CS 4 this week

Building classes...

hw10pr2

Connect Four Board class

... vs. using the library

hw10pr3

files and the dictionary class

files
dictionaries

If I had a dictionary, I guess I could look up what it was!
CS 4 this week

Building classes...

hw10pr2

Connect Four Board class

... vs. using the library

hw10pr3

files and the dictionary class

If an Algorithm Wrote This, How Would You Even Know?

By SHELLEY PODOLNY  MARCH 7, 2015
Algorithmic Authorship... ?

suppose this text represents my "style" ...

How could a program author new prose in this same style?!
Algorithmic Authorship...!

I like poptarts and 42 and spam. Will I get spam and poptarts for the holidays? I like spam poptarts!

Suppose this text represents my "style"...

What would be a reasonable first word to start a newly-generated sentence?

What would be a reasonable next word to follow the first?

What would be a reasonable test for sentence-ending?
'Cause somethin' like he left knee and a harp," said he had to the whole school? The shouting and then some strange and Mrs. "Well, I know Hagrid; they spotted handkerchief and get him get rid of course, had a gigantic beet with her," he knew what to all he's Wanna live while we're cool, so tonight What a feeling to be doing what I wish I know we only met but it ain't hard to be nothing left The story of my life I'm watching her eyes smile you flip your eyes You don't know what makes you got stars, they're in the wire She said, "Can I got a feeling to be a dentist

Who's the original human author of each of these? Hint: they're all British... Brit Lit's it!

This is but ourselves. No, faith, My uncle! O royal bed of confession Of your rue for leave to nature; to this time I should weep for thy life is rotten before he is. have sworn 't. Or my blood. I have closely sent for nine; and unprofitable,

The Senators and the date of a written declaration that Purpose, they shall consist of nine States, shall not, when he shall have such Vacancies. The President pro tempore, in the Desire of a Qualification to the Speaker of the Senate. Article 6. When vacancies by the office upon probable
Markov Models

Techniques for modeling any sequence of natural data

1st-order Markov Model (defining property)

Each item depends only on the one immediately before it.
Lists are *sequential* containers:

\[
L = [47, 5, 47, 42]
\]

elements are looked up by their *location*, or *index*, starting from 0

Dictionaries are *arbitrary* containers:

\[
d = \{47: 2, 42: 1\}
\]

elements (or *values*) are looked up by a *key* starting anywhere you want! *Keys* don't have to be ints!

We need a new data structure!

(A new *class*...)

elements are looked up by a *key* starting anywhere you want!
Lists are *sequential* containers:

\[ L = [ 47, 5, 47, 42 ] \]

elements are looked up by their *location*, or *index*, starting from 0

Dictionaries are *arbitrary* containers:

\[ d = \{ 47: 2, 42: 1 \} \]

elements (or *values*) are looked up by a *key* starting anywhere you want! *Keys* don't have to be ints!
Dictionaries are *arbitrary* containers:

\[
zd = \{ \text{'rabbit': 1999, 'ox': 1997} \}
\]

Elements (or *values*) are looked up by a *key* starting anywhere you want! Keys don't have to be ints!

What's *zd's* data here?

Now I see the *key* to dictionaries' *value*...
Dictionaries are *arbitrary* containers:

\[ zd = \{ \text{'rabbit': 1999, 'ox': 1997} \} \]

Elements (or *values*) are looked up by a *key* starting anywhere you want! *Keys* don't have to be ints!

Now I see the *key* to dictionaries' *value*...

<table>
<thead>
<tr>
<th>Animal</th>
<th>Dates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rat</td>
<td>Feb 19 1996–Feb 06 1997</td>
</tr>
<tr>
<td>Ox</td>
<td>Feb 07 1997–Jan 27 1998</td>
</tr>
<tr>
<td>Tiger</td>
<td>Jan 28 1998–Feb 15 1999</td>
</tr>
<tr>
<td>Rabbit</td>
<td>Feb 16 1999–Feb 04 2000</td>
</tr>
<tr>
<td>Dragon</td>
<td>Feb 05 2000–Jan 23 2001</td>
</tr>
<tr>
<td>Snake</td>
<td>Jan 24 2001–Feb 11 2002</td>
</tr>
<tr>
<td>Horse</td>
<td>Feb 12 2002–Jan 31 2003</td>
</tr>
<tr>
<td>Goat</td>
<td>Feb 01 2003–Jan 21 2004</td>
</tr>
<tr>
<td>Monkey</td>
<td>Jan 22 2004–Feb 08 2005</td>
</tr>
</tbody>
</table>

Now \[ zd[\text{'ox'}] \]

\[ zd[\text{'ox'}] \rightarrow 1997 \]

\[ zd[\text{'ox'}] \cdot 2 = 2009 \]

12-year zodiac...
Dictionaries are *arbitrary* containers:

\[ z = \{ 'rabbit': [1999, 1987, 1975, \ldots], \\
'ox': [1997, 1985, 1973, \ldots], \\
'tiger': [1998, 2010, \ldots], \ldots \} \]

\[ z[rabbit][1] \]

*What type are the keys?*  
*Strings*

*What type are the values?*  
*Lists of ints*
Dictionaries are *arbitrary* containers:

\[
z = \{ \text{'rabbit': [1999, 1987, 1975, ...]}, \\
    \text{'ox': [1997, 1985, 1973, ...]}, \\
    \text{'dragon': [2000, 1988, 1976, ...]}, \\
    ... \}
\]

Is 'dragon' a key in \(z\)?

Is 1969 in \(z[\text{'dragon'}]\)?
Dictionaries are in!

\[ z = \{ \text{'rabbit'}: [1999,1987,1975,\ldots], \]

'ox': [1997,1985,1973,\ldots],

'tiger': [1998,2010,\ldots], \ldots \} \]

Is 'dragon' a key in \( z \)?

\[ \text{if 'dragon' in } z \]

Is 1969 in \( z[\text{'dragon'}] \)?

\[ \text{if 1969 in } z[\text{'dragon'}] \]
\[
\text{LoW} = [ \text{'spam'}, \text{'spam'}, \text{'poptarts'}, \text{'spam'} ]
\]

\[
d = \{}
\]

\[
\text{for w in LoW:}
\]

\[
\text{if w not in d:}
\]

\[
d[w] = 1
\]

\[
\text{else:}
\]

\[
d[w] += 1
\]

\[
d\text{ will be...}
\]

\[
\{ \}
\]

\[
\text{2 'spam': 13}
\]

\[
\text{2 'spam': 23}
\]

\[
\text{2 'spam': 2, 'poptarts': 13}
\]

\[
\{ \text{'poptarts':1, 'spam':3} \}
\]

\[
\text{vc_print(LoW)}
\]

\[
\text{vc_print("a.txt")}
\]
LoW = ['spam', 'spam', 'poptarts', 'spam']

d = {}
for w in LoW:
    if w not in d:
        d[w] = 1
    else:
        d[w] += 1

Oldenborg's menu!

vc_print(LoW)
vc_print("a.txt")
LoW = ['spam', 'spam', 'poptarts', 'spam']

d = {}

for w in LoW:
    if w not in d:
        d[w] = 1
    else:
        d[w] += 1

Oldenborg's menu!
d will be...
{}
{}
{}
{}

but where to get so many words?

FILEs!

final d
Files...

```python
f = open('a.txt')

# opens the file and calls it f

text = f.read()

# reads the whole file into the string text

f.close()

# closes the file (optional)

text = 'I like poptarts and 42 and spam.\nWill I get spam and poptarts for the holidays? I like spam poptarts!'

# In Python reading files is smooth...

LoW = text.split()

# returns a list of each "word"

[ 'I', 'like', 'poptarts', ... ]
```
def word_count(filename):

    f = open(filename)
    text = f.read()
    f.close()

    LoW = text.split()
    print("There are", len(LoW), "words")

What if we wanted the number of different words in the file?

This would be the author's vocabulary count, instead of the total word count.
Vocabulary, anyone?

Shakespeare used **31,534 different words** -- and a grand total of 884,647 words, counting repetitions across all of his works....

http://www.math.cudenver.edu/~wbriggs/qr/shakespeare.html

Shakespearean coinages

- gust
- besmirch
- unreal
- superscript
- watchdog
- swagger

- affined
- rooky
- attaskaed
- out-villained

Successful

Unsuccessful

There's also **one contemporary** British author in the Oxford English Dictionary...

Who? with what word?

http://www.pathguy.com/shakeswo.htm
http://www.shakespeare-online.com/biography/wordsinvented.html
Vocabulary, anyone?

Shakespeare used **31,534 different words** -- and a grand total of 884,647 words, counting repetitions across all of his works....

http://www.math.cudenver.edu/~wbriggs/qr/shakespeare.html

Shakespearean coinages

- gust
- besmirch
- unreal
- superscript
- watchdog
- swagger
- successful
- unsuccessful
- affined
- rooky
- attasked
- out-villained
- muggle

'**Muggle**' goes into Oxford English Dictionary

JK Rowling's word for non-wizards - "muggle" - has made it into the new edition of the Oxford English Dictionary (OED).

The draft definition according to the dictionary's website says:

- **Muggle**: invented by JK (Joanne Kathleen) Rowling (b. 1965), British author of children's fantasy fiction (see quot. 1997).

  In the fiction of JK Rowling: a person who possesses no magical powers. Hence in allusive and extended uses: a person who lacks a particular skill or skills, or who is regarded as inferior in some way.

http://www.pathguy.com/shakeswo.htm
http://www.shakespeare-online.com/biography/wordsinvented.html

J. K. Rowling
from filename import defaultdict

def vocab_count( filename ):
    f = open( filename )
    text = f.read()
    f.close()

    LoW = text.split()
    print "There are", len(LoW), "words."

    d = {}

    for w in LoW:
        if w not in d:
            d[w] = 1
        else:
            d[w] += 1

    print "There are", len(d), "_distinct_ words."

    return d  # return d for later use by other code...

Tracking the number of occurrences of each word with a dictionary, d.

Same as before...
Markov Models can be *generative*!

A key benefit of Markov Models is that they can generate feasible data!

*Original file:*

I like poptarts and 42 and spam. Will I get spam and poptarts for the holidays? I like spam poptarts!

d = create_model('hpwhich.txt')
d = create_model('randj.txt')
d = create_model('oneD.txt')
d = create_model('a.txt')
gt(d, 250)
Markov Models can be *generative*!

A key benefit of Markov Models is that they can *generate* feasible data!

*Original file:*

I like poptarts and 42 and spam. Will I get spam and poptarts for the holidays? I like spam poptarts!

*Generated text:*

I get spam poptarts! I like poptarts and 42 and spam. I like spam and 42 and 42 and 42 and 42 and spam. Will I like poptarts and 42 and poptarts and 42 and poptarts and 42 and 42 and poptarts and spam. I get spam and 42 and 42 and...
Our Markov Model

Try it!

Markov Model
A dictionary!

What are the keys?
What are the values?
What are the missing values?
What is the '$_$'?
Why do some keys seem missing?

Original file

I like poptarts and 42 and spam. Will I get spam and poptarts for the holidays? I like spam poptarts!

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'$'</td>
<td>['I', 'Will', 'I']</td>
</tr>
<tr>
<td>'I'</td>
<td>['like', 'get', 'like']</td>
</tr>
<tr>
<td>'like'</td>
<td>['poptarts', 'spam']</td>
</tr>
<tr>
<td>'poptarts'</td>
<td>['and', 'for']</td>
</tr>
<tr>
<td>'and'</td>
<td>['42', 'spam.', 'poptarts']</td>
</tr>
<tr>
<td>'42'</td>
<td>['and']</td>
</tr>
<tr>
<td>'Will'</td>
<td>['I']</td>
</tr>
<tr>
<td>'the'</td>
<td></td>
</tr>
<tr>
<td>'spam'</td>
<td>['and', 'poptarts!']</td>
</tr>
<tr>
<td>'get'</td>
<td>['spam']</td>
</tr>
<tr>
<td>'for'</td>
<td>['the']</td>
</tr>
</tbody>
</table>

dictionary's end
Our Markov Model

Try it!

Markov Model
A dictionary!

What are the keys?
What are the values?
What are the missing values?
What is the '$'?
Why do some keys seem missing?

Original file

```
I like poptarts and 42 and spam. Will I get spam and poptarts for the holidays? I like spam poptarts!
```

keys
values

```
{
'\$': ['I', 'Will', 'I'],
'I': ['like', 'get', 'like'],
'like': ['poptarts', 'spam'],
'poptarts': ['and', 'for'],
'and': ['42', 'spam.', 'poptarts'],
'42': ['and'],
'Will': ['I'],
'the': ['holidays?'],
'spam': ['and', 'poptarts!'],
'get': ['spam'],
'for': ['the']
}
```
Markov-modeling's *algorithm*

```
LoW  ['I','like','spam.','I','eat','poptarts!']

pw

nw

d = {}
pw = '$'

for nw in LoW:
    if pw not in d:
        d[pw] = [nw]
    else:
        d[pw] += [nw]

pw = ＿＿＿＿
```

```
d's final form
(without quotes)

$ : [ l, l ]
I : [ like, eat ]
like : [ spam. ]
eat : [ poptarts! ]
```
Model creation:

1) start with the previous word, \texttt{pw} as '$$
2) for each next word, \texttt{nw}, in the list of words, add it in ...
3) then change \texttt{pw} to \texttt{nw} ...
   
   (a) except if \texttt{nw[-1]} was punctuation: change \texttt{pw} to...

Generating text:

1) start with \texttt{pw} as the '$$' string
2) choose a \texttt{nw} that follows \texttt{pw}, at random.
3) print \texttt{nw}, \hspace{1cm} (the comma continues on the same line)
4) \texttt{pw} gets set to either \texttt{nw} or '$$' \hspace{1cm} \textcolor{red}{\leftarrow}
   
or if \texttt{nw[-1]} was punctuation: change \texttt{pw} to...
Dear Zachary,

We invite you to submit a paper/abstract to The 15th World Multi-Conference on Systemics, Cybernetics and Informatics: WMSCI 2011, to be held in Orlando, Florida, USA, on July 19th - July 22nd, 2011 (www.2011iiisconferences.org/wmsci).

If you have any colleagues who might be interested in making a submission to the conference, please feel free to forward this e-mail to them.

Below are the next deadlines for WMSCI 2011 (Check the web site for possible extensions or new set of deadlines):

Papers/Abstracts Submission and Invited Session Proposals: November 25th, 2010
Authors Notifications: January 31st, 2011
Camera-ready, full papers: February 28th, 2011
Dear Zachary Dodds,

We would like to inform you that we extended to *April 5, 2017* the submission deadline for your potential contribution in the area "Robotics" or any other included in the 21st World Multi-Conference on Systemics, Cybernetics and Informatics: WMSCI 2017 (http://www.2017iiisconf.org/wmsci), to be held on July 8 - 11, 2017, in Orlando, Florida, USA, jointly with:

- The 11th International Multi-Conference on Society, Cybernetics, and Informatics: IMSCI 2017
- The 15th International Conference on Education and Information Systems, Technologies and Applications: EISTA 2017
- The 10th International Multi-Conference on Engineering and Technological Innovation: IMETI 2017

The respective websites of the above events and the others being jointly organized can be found at the general CFP posted at: http://www.2017iiisconf.org/cfp-summer2017.asp

To submit your article, please click the "Authors" tab on the conference website. Submissions for face-to-face and virtual presentations are both accepted.

WMSCI and all its collocated events are being indexed by Elsevier's SCOPUS since 2005. The 2017 proceedings will also be sent to Elsevier’s SCOPUS.
Rooter: A Methodology for the Typical Unification of Access Points and Redundancy

Jeremy Stribling, Daniel Aguayo and Maxwell Krohn

http://pdos.csail.mit.edu/scigen/

Markov-generated submission accepted to WMSCI 2005

Not a first-order model ... but a third-order model
Not a first-order model ... but a third-order model
There are no one-sided coins...

iOS Just Got A Paper On Nuclear Physics Accepted At A Scientific Conference

Posted by Christoph Bartneck on Oct 20, 2016 in Featured, Research | 7 comments

Automatically generating scientific articles has become easy with dedicated software such as SCiGen. Even a paper that only repeated the sentence “Get me of your mailing list” was recently accepted for publication. Today I received an invitation from the International Conference on Atomic and Nuclear Physics to submit a paper. Since I have practically no knowledge of Nuclear Physics I resorted to iOS auto-complete function to help me writing the paper. I started a sentence with “Atomic” or “Nuclear” and then randomly hit the auto-complete suggestions. The text really does not make any sense. After adding the first illustration on nuclear physics from Wikipedia, some references and creating a fake identity (Iris Pear, aka Siri Apple) I submitted the paper which was accepted only three hours later! I know that iOS is a pretty good software, but reaching tenure has never been this close.

UPDATE (27/10/2016): Turns out that conference organizer, OMICS Group, is currently under federal investigation.

http://www.bartneck.de/2016/10/20/ios-just-got-a-paper-on-nuclear-physics-accepted-at-a-scientific-conference/
Project deadlines?
Papers due?

Have a **worry-free** weekend!

Have Python write your papers for you...

... *you're still the author!*
Last weeks in CS 5

Mon., 4/22 – hw11 + project start due
Mon., 4/29 – hw12 + milestone due
Fri., 5/3 – 8pm: final proj. due

**final project**

Last class: 5/2
Thurs., 5/9 class time – review
Tues., 5/14 2 or 7 pm – final
Final-project preview!

Choices of final project:

- TextID
- Picobot
- vPython
- Connect 4 AI

Today ~ overview

next time ~ overview
Final 2 lab days...

Labs **will** meet the last 2 weeks of class, *but*...

- they're *extra-optional* ~ no lab prob.  
  - **no sign in**
- labtime: *final projects* ~ assistance + progress
- there are *theocomp* problems, too  
  - (hw12)
- Plus, hw12 has up to +50 pts of extra credit!

Now we're in a state!
'tis the season for final projects...

Today is about the CS 5 final projects + it's a sales pitch for the four possible options:

Picobot
C4 AI
TextID
vPool

I've got my eyes on some of these projects!

Eye'll bet!
Final projects

Final CS hw

- open-ended
- comprehensive
- same projects across sections
- several choices...

Working in teams of 1-3 is OK

Teams need to work together - *in the same place* - and need to share the work equally...

Pairs/trios are welcome (larger should split)

*Teaming is extra-encouraged* on the final project!
**4/22**
- "Start" ~ part of hw11

**Mon. 4/29**
- "Milestone" ~ part of hw12
- project-specific tasks to help w/progress...

**Fri. 5/3**
- Final project & short reflection on how to run it and how it went.
- due at 8pm
- Euros ok; grutoring tapers.
Final projects

Overview: Project options

- vPool: 3D graphics
- Connect Four: Game AI
- TextID: Text-style matching
- Picobot Project: Genetic algorithms

Start, milestone, and final-project submission

Dates:

- The final version is due Friday, May 3rd at 8pm
- The milestone version is due Monday, April 29th by 11:59pm.
- The starter version is due Monday, April 22nd by 11:59pm.
The projects...

- VPool
- TextID
- Picobot!
- C4 AI
VPool

Not 3... but 4!

Next week's lab!
A few constraints...

- need ≥4 physically interacting objects
- allow the user to direct 1+ objects, either by keyboard or mouse or both
- needs a game goal + be winnable!
- must detect some "linear" and some "spherical" collisions and implement their results on the motion of the objects
Do you see the pool table!? the pool balls? the alien?
The projects...
The Picobot project

Big idea

(1) Implement Picobot in Python
(2) *Train Python to write successful Picobot programs!*

talk about going *full circle*...
Picobot, behind the curtain...

What data structures (classes) might be helpful in implementing Picobot?
Picobot's classes

**class** Program:

What in Python could most usefully hold all of these *rules*?

What *type* should *self.rules* be?
Picobot's classes

class Program:

What in Python could most usefully hold all of these rules?

What type should self.rules be?

Python dictionary

both tuples

self.rules[ (1,"NExx") ] = ("W",0)
Picobot's classes

What type in Python could most usefully hold the environment?

```python
class World:
    # What class that you've already written will be most similar to Picobot's World?
    # What will self.room be?
```

Wall: +
Visited: O
Picobot: P
Picobot's classes

What type in Python could most usefully hold the *environment*?

```python
class World:
    class Board:
        What class that you've already written will be most similar to Picobot's `World`?
        What will `self.room` be?
        The same as the Connect-Four board's `self.data`!
```

Wall: +
Visited: O
Picobot: P
Picobot's project

Current State: 1
Current Rule: 1 N*W* \rightarrow X 2

First, build an ASCII simulation

demo!

Your actual ASCII is likely to be more monochromatic!

http://rednuht.org/genetic_cars_2/ or http://boxcar2d.com/
Box2d: https://www.youtube.com/watch?v=uxourrIPIf8
Picobot's project

Current State: 1
Current Rule: 1 N*W* -> X 2

Picobot started here...

and is now here...

Your actual ASCII is likely to be more monochromatic!

First, build an ASCII simulation

demo!

http://rednuht.org/genetic_cars_2/ or http://boxcar2d.com/  
Box2d: https://www.youtube.com/watch?v=uxourlPlf8
Picobot's project

Current State: 1
Current Rule: 1 N*W* -> X 2

Picobot started here...

and is now here...

Your actual ASCII is likely to be more monochromatic!

First, build an ASCII simulation
then, evolve it...
with your own genetic algorithm

http://rednuht.org/genetic_cars_2/ or http://boxcar2d.com/
Box2d: https://www.youtube.com/watch?v=uxourrIPif8
Program evolution

An example of genetic algorithms, which are used for optimizing hard-to-describe functions with easily-splittable solutions.

Start with a population of, say, ~200 random Picobot programs...
Program evolution

An example of *genetic algorithms*, which are used for optimizing hard-to-describe functions with easily-splittable solutions.

Then, *measure* the fitness of all of those programs.

Measure? How??
program p1
fitness = 0.03

mate + mutate the fittest rulesets

to create a *new generation* of ~200 programs...

program p2
fitness = 0.05

program c1
fitness = 0.19

What the goal?
Repeat this "survival of the fittest" process for many generations...

... and by the end, your Python code should/will have evolved a much more capable Picobot program!
Genetic Algorithms ~ the 3rd way?!

the current "fitness" leader

the current "generation"

fitnesses!

Rafael Matsunaga: rednuht.org/
http://rednuht.org/genetic_cars_2/
The projects...

VPool

TextID

C4 AI

Picobot!
What CSers "do" think they or think as much...

final project *algorithms*...
A couple of years ago...

Though Robin Ellacott’s twenty-five years of life had seen their moments of drama and incident, she had never before woken up in the certain knowledge that she would remember the coming day for as long as she lived.
Though Robin Ellacott's twenty-five years of life had seen their moments of drama and incident, she had never before woken up in the certain knowledge that she would remember the coming day for as long as she lived.
How Robert Galbraith was found to be JK Rowling

SARAH PAVIS • AUG 08 2013

I was given e-text copies of Cuckoo to compare against Rowling’s own The Casual Vacancy, Ruth Rendell’s The St. Zita Society, P.D. James’ The Private Patient and Val McDermid’s The Wire in the Blood. [...] 

I actually ran four separate types of analyses focusing on four different linguistic variables. While anything can in theory be an informative variable, my work focuses on variables that are easy to compute and that generate a lot of data from a given passage of language. One variable that I used, for example, is the distribution of word lengths. Each novel has a lot of words, each word has a length, and so one can get a robust vector of % of the words in this document have exactly letters. Using a distance formula (for the mathematically minded, I used the normalized cosine distance formula instead of the more traditional Euclidean distance you remember from high school), I was able to get a measurement of similarity, with 0.0 being identity and progressively higher numbers being greater dissimilarity.
I like poptarts and 42 and spam. Spamful poptarts are like poptartful spams -- and are liked by all! Will _Thanksgiving_ bring spam poptarts?

File

TextModel

... will have at least five Python dictionaries, e.g.,

```
{'and': 3, 'poptartful': 1, 'liked': 1, 'spamful': 1, 'like': 2, ':': 1, 'spam': 2, 'i': 1, '42': 1, 'all': 1, 'thanksgiving': 1, 'will': 1, 'bring': 1, 'poptarts': 3, 'spams': 1, 'by': 1, 'are': 2}
```

```
{0: 1, 1: 1, 2: 2, 3: 6, 4: 5, 5: 3, 7: 1, 8: 3, 10: 1, 12: 1}
```

```
{'and': 3, ':': 1, 'all': 1, 'like': 3, 'thanksgiving': 1, 'will': 1, 'bring': 1, 'poptart': 4}
```

```
{12: 1, 5: 1, 7: 1}
```

```
{':': 1, 'i': 2, '?': 1, '_': 2, '!': 1}
```

What are these four other dictionaries counting?!
I like poptarts and 42 and spam. Spamful poptarts are like poptartful spams -- and are liked by all!
Will _Thanksgiving_ bring spam poptarts?

I like poptarts and 42 and spam.
Spamful poptarts are like poptartful spams -- and are liked by all!
Will _Thanksgiving_ bring spam poptarts?

... will have at least five Python dictionaries, e.g.,

```python
{'and': 3, 'poptartful': 1, 'liked': 1, 'spamfull': 1, 'like': 2, ':': 1, 'spam': 2, 'i': 1, '42': 1, 'all': 1, 'thanksgiving': 1, 'will': 1, 'bring': 1, 'poptarts': 3, 'spams': 1, 'by': 1, 'are': 2}
```

```python
{0: 1, 1: 1, 2: 2, 3: 6, 4: 5, 5: 3, 7: 1, 8: 3, 10: 1, 12: 1}
```

```python
{'and': 3, ':': 1, 'all': 1, 'like': 3, 'thanksgiv': 1, 'spam': 4, 'i': 1, '42': 1, 'by': 1, 'will': 1, 'bring': 1, 'poptart': 4}
```

```python
{12: 1, 5: 1, 7: 1}
```

```python
{':': 1, '-': 2, '?': 1, '_': 2, '.': 1}
```

What are these four other dictionaries counting?!
Suppose these two Python dictionaries count words from various texts, e.g., by J.K. Rowling and W. Shakespeare. Which of these two text models does the third dictionary, at right – with unknown author -- better match? *Why?*

**JKR**
```
{ "love": 25,
  "spell": 275,
  "potter": 700 }
```

**WS**
```
{ "love": 50,
  "spell": 8,
  "thou": 42 }
```

**word-count dictionary for an unknown author**
```
{ "love": 3,
  "thou": 1,
  "potter": 2,
  "spam": 4 }
```

*Extra: what algorithm would you devise to quantify the two matches here?!!*
Naïve Bayes classification

Bayesian spam filtering

Bayesian spam filtering (pronounced /ˈbeɪzjən/; after Rev. Thomas Bayes) is a statistical technique of e-mail filtering. In its basic form, it makes use of a naïve Bayes classifier on bag of words features to identify spam e-mail, an approach commonly used in text classification.

Constructing a classifier from the probability model

The discussion so far has derived the independent feature model, the assumption model. The naïve Bayes classifier combines this model with a decision rule to pick the hypothesis that is most probable; this is known as the Bayes decision rule. The corresponding classifier, a Bayes classifier, is the function defined as follows:

\[
\text{classify}(f_1, \ldots, f_n) = \arg\max_c p(C = c) \prod_{i=1}^{n} p(F_i = f_i | C = c).
\]

Don't take these formulas too seriously...
Model matching

WS: { "love": 50, 
    "spell": 8, 
    "thou": 42 } 

JKR: { "love": 25, 
      "spell": 275, 
      "potter": 700 } 

Unknown text: { "love": 3, 
                 "thou": 1, 
                 "potter": 2, 
                 "spam": 4 } 

Suppose we have two trained models:

These must have been some really avant-garde texts!
Model matching

Suppose we have two trained models:

WS: {
  "love": 50,
  "spell": 8,
  "thou": 42
}

JKR: {
  "love": 25,
  "spell": 275,
  "potter": 700
}

normalize for size

WS: {
  "love": 0.50,
  "spell": 0.08,
  "thou": 0.42
}

JKR: {
  "love": 0.025,
  "spell": 0.275,
  "potter": 0.700
}

Unknown text: {
  "love": 3,
  "thou": 1,
  "potter": 2,
  "spam": 4
}
**Model matching**

WS: { "love": 0.50,  
"spell": 0.08,  
"thou": 0.42 }  

JKR: { "love": 0.025,  
"spell": 0.275,  
"potter": 0.700 }  

**how do we compare the models with an unknown text?**

*Unknown text*: { "love": 3,  
"thou": 1,  
"potter": 2,  
"spam": 4 }

Pretend the words are all independent

Suppose we have two *normalized models*:

What's the **likelihood** of the new model arising from each?  

There's probably a way to do this!
Model matching

Suppose we have two normalized models:

WS: { "love": 0.50,  
     "spell": 0.08,  
     "thou": 0.42 }  

JKR: { "love": 0.025,  
      "spell": 0.275,  
      "potter": 0.700 }

The WS-based probability of each word in Unknown text:

love love love thou potter potter spam spam spam spam

Unknown text: { "love": 3,  
                 "thou": 1,  
                 "potter": 2,  
                 "spam": 4 }  

I've got near-zero ideas on this one!
Model matching

Suppose we have two normalized models:

WS: { "love": 0.50, "spell": 0.08, "thou": 0.42 }

JKR: { "love": 0.025, "spell": 0.275, "potter": 0.700 }

the WS-based probability of each word in Unknown text:

Unknown text: { "love": 3, "thou": 1, "potter": 2, "spam": 4 }

The WS-based probability of each word in Unknown text:

- love: 0.50
- love: 0.50
- love: 0.50
- thou: 0.42
- potter: 0
- potter: 0
- spam: 0
- spam: 0
- spam: 0
- spam: 0

Correct, but not helpful!

I've got near-zero ideas on this one!
Model matching

Suppose we have two normalized models:

WS: { "love": 0.50, "spell": 0.08, "thou": 0.42 }

JKR: { "love": 0.025, "spell": 0.275, "potter": 0.700 }

the WS-based probability of each word in Unknown text

Unknown text: { "love": 3, "thou": 1, "potter": 2, "spam": 4 }

Why is this especially incorrect?

WS-based probability: .50 • .50 • .50 • .42 • 1 • 1 • 1 • 1 • 1 • 1 • 1

= 0.0525
Model matching

Suppose we have two normalized models:

WS: { "love": 0.50, "spell": 0.08, "thou": 0.42 }

JKR: { "love": 0.025, "spell": 0.275, "potter": 0.700 }

the \textbf{WS}-based probability of each word in \textit{Unknown text}

for missing words, use \textbf{half} the smallest value – across both normalized models!

\[
\begin{array}{cccccccc}
\text{love} & \text{love} & \text{love} & \text{thou} & \text{potter} & \text{potter} & \text{spam} & \text{spam} \\
3 & 1 & 2 & 4
\end{array}
\]

\[
\text{Unknown text:} \quad \{ \text{"love": 3, } \text{"potter": 2, } \text{"thou": 1, } \text{"spam": 4 } \}
\]

\[
\frac{.50 \cdot .50 \cdot .50 \cdot .42 \cdot .012 \cdot .012 \cdot .012 \cdot .012 \cdot .012 \cdot .012}{2} = 1.57 \times 10^{-13}
\]

What?
Model matching

Suppose we have two normalized models:

WS: { "love": 0.50, "spell": 0.08, "thou": 0.42 }

JKR: { "love": 0.025, "spell": 0.275, "potter": 0.700 }

the WS-based probability of each word in Unknown text

for missing words, use half the smallest value – across both normalized models!

Unkonwn text: { "love": 3, "thou": 1, "potter": 2, "spam": 4 }

take logs of everything!

too small!

= 1.57e-13

half of e!
Suppose we have two normalized models:

WS: { "love": 0.50, "spell": 0.08, "thou": 0.42 }

JKR: { "love": 0.025, "spell": 0.275, "potter": 0.700 }

the WS-based probability of each word in Unknown text

for missing words, use half the smallest value – across both normalized models!

Unknown text: { "love": 3, "potter": 2, "thou": 1, "spam": 4 }
Model matching

Suppose we have two normalized models:

WS: \{
  "love": 0.50,
  "spell": 0.08,
  "thou": 0.42
\}

JKR: \{
  "love": 0.025,
  "spell": 0.275,
  "potter": 0.700
\}

the WS-based probability of each word in Unknown text

\[
3 \times \log(0.025) + \log(0.012) + 2 \times \log(0.7) + 4 \times \log(0.012) = -33.89
\]

Unknown text: \{
  "love": 3,
  "potter": 2,
  "thou": 1,
  "spam": 4
\}

this looks close...
Model matching

Suppose we have two normalized models:

WS: { "love": 0.50,  
    "spell": 0.08,    
    "thou": 0.42 }

JKR: { "love": 0.025,  
    "spell": 0.275,    
    "potter": 0.700 }

Unknown text:
{ "love": 3,    
  "potter": 2,    
  "thou": 1,    
  "spam": 4 }

-29.48

-33.89

the (much) better match...
Naïve Bayes classification

Bayesian spam filtering

From Wikipedia, the free encyclopedia

Bayesian spam filtering (ˈbeɪzɪən/ bay-zee-ən; after Rev. Thomas Bayes) is a statistical technique of e-mail filtering. In its basic form, it makes use of a naïve Bayes classifier on bag of words features to identify spam e-mail, an approach commonly used in text classification.

Constructing a classifier from the probability model

The discussion so far has derived the independent feature model, that is, the naïve Bayes model. The naïve Bayes classifier combines this model with a decision rule. Given the probability of the hypothesis that is most probable; this is known as the maximum a posteriori (MAP) rule. The corresponding classifier, a Bayes classifier, is the function classifying inputs:

\[
\text{classify}(f_1, \ldots, f_n) = \arg\max_c \ p(C = c) \prod_{i=1}^{n} p(F_i = f_i|C = c).
\]

Don't take these formulas too seriously...
Naïve Bayes classification

Bayesian spam filtering

Bayesian spam filtering (ˈbeɪz iən/ bay-zee-en; after Rev. Thomas Bayes) is a statistical approach used for classification. It is based on Bayes' theorem and makes the assumption of independence between predictors. This assumption simplifies the calculation of posterior probabilities.

The formula for Bayes' theorem in spam filtering is:

$$
Pr(S|W) = \frac{Pr(W|S)}{Pr(W|S) + Pr(W|H)}
$$

This quantity is called "spamicity" (or "spaminess") of the word "replica", and can be computed. The numerator is the probability of receiving a spam email containing the word "replica", and the denominator is the probability of receiving a spam or ham email containing the word "replica".

The discussion so far has derived the independent feature model, the form of the naive Bayes model. The naive Bayes classifier combines this model with a decision rule. One commonly used decision rule is the maximum a posteriori (MAP) hypothesis, which is the hypothesis that is most probable. The corresponding classifier, a Bayes classifier, is the function of classification.

$$
\text{classify}(f_1, \ldots, f_n) = \arg\max_c p(C = c) \prod_{i=1}^{n} p(F_i = f_i|C = c).
$$

Don't take these formulas too seriously...
More projects next time!

An unusual variation on VPool