, …
  - e.g. WordNet
    - http://www.nltk.org/howto/wordnet.html

WordNet @ NLTK

- Lemma → senses:

```
from nltk.corpus import wordnet as wn

wn.synsets('bank')
[Synset('bank.n.01'), Synset('depository_financial_institution.n.01'),
Synset('bank.v.01'),...]
```

- Senses for a given PoS-tag only:

```
wn.synsets('bank', pos=wn.VERB)
[Synset('bank.v.01'), Synset('bank.v.02'),...]
```

- Sense definitions:

```
print wn.synsets('bank')[1].definition()
Synset('depository_financial_institution.n.01') a financial institution
that accepts deposits and channels the money into lending activities
```

```
from nltk.corpus import wordnet as wn
print "Senses"
print wn.synsets('bank')
print "\nSenses for a given PoS-tag only"
print wn.synsets('bank', pos=wn.VERB)
print "\n"
print wn.synsets('bank')[1]
print "\n"
print wn.synsets('bank')[1].definition()
print "\n"
print wn.synsets('bank')[1].lemmas()
print "\n"
print wn.synsets('bank')[1].lemma_names()
print "\n"
print wn.synsets('bank')[0]
print wn.synsets('bank')[0].definition()

print "\nother languages"
#print wn.langs()
print "\n"
#print wn.synsets('fromage')
print "\n"
#print wn.synsets('fromage', lang='fra')
```

```
print "\n"
dog = wn.synset('dog.n.01')
print dog.hypernyms()
print "\n"
print dog.hyponyms()
print "\n"
print dog.member_holonyms()
print "\n"
print dog.root_hypernyms()
print "\n"
good = wn.synset('good.a.01' )
print "\n"
print good.lemmas()
print "\n"
print good.lemmas()[0].antonyms()

dog = wn.synset('dog.n.01')
cat = wn.synset('cat.n.01')
print dog.path_similarity(cat)
print wn.path_similarity(dog, cat)
for s in wn.synsets('wolf'):
    print
    print s.lemma_names()
    print s.definition()
    print dog.lowest_common_hypernyms(s)
    print dog.path_similarity(s)
print dog.lowest_common_hypernyms(cat)
print dog.lowest_common_hypernyms(wn.synset('wolf.n.01'))
print dog.lowest_common_hypernyms(wn.synset('wolf.n.03'))
```

Word sense disambiguation

- another computational semantics task
- given a sentence containing words with multiple meanings, determine which meanings are used
  - e.g. “Today I am going to rob a bank”
  - bank → bank.n.01 / depository_financial_institution.n.01 / etc.?
- different from tagging: how?

  different number of senses per word

Word sense disambiguation

Can the word senses be labeled independently?

- words in sentence and their sentence are not independent
- choice of one sense can influence another:

```
time
[Synset('time.n.01'), Synset('fourth_dimension.n.01'), Synset('clock.v.
01'),...]

flies
[Synset('flies.n.01'), Synset('fly.n.01'), Synset('fly.v.01'),...]

like
[Synset('like.n.01'), Synset('like.n.02'), Synset('wish.v.02'), Synset
('like.v.02'), Synset('comparable.s.02'),...]
```

- in practice this can be ignored

Word Sense Disambiguation
Lesk Algorithm

1. go through the definitions of all senses of the word
2. check the overlap between the words in the definition and in the context
3. select the sense with the highest overlap
   - in case none of the senses have an overlap, select the most frequent sense of the word (first in WordNet)

- simplified version: perform the algorithm independently for each word in sentence

```
from nltk.wsd import lesk
from nltk import word_tokenize

sent = word_tokenize("I went to the bank to deposit money.")
word = "bank"
pos = "n"

print(lesk(sent, word, pos)) #Synset('savings_bank.n.02')


for s in wn.synsets('bank'):
    print
    print s.lemma_names()
    print s.definition()
for s in wn.synsets('deposit'):
    print
    print s.lemma_names()
    print s.definition()
for s in wn.synsets('money'):
    print
    print s.lemma_names()
    print s.definition()
```

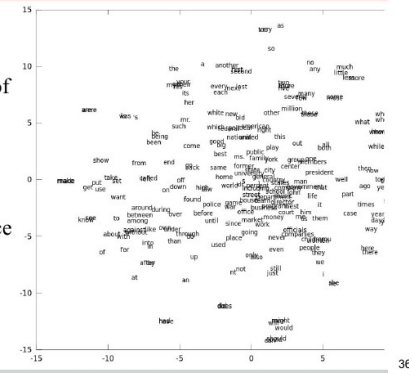
Word Sense Disambiguation
Transformation-based Learning

1. learn to disambiguate from a sense-annotated corpus
2. tag the corpus with a baseline tagger (e.g. most frequent sense)
3. learn transformations
   e.g.
   if current_sense = 'bank.n.01' (river bank)
   and sentence includes 'money'
   then change sense to 'depository_financial_institution.n.01'
4. when doing WSD, apply the baseline and then the learned transformations

Word Embeddings

Word embeddings: vector representation of words

Vector space models: represent words in a continuous vector space

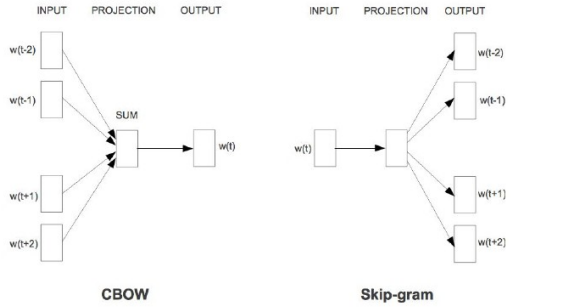


Word Embeddings
Approaches

1. Count-based methods
   How often some word co-occur with its neighbor words
   e.g. Latent Semantic Analysis

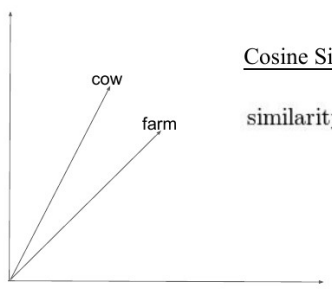
2. Predictive Methods
   Predict a word from its neighbors
   e.g. Neural Probabilistic Language Models
   e.g. Word2Vec [Mikolov et al., 2013]

Predictive Method: Word2Vec
Model Architectures



Predictive Method: Word2Vec
Vector Similarity

How to compute similarity between words?

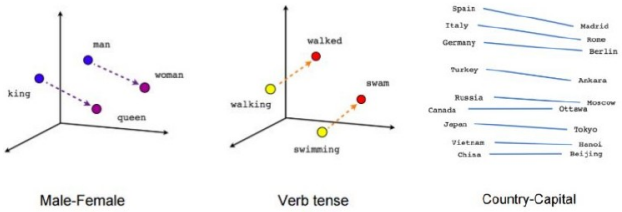


Cosine Similarity

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Predictive Method: Word2Vec
Semantic Relationships

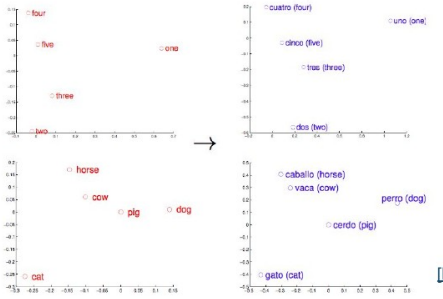
"What is the word that is similar to *king* in the same sense as *woman* is similar to *man*?"



X = vector("*woman*") - vector("*man*") + vector("*king*")
Queen is the word in the vectors space closest to X

Predictive Method: Word2Vec Semantic Relationships

Exploiting Similarities among Languages



[Mikolov, et al., 2013]

Predictive Method: Word2Vec

Words with several meanings:

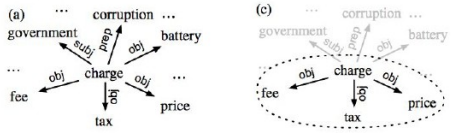
e.g. *Charge* appears in the expressions:

- *Charge* a fee
- *Charge* a battery

Problem: Vectors do not reflect charge/impose as near-synonyms.

Predictive Method: Word2Vec

1. Word embeddings & external resources (e.g. WordNet) [Rothe S. and Schütze H., 2015] http://www.cis.lmu.de/~sascha/AutoExtend/
2. Contextualized vectors [Thater et al., 2011]



Information Extraction

Information extraction

- Acquisition of structured data from unstructured text
- not to confuse with information retrieval
  - IR finds documents within a collection, that are relevant for a specific information need
    - input: query
    - output: set of relevant documents
  - IE actually "looks into" the documents
    - input: text
    - output: structured info from it

Temporal expr. detection/analysis

In 1854 Matthew Perry, a U.S. Navy commodore performed a blockade of Japan in the Yokohama bay in order to force it to end its policy of isolation, which has lasted for more than 200 years. As a result diplomatic relations were established between the United States and the Japanese Empire. In five years, similar treaties were signed with several other Western countries.

what: ?
who: ?
whom: ?
where: ?
when: ?

named entity:
person / location / geo-political / ...

temporal expression:
absolute / relative / duration

(Co-)reference resolution

In 1854 Matthew Perry, a U.S. Navy commodore performed a blockade of Japan in the Yokohama bay in order to force it to end its policy of isolation, which has lasted for more than 200 years. As a result diplomatic relations were established between the United States and the Japanese Empire. In five years, similar treaties were signed with several other Western countries.

what: ?
who: ?
whom: ?
where: ?
when: ?

reference resolution:
U.S. = United States
Japan = the Japanese Empire

Anaphora resolution

In 1854 Matthew Perry, a U.S. Navy commodore performed a blockade of Japan in the Yokohama bay in order to force it to end its policy of isolation, which has lasted for more than 200 years. As a result diplomatic relations were established between the United States and the Japanese Empire. In five years, similar treaties were signed with several other Western countries.

what: ?
who: ?
whom: ?
where: ?
when: ?

Relation detection and classification

In 1854 Matthew Perry, a U.S. Navy commodore performed a blockade of Japan in the Yokohama bay in order to force it to end its policy of isolation, which has lasted for more than 200 years. As a result diplomatic relations were established between the United States and the Japanese Empire. In five years, similar treaties were signed with several other Western countries.

what: ?
who: ?
whom: ?
where: ?
when: ?

relation detection:
who is what of/to whom/what?
family / employment / part-whole / membership / geo-spatial..

Event detection and classification

In 1854 Matthew Perry, a U.S. Navy commodore performed a blockade of Japan in the Yokohama bay in order to force it to end its policy of isolation, which has lasted for more than 200 years. As a result diplomatic relations were established between the United States and the Japanese Empire. In five years, similar treaties were signed with several other Western countries.

what: ?
who: ?
whom: ?
where: ?
when: ?

event detection and classification:
detect and classify events, binding the named entities and others

Template filling

In 1854 Matthew Perry, a U.S. Navy commodore performed a blockade of Japan in the Yokohama bay in order to force it to end its policy of isolation, which has lasted for more than 200 years. As a result diplomatic relations were established between the United States and the Japanese Empire. In five years, similar treaties were signed with several other Western countries.

what: policy of isolation **where:** Japan
when: (period) before 1654 - 1854

who: Matthiew Perry **organization:** U.S. Navy
relation: member **type:** commodore

what: blockade **who:** Matthew Perry / U.S.
whom: Japan **when:** 1854
where: Yokohama bay

what: establishment of diplomatic relations
who: Japan **when:** 1854
who: U.S.

what: establishment of diplomatic relations
who: Japan **when:** (by) 1859
who: other Western countries

Information Extraction

- **(Chunking)**
- **Named entity recognition**
- Reference and anaphora resolution
- **Relation detection/classification**
- **Temporal expression detection/analysis**
- Event detection/classification
- Template filling

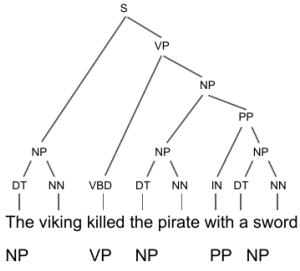
Chunking

- Detection of multi-token sequences, e.g.
  - New York
  - The big bad wolf
- Approaches:
  - based on manually defined regular expressions
  - automatically learned taggers/classifiers

Chunking vs consituency parsing

Parsing:
- Deeper Tree
- Nested Structures + Recursive Structures
- More information

Chunking:
- Shallow
- No nested Structures
- Less information (but often still enough)
- Lighter
- Often limited to one Chunk Type



RegExp Chunking

- define regular expressions of PoS-tags
- matching sequences = chunks

```
>>> sentence = [("the", "DT"), ("little", "JJ"), ("yellow", "JJ"),
... ("dog", "NN"), ("barked", "VBD"), ("at", "IN"),  ("the", "DT"), ("cat", "NN")]

>>> grammar = "NP: {<DT>?<JJ>*<NN>}"

>>> cp = nltk.RegexpParser(grammar)
>>> result = cp.parse(sentence)
>>> print result
(S
  (NP the/DT little/JJ yellow/JJ dog/NN)
  barked/VBD
  at/IN
  (NP the/DT cat/NN))
```

RegExp Ch_i_nking

- define regular expressions of PoS-tags
- matching sequences **excluded** from chunks

```
grammar = r"""
  NP:
    {<.*>+}          # Chunk everything
    }<VBD|IN>+{      # Chink sequences of VBD and IN
  """
sentence = [("the", "DT"), ("little", "JJ"), ("yellow", "JJ"),
       ("dog", "NN"), ("barked", "VBD"), ("at", "IN"),  ("the", "DT"), ("cat", "NN")]
cp = nltk.RegexpParser(grammar)

>>> print cp.parse(sentence)
(S
  (NP the/DT little/JJ yellow/JJ dog/NN)
  barked/VBD
  at/IN
  (NP the/DT cat/NN))
```

Chunking as Tagging

- The IOB format
  - Inside/outside/beginning
  - token-level annotation
- Possible to sub-specify chunk types
  - NP / VP / …
  - tags turn into O / B-NP / I-NP / B-VP / …

In	O
1854	O
Matthew	B
Perry	I
,	O
a	B
U.S.	I
Navy	I
commodore	I
performed	O
a	B
blockade	I
of	O
Japan	B
in	O
the	B
Yokohama	I
bay	I
.	O

Chunking as Tagging

- Take a IOB-tagged corpus
- Train a tagger/classifier on it
  - IOB / IOB with chunk types

Chunking in NLTK

- CoNLL-2000 corpus:
  - NP / VP / PP

```
>>> from nltk.corpus import conll2000
>>> print conll2000.chunked_sents('train.txt')[99]
(S
  (PP Over/IN)
  (NP a/DT cup/NN)
  (PP of/IN)
  (NP coffee/NN)
  ,/,
  (NP Mr./NNP Stone/NNP)
  (VP told/VBD)
  (NP his/PRP$ story/NN)
  ./.)
```

```
>>> print conll2000.chunked_sents('train.txt',
chunk_types=['NP'])[99]
(S
  Over/IN
  (NP a/DT cup/NN)
  of/IN
  (NP coffee/NN)
  ,/,
  (NP Mr./NNP Stone/NNP)
  told/VBD
  (NP his/PRP$ story/NN)
  ./.)
```

Chunking in NLTK, RegExp:

```
>>> from nltk.corpus import conll2000
>>> test_sents = conll2000.chunked_sents('test.txt', chunk_types=['NP'])
>>> cp = nltk.RegexpParser(r"NP: {<[CDJNP].*>+}")
>>> print cp.evaluate(test_sents)
ChunkParse score:
    IOB Accuracy:  87.7%
    Precision:     70.6%
    Recall:        67.8%
    F-Measure:     69.2%
```

Chunking in NLTK, tagging:

```
>>> from nltk.corpus import conll2000
>>> from nltk.tag.hmm import HiddenMarkovModelTagger as HmmTagger
>>> train_sents = [ [(t, c if c[-2:] == 'NP' else 'O') for w, t, c in snt]
          for snt in conll2000.iob_sents('train.txt')]
>>> defTagger = nltk.DefaultTagger('O')
>>> uniTagger = nltk.UnigramTagger(train_sents, backoff=defTagger)
>>> biTagger = nltk.BigramTagger(train_sents, backoff=uniTagger)
>>> hmm_tagger = HmmTagger.train(train_sents)
>>> test_sents_iob = [ [(t, c if c[-2:] == 'NP' else 'O') for w, t, c in snt]
          for snt in conll2000.iob_sents('test.txt')]
>>> print defTagger.evaluate(test_sents_iob)
0.43436688688604175
>>> print uniTagger.evaluate(test_sents_iob)
0.8321126284906178
>>> print biTagger.evaluate(test_sents_iob)
0.9341663676467484
>>> print hmm_tagger.evaluate(test_sents_iob)
0.9360449163096017
```

```
import nltk
from nltk.corpus import conll2000
from nltk.tag.hmm import HiddenMarkovModelTagger as HmmTagger
test_sents = conll2000.chunked_sents('test.txt',
                                     chunk_types=['NP'])
print test_sents[:10]
cp = nltk.RegexpParser(r"NP: {<[CDJNP].*>+}")
print cp.evaluate(test_sents)
train_sents = [ [(t, c if c[-2:] == 'NP' else 'O')
                for w, t, c in snt]
                for snt in conll2000.iob_sents('train.txt')]
print train_sents[:10]
defTagger = nltk.DefaultTagger('O')
uniTagger = nltk.UnigramTagger(train_sents, backoff=defTagger)
biTagger = nltk.BigramTagger(train_sents, backoff=uniTagger)
hmm_tagger = HmmTagger.train(train_sents)
test_sents_iob = [ [(t, c if c[-2:] == 'NP' else 'O')
                    for w, t, c in snt]
                    for snt in conll2000.iob_sents('test.txt')]
print defTagger.evaluate(test_sents_iob)#0.43436688688604175
print uniTagger.evaluate(test_sents_iob)#0.8321126284906178
print biTagger.evaluate(test_sents_iob)#0.9341663676467484
print hmm_tagger.evaluate(test_sents_iob)#0.9360449163096017
```

Named entity recognition

- task: detect mentions of things with proper names (mostly)
  - people
  - locations
  - organizations
  - geo-political (country, kanton,...)
  - facility (bridge, building, airport,...)
  - vehicle

Specific NEs

- BioMed: biological entities (genes/proteins/...)
- Text+Berg: mountains/glaciers/huts/people
- Special kinds of NEs
  - temporal expressions
  - numerical expressions
  - → typically handled **separately**

Automatic NE recognition

Essentially:

- find phrases that are named entities
- classify into people/locations/organizations/...

In 1854 [$_{\text{PERS}}$Matthew Perry], a [$_{\text{GPE}}$U.S.] [$_{\text{ORG}}$Navy] commodore performed a blockade of [$_{\text{GPE}}$Japan] in the [$_{\text{LOC}}$Yokohama bay].

Relation Extraction

- extract relations between (typically) named entities
- specified as NE_1 x NE_2, where
  - x is a connecting word, such as “in”:

```
>>> IN = re.compile(r'.*\bin\b(?!\b.+ing)')
>>> for doc in nltk.corpus.ieer.parsed_docs('NYT_19980315'):
...     for rel in nltk.sem.extract_rels('ORG', 'LOC', doc,
...                                      corpus='ieer', pattern = IN):
...         print nltk.sem.show_raw_rtuple(rel)
[ORG: 'WHYY'] 'in' [LOC: 'Philadelphia']
[ORG: 'McGlashan &AMP; Sarrail'] 'firm in' [LOC: 'San Mateo']
[ORG: 'Freedom Forum'] 'in' [LOC: 'Arlington']
[ORG: 'Brookings Institution'] ', the research group in' [LOC: 'Washington']
```

Relation Extraction

- additional info: PoS-tags for fillers

```
>>> from nltk.corpus import conll2002
>>> vnv = """
... (
... is/V|    # 3rd sing present and
... was/V|   # past forms of the verb zijn ('be')
... werd/V|  # and also present
... wordt/V  # past of worden ('become')
... )
... .*       # followed by anything
... van/Prep # followed by van ('of')
... """
>>> VAN = re.compile(vnv, re.VERBOSE)
>>> for doc in conll2002.chunked_sents('ned.train'):
...     for r in nltk.sem.extract_rels('PER', 'ORG', doc,
...                                    corpus='conll2002', pattern=VAN):
...         print  nltk.sem.show_clause(r, relsym="VAN")
VAN("cornet_d'elzius", 'buitenlandse_handel')
VAN('johan_rottiers', 'kardinaal_van_roey_instituut')
VAN('annie_lennox', 'eurythmics')
```

Exam

- 24h
- Practical tasks, similar to labs
  - most likely one simple task in the beginning
  - most likely one on machine learning
  - most likely one on creating a pipeline using NLTK’s linguistic pre-processing (tokenization, sentence splitting, pos-tagging, …)
- Analyze output and/or report results

To conclude PCL2

- some programming concepts
  - classes, packages
  - file I/O, XML
  - external processes
- some organizational things
  - code style and comments
  - code sharing
  - testing, debugging
- algorithms and approaches
  - probabilities
  - machine learning, classification
  - dynamic programming

To conclude PCL2

- NLP tasks
  - handling files and corpora
  - n-grams, language modelling
  - classification (documents, words)
  - tagging
  - longest common subsequence, edit/Levenshtein distance
  - parallel sentence alignment
  - syntax and parsing, CFG, PCFG
  - semantics: logic, WSD
  - information extraction

To conclude PCL2

In other words:

- what you can do
  - NLP
  - programming

but most importantly:

- how stuff works
  - to know what you're dealing with
  - more importantly:

    to understand algorithms and learn to design and create your own
