Final-week overview...

Lab today:

- TextGame
- vPython
- TextID

Choices

Whew!

Chat Q'n of the day:

What is your preferred temperature (outdoor)?

and... how close is your weather right now?!

Start!

Idea + breakdown
"Final" exam...
"Final" week...

[90%] Final project

5 choices...

[10%] Final hw problem

"Finite-state machines"

5 machines...
really, 1 hw problem
Final-projects ...

**Choices**

- TextGame
- vPython
- TextID
- Picobot
- Game of Life

Lab today: Decide on a project...

Hw12 Finite state machines...
Final-projects ...  Choices

Today's overview

- TextGame
- vPython
- TextID
- Picobot
- Game of Life

We'll return to these in the latter part of today...

Lab today: Decide on a project...

Hw12  Finite state machines...
"Theocomp"

What about *this movie*?
"Theocomp"
CS's broad view:

What's else?

Final Projects

CS Applications

- simulation + analysis/ algorithms
- graphics / media / games ...
- feature-based modeling + classification
- other state-machines + fun stuff

CS Foundations

is under here:

CS Theory

What can we compute... ... and how well?

CS Practice

Picobot
vPool
TextID
TextGame
Game of Life

CS 5

functions

variables

loops

recursion

data: classes and objects

circuits and memory

CS Applications

CS's broad view:
"Theocomp"

1. define "computer" precisely
2. define "compute" precisely
3. see what computers provably can+can't compute...
4. go back to step (1) and define things better
5. ... until time runs out...

Aargh!

CS Foundations

What can we compute... ... and how well?
"Theocomp"

things get Strange ...

What can be computed... and how efficiently?

What can we compute... and how well?

Things get Strange...
computers ~ state machines

What is this state machine?

State Machine

2,3 neighbours

3 neighbours

1, 4-8 neighbours

1, 2 4-8 neighbours
computers ~ state machines

What is this state machine?

Game-of-life State Machine
Unifying idea: **State**

The *state* of a computation is **all the internal information** needed to take the next step.

Picobot takes "next step" literally!
states ~ subtasks

State Machine:
each oval represents a different Picobot state

surroundings
state pattern -> move new state

0 x*** -> N 0
0 N*** -> X 1
1 ***x -> S 1
1 ***S -> X 0

starting funnel
the "go North" state

transitions move from state to state

the "go South" state
**Computation** is a deliberate sequence of state-changes.

Before:

```
10101001011
00000000000
```

After:

```
10101001011
00000000110
```
Computation is a deliberate sequence of state-changes.

Computer ~ a Finite State Machine

https://www.youtube.com/watch?v=FiEoxQp4DhQ
Finite state machine

- State 0
  - Transition on 0
  - Transition on 1

- State 1
  - Transition on 0
  - Transition on 1
Finite state machine

State 0

transition on 0

State 1

transition on 0

transition on 1

an input 001011

"input funnel"

start state(s)

"where to go"

double circled

transition on 0

accepting state(s)

transition on 1

transition on 1

Finite state machine
FSM: *Finite state machine*

What does each state MEAN?
What does this FSM do overall?
FSM: Finite state machine

State 0
- Transition on 0
- Transition on 1

State 1
- Transition on 0
- Transition on 1

Input sequence: 001011
Output for this input: accepted!

"I've seen an ODD # of 1's"

"I've seen an EVEN # of 1's"

What does each state MEAN?
What does this FSM do overall?
FSM: *Finite state machine*

State 0

Transition on 0

Transition on 1

Transition on 1

State 1

An input sequence always left-to-right

0010111

Output for this input

Rejected!

"I've seen an ODD # of 1's"

"I've seen an EVEN # of 1's"

What does each state MEAN?

What does this FSM do overall?
JSFLAP!  

our two-state machine...

empty string, ",", or \( \lambda \) 

graphical state-machine builder for hw12
JSFLAP! graphical state-machine builder for hw12

Keep "tabs" on your state machines – at least until you've submitted them – and they work!

Our two-state machine...

empty string, '', or λ

lambda
JSFLAP! graphical state-machine builder for hw12

Keep a tab for each problem in a browser window...

Keep "tabs" on your state machines – at least until you've submitted them – and they work!
In general, what English phrase describes the rejected inputs?

This machine rejects strings with ...

Extra: Could fewer states produce the same accept-and-reject behavior here? What's the minimum #?

Hint: which strings have to be in separate states?
In general, what English phrase describes the rejected inputs?

This machine rejects strings with ... two 1's in a row (anywhere in the string)

Extra Could fewer states produce the same accept-and-reject behavior here? What's the minimum #?

Hint: which strings have to be in separate states?

3 states min.

0 1 11
add 1
add 0

00 10 101 101010 11 0011 110 0011001

Label each state with 1-2 inputs that "land" there...
**Quiz**

In general, what English phrase describes the rejected inputs?

This machine rejects strings with ...

Extra Could fewer states produce the same accept-and-reject behavior here? What's the minimum #?

Hint: which strings have to be in separate states?

Name(s) ____________________________

<table>
<thead>
<tr>
<th>State</th>
<th>Input 0</th>
<th>Input 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0</td>
<td>0</td>
<td>0, 1</td>
</tr>
<tr>
<td>s1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>s2</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

ends in zero 1's
ends in one 1
contains two (or more) 1's in a row

"graveyard" state: uses two transitions in JFLAP

ends in zero 1's
ends in one 1
contains two (or more) 1's in a row

add 1
add 0
Extra Could fewer states produce the same accept-and-reject behavior here? What's the minimum #?

Hint: which strings have to be in separate states?

In general, what English phrase describes the rejected inputs?

This machine rejects strings with two 1's in a row (anywhere in the string)

Label each state with 1-2 inputs that "land" there...

"graveyard" state: uses two transitions in JFLAP

contains two (or more) 1's in a row
FSM ~ Game AI

The state-machine that controls Quake's Shambler monsters...

I'm Quaking in my AstroBoots
Software state machine

still broken

stop adding features + start adding print statements

it's broken

it works

still works

comment out print statements + start adding more features
Software state machine

Stop adding features + start adding print statements → still broken

still broken → comment out print statements + start adding more features → still works

still works → store a copy somewhere else!

it's broken → it works
Build-your-own FSMs

Draw a FSM accepting strings with at least two 1s (anywhere). Others are rejected.

Accepted examples: 0101, 00010110, 111011, 11
Rejected examples: 0100, 1000, 000000, 1, 0

Hint - modify this starter FSM by adding labels, transitions, and one more state:

1

→

Draw a FSM accepting strings in which the number of zeros (0s) is a multiple of 3, so there are 0, 3, 6, ... zeros. 1s don't matter!

Accepted: 110101110, 11, 0000010
Rejected: 101, 0000, 1101110111

Hint: 1s never change the state!

Another hint: make a triangle!

What's the minimum number of states needed?

Extra! Draw a FSM accepting strings whose third-to-last digit (3d from the right) is a 1.

Accepted: 0100 and 01101
Rejected: 10101 and 11

Big picture

CS ~ building state machines
Build-your-own FSMs

Draw a FSM accepting strings with at least two 1s (anywhere). Others are rejected.

Accepted examples: 0101, 00010110, 111011, 11
Rejected examples: 0100, 1000, 000000, 1, 0

Draw a FSM accepting strings that don’t contain the pattern 110 anywhere.

Accepted: 1010001, 011
Rejected: 101001100, 0110

If '110' not in s:
  return True
else:
  return False

if count('1',s) >= 2:
  return True
else:
  return False

if count('0',s)%3 == 0:
  return True
else:
  return False

if s[2] == '1':
  return True
else:
  return False

if s[-3] == '1':
  return True
else:
  return False

Extra! Draw a FSM accepting strings whose third-to-last digit (3d from the right) is a 1.

Accepted: 0100 and 01101
Rejected: 101001 and 11
**Build-your-own FSMs**

Draw a FSM accepting strings with at least two 1s (anywhere). Others are rejected.

**Accepted examples:** 0101, 00010110, 111011, 11

**Rejected examples:** 0100, 1000, 000000, 1, 0

**Hint** - modify this starter FSM by adding labels, transitions, and one more state:

Draw a FSM that accepts strings that *don't* contain the pattern 110 anywhere.

**Accepted:** 1010001, 011

**Rejected:** 10100100, 01101

**Hint** - there are FIVE more transitions – but no more states - needed here

Draw a FSM accepting strings in which the number of zeros (0s) is a multiple of 3, so there are 0, 3, 6, ... zeros. *1s don't matter!*

**Accepted:** 110101110, 11, 0000010

**Rejected:** 101, 0000, 11101110111

**Extra!** Draw a FSM accepting strings whose third-to-last digit (3d from the right) is a 1.

**Acc:** 0100 and 01101

**Rej:** 101001 and 11

Draw a FSM accepting strings in which the third digit (3d from the left) is a 1.

**Accepted:** 1010001, 011

**Rejected:** 11000100, 11, 0

**What's the minimum number of states needed?**

4

8
Has at least two 1s... ?

Draw a FSM accepting strings with at least two 1s (anywhere). Others are rejected.

**Accepted:** 0101, 00010110, 111011, 11

**Rejected:** 0100, 1000, 000000, 1, 0

What do we need to complete this machine?
Number of 0s is div. by 3

Draw a FSM accepting strings in which the number of zeros (0s) is a multiple of 3, so there are 0, 3, 6, ... zeros. 1s don't matter.

Accepted: 110101110, 11, 0000010
Rejected: 101, 0000, 11101110111

Minimum number of states?
No occurrences of 110?

Draw a FSM accepting strings that do NOT anywhere contain the pattern 110

Accepted: 1010001, 0001011

Rejected: 101001100, 011001

Which transitions are still needed here?
Third character is a 1

Draw a FSM accepting strings in which the third digit (from the left) is a 1.

Accepted: 1010001 and 0110
Rejected: 11000100 and 11

Why must $s_1$ and $s_2$ be separate states?

Minimum number of states?
Third character is a 1

Draw a FSM accepting strings in which the third digit (from the left) is a 1.

Accepted: 1010001 and 0110
Rejected: 11000100 and 11

Why must s1 and s2 be separate states?

Minimum number of states?

Why must s1 and s2 be separate states?

1 11
s1
11 111
s2

s0

0 1

s1

0 1

s2

1 0

s3

1 0

s4

3rd is "1"

3rd is "0"
Third-to-last character is a 1?

Draw a FSM accepting strings whose third-to-last digit (from the right) is a 1.

Accepted: 0100 and 01101
Rejected: 101001 and 11

Minimum number of states?
Third-to-last character is a 1

I don't accept this solution!
Something's not right here: it's downright harrowing!

Do we *need* 15 states?
Third-to-last character is a 1

8 states?

8 states are required!
All robots use FSM control

... send me your FSM so that I can show it off in 2020!
All robots use FSM control

What states can you "factor out" from watching this towel-folding?
Towel-folding states!

Fig. 2. The state machine model of the procedure: dashed lines indicate failure recovery cases. The images show an actual run.
Final-projects ...

Lab today: Decide on a project...

Hw12  Finite state machines...
Final-project overview...

Why?

(1) Story!

(2) Experience!

State 0  
Vision

The exploratory-software experience
Today

Decide on a project...

Thursday

• "Milestone" ~ working progress

Sunday

• "Final" ~ complete, with adjustments...

When?!
Overview

Taking most of the oxygen for the last two "weeks" (really: days!) will be the final project. The project is larger in scope than a regular assignment and so is worth 242 points across the two deliverables. We have four possible final project themes in CS5. All of them provide the chance to build a significant software application in Python—and all of them also provide opportunities for creative expansion of the basic themes...

Here are the project descriptions/links:

- Text-based game project
- TextID: Text-style matching
- Picobot Project: Genetic algorithms
- More Life! Variations on Conway's Game of Life
- vPython: 3D graphics

Project Breakdown

Dates:

- The final version is due **Friday by 11pm** The weekend is allowed as a "buffer," as well...
- The milestone version is due **Thursday by 11pm**.
- The starter version is not due for our summer session (we're concentrated already!)
  - That said, the starter tasks are a **great thing** to work on initially, e.g., Tue/Wed.
The projects...

VPool

Game!

TextID

Life

Picobot!
Project *space*...

- Picobot!
- Life
- TextID
- Text Game
- VPool

- Algorithmic
- Open-ended (and 3d!)
- Practical + checkable
vPool ~ vPython ~ GlowScript!

VPython
3D Programming for Ordinary Mortals

No Exit

can be very intricate!
VPython ~ GlowScript!

built by and for physicists to simplify 3d simulations

lots of available classes, objects and methods in its API

or very simple...

www.glowscript.org/
API ... stands for *Application Programming Interface*

<table>
<thead>
<tr>
<th>3D Objects</th>
<th>3D Objects</th>
<th>3D Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrow</td>
<td>box</td>
<td>clone</td>
</tr>
<tr>
<td>compound</td>
<td>cone</td>
<td>text</td>
</tr>
<tr>
<td>cylinder</td>
<td>extrusion</td>
<td>frame</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Shapes + Docs!**

**Constructors + Methods!**

**Cool Stuff...**
API

... stands for Application Programming Interface

A demo of vPython's API:

```python
# the simplest possible vpython program:
box( color = vector(1, 0.5, 0) )

# try changing the color: the components are
# red, green, blue each from 0.0 to 1.0

# then, add a second parameter: size=vector(2.0,1.0,0.1)
# the order of those three #s: Length, Height, Width

# then, a third parameter: axis=vector(2,5,1)
# the order of those three #s: x, y, z
```

vPython example API call(s)

What's box?
What's color?
What's vector?
Getting used to everything!
API stands for Application Programming Interface

Here is how to create a box object:

```python
mybox = box(
    pos=vec(x0, y0, z0),
    size=vec(L, H, W)
)
```

The given position is in the center of the box, at (x0, y0, z0). This is different from cylinder, whose pos attribute is at one end of the cylinder. Just as with a cylinder, we can refer to the individual vector components of the box as mybox.pos.x, mybox.pos.y, and mybox.pos.z. For this box, we have mybox.axis = vec(1, 0, 0). Note that the axis of a box is just like the axis of a cylinder.

For a box that isn’t aligned with the coordinate axes, additional issues come into play. The orientation of the length of the box is given by the axis:

```python
mybox = box(
    pos=vec(x0, y0, z0),
    axis=vec(a, b, c),
    size=vec(L, H, W)
)
```

The axis attribute gives a direction for the length of the box, and the length, height, and width of the box are given as before.

You can rotate the box around its own axis by changing which way is “up” for the box, by specifying an up attribute for the box that is different from the up vector of the.
The **compound** object lets you group objects together and manage them as though they were one object, by specifying in the usual way `pos`, `color`, `size` (and `length`, `width`, `height`), `axis`, `up`, `opacity`, `shininess`, `emissive`, and `texture`. Moreover, the display of a complicated compound object is faster than displaying the individual objects one at a time. (In GlowScript version 2.1 the details were somewhat different.)

The object shown above is a compound of a cylinder and a box:

```glow
handle = cylinder(  size=vec(1,.2,.2),
                   color=vec(0.72,0.42,0)  )

head = box(  size=vec(.2,.6,.2),
              pos=vec(1.1,0,0),
              color=color.gray(.6)  )

hammer = compound([handle, head])
hammer.axis = vec(1,1,0)
```

The **size of the object**: After creating the compound named "hammer", `hammer.size` represents the size of the bounding box of the object.
The **compound** object lets you group objects together and manage them as though they were one object, by specifying in the usual way `pos`, `color`, `size` (and `length`, `width`, `height`), `axis`, `up`, `opacity`, `shininess`, `emissive`, and `texture`. Moreover, the display of a complicated compound object is faster than displaying the individual objects one at a time. (In GlowScript version 2.1 the details were somewhat different.)

The object shown above is a compound of a cylinder and a box:

```glow
alien_body = sphere( size=1.0*vector(1,1,1), pos=vector(0,0,0), color=color.green )
alien_eye1 = sphere( size=0.3*vector(1,1,1), pos=.42*vector(.7,.5,.2), color=color.white )
alien_eye2 = sphere( size=0.3*vector(1,1,1), pos=.42*vector(.2,.5,.7), color=color.white )
alien_hat = cylinder( pos=0.42*vector(0,.9,-.2), axis=vector(.02,.2,-.02),
    size=vector(0.2,0.7,0.7), color=color.magenta )
alien_objects = [alien_body, alien_eye1, alien_eye2, alien_hat]
com_alien = compound( alien_objects, pos=starting_position )
```
GlowScript / vPython examples...

Hey!
I see what's happening here!

Theoretical and averaged speed distributions (meters/sec). Initially all atoms have the same speed, but collisions change the speeds of the colliding atoms. One of the atoms is marked and leaves a trail so you can follow its path.

Number of atoms

0 500 1000 1500 2000 2500 3000

10 8 5 3 2 1 0

10 by 10 by 10 = 1000 rotating cubes

59.1 renders/s * 2.1 ms/render = 123.3 ms rendering/s

Click a box to turn it white

Widgets (buttons, etc.)

Pause
Cyan

Vary the rotation speed:

Choose an object

Cyan
Red

1.50 radians/s

Surreal Stonehenge...
The projects...

VPool

Vironhagrid

TextID

Game!

Life

Picobot!
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:

0 xxxx -> N 0
0 Nxxx -> W 0
0 NxWx -> S 0
0 xxWx -> S 0
0 xxWS -> E 0
0 xxxxS -> E 0
0 xExS -> N 0
0 xExx -> N 0
0 NEexx -> S 1
1 xxxx -> S 1
1 Nxxx -> E 1
1 NxWx -> E 1
1 xxWx -> N 1
1 xxWS -> N 1
1 xxxxS -> W 1
1 xExS -> W 1
1 xExx -> S 1
1 NEexx -> W 0

class World:

design thoughts?

what classes that we've already used would work best for these two?
Picobot's classes

```python
class Program:
    # How in Python could we most usefully hold all of these rules?

    # What type should self.rules be?
```

The rules are:

- 0 xxxx -> N 0
- 0 Nxxx -> W 0
- 0 NxWx -> S 0
- 0 xxWx -> S 0
- 0 xxWS -> E 0
- 0 xxxxS -> E 0
- 0 xExS -> N 0
- 0 xExx -> N 0
- 0 NExx -> S 1
- 1 xxxx -> S 1
- 1 Nxxx -> E 1
- 1 NxWx -> E 1
- 1 xxWx -> N 1
- 1 xxWS -> N 1
- 1 xxxxS -> W 1
- 1 xExS -> W 1
- 1 xExx -> S 1
- 1 NExx -> W 0
Picobot's classes

```python
class Program:

    How in Python could we most usefully hold all of these rules?

    What type should `self.rules` be?

    a Python dictionary

    Both tuples

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

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

class World:

What class that you've already written will be most similar to Picobot's World?

What will *self.room* be?
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?
    __init__(self, environment):
        # What will `self.room` be?
        self.room = 

    def __repr__(self):
        return [row for row in self.room]
```

The same as the Game-of-Life's data, A!

here, a list-of-lists-of-one-character-strings....
Picobot's project

First, build an ASCII simulation

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!

http://rednuht.org/genetic_cars_2/ or http://boxcar2d.com/
Box2d: https://www.youtube.com/watch?v=uxourrlPlf8
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...

demo!

http://rednuht.org/genetic_cars_2/ or http://boxcar2d.com/
Box2d: https://www.youtube.com/watch?v=uxourrPlf8
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.

Start with a population of, say, ~200 *random* Picobot programs...
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.
program p1
fitness = 0.03

program p2
fitness = 0.05

mate + mutate the fittest
10-20% of programs
to create a new generation
of ~200 programs...

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 3\textsuperscript{rd} way?!
The projects...

- Life
- Picobot!
- Game!
- VPool
- TextID
What CSers "do"

think they

final-project *algorithms*...
A while back...

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.

First paragraph of *The Cuckoo's Calling* by R. Galbraith
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.
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 the 
Casual Vacancy, Ruth Rendell’s The St. Zita’s Patient and 
Val McDermid’s The Wire in the Blood.

I actually ran four separate types of analyses that focus on 
variables that are easy to compute from a given passage of 
language. One variable is the distribution of word lengths. Each novel has a 

length, and so one can get a robust vector of 

have exactly letters. Using a distance formula (I used the normalized cosine distance formula in the 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.

demo: CC vs WS vs CC vs JKR

that a lot of versus!
I like poptarts and 42 and spam. Spamful poptarts are like poptartful spams -- and are liked by all!

Will _Thanksgiving_ bring spam poptarts?

```python
class 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, 'spam': 4, 'i': 1, '42': 1, 'by': 1, 'will': 1, 'bring': 1, 'ar': 2, 'poptart': 4}  

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

{1: '!', 2: '-', 3: '?', 4: '_', 5: ':', 6: 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?

class TextModel

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

```python
dictionary_1 = {'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}

dictionary_2 = {0: 1, 1: 1, 2: 2, 3: 6, 4: 5, 5: 3, 7: 1, 8: 3, 10: 1, 12: 1}

dictionary_3 = {'and': 3, ':': 1, 'all': 1, 'like': 3, 'thanksgiving': 1, 'spam': 4, 'i': 1, '42': 1, 'by': 1, 'will': 1, 'bring': 1, 'ar': 2, 'poptart': 4}

dictionary_4 = {12: 1, 5: 1, 7: 1}

dictionary_5 = {'!': 1, '-': 2, '?': 1, '_': 2, ':': 1}
```

Each of these dictionaries is a different feature or model.
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?

**J.K. Rowling**

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

*word-count dictionary for J.K. Rowling*

**W. Shakespeare**

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

*word-count dictionary for W. Shakespeare*

**Word-count dictionary for an unknown author**

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

*does this better match JKR or WS? Why?*

**Extra!** What algorithm would you devise to quantify these dictionary-matches?!
Naïve Bayes classification

Bayesian spam filtering

From Wikipedia, the free encyclopedia

Bayesian spam filtering (/ˈbɛtʃən/ 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, the independent probability 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 expressed 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... what?!
Model matching

Suppose we have two trained models:

WS: { "love": 50, "spell": 8, "thou": 42 }  
JKR: { "love": 25, "spell": 275, "potter": 700 }

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

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
}

we 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

Suppose we have two normalized models:

How do we compare these models with an unknown text?

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

Pretend the words are all independent.

What's the *likelihood* of the unknown 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 }

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**

- **love**: 0.50, 0.50, 0.50
- **thou**: 0.42
- **potter**: 0.0, 0.0, 0.0, 0.0, 0.0

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

The probability of the unknown text is 0.0.

Correct, but not helpful!
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

\[ \frac{0.50 \cdot 0.50 \cdot 0.50 \cdot 0.42 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1}{0.0525} \]

Why is this especially incorrect?

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 }

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

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

What?

half of $\epsilon$!
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 }

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

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

so, take the log of everything

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

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

**Unknown text:**
- "love": 3
- "potter": 2
- "thou": 1
- "spam": 4

\[
3 \times \log(0.50) + \log(0.42) + 2 \times \log(0.012) + 4 \times \log(0.012)
\]

\[= -29.48\]

OK!

Take the \(\log_2\) of everything!
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*

\[
3\log(.025) + \log(.012) + 2\log(.7) + 4\log(.012)
\]

\[= -33.89\]

*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

Bayesian spam filtering (ˈbetʒən/ 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 hypothesis that is most probable, this is known as the maximum a posteriori decision rule. The corresponding classifier, a Bayes classifier, is the function of this decision rule:

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

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

Bayesian spam filtering

Don't take these formulas too seriously…

---

Bayesian spam filtering

The discussion so far has derived the independent feature model, that is, the naive Bayes model. The naive Bayes classifier combines this model with a decision rule. One can compute the hypothesis that is most probable; this is known as the maximum a posteriori decision. The corresponding classifier, a Bayes classifier, is the function classifier:

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

- VPool
- Life
- TextID
- Game!
- Picobot!
TextGame example...
```
import random
import webbrowser

class RPSGame:
    """ an RPS rivalry that tracks, saves, and loads games """

    def __init__(self):
        """ the constructor """
        self.num_comp_wins = 0
        self.num_user_wins = 0
        self.num_ties = 0

    def __repr__(self):
        """ the representation function """
        return "Rivalry Game: Comp Wins: {}, User Wins: {}, Ties: {}\n".format(self.num_comp_wins, self.num_user_wins, self.num_ties)

    def play_one_round(self):
        """ plays one game """
        # get the computer's choice
        comp = random.choice(['r', 'p', 's'])

        # get the user's choice
        user = input("Let's play! Choose r, p, or s: ")
        user = user.lower()

        # sanity-check user's input
        while user not in ['r', 'p', 's']:
            print("I didn't get what you meant by", user)
            user = input("Choose r, p, or s: ")
            user = user.lower()

        # judge the outcome
        outcome = 'tie'
        if user == comp:
            outcome = 'tie'
        elif user in ['r', 'sp', 'pr']:
            outcome = 'user'
        else:
            outcome = 'comp'

        # don't need the else, because of the default
        # print and save the results
        print("You chose", user,", self.lookup(user)")
        if outcome == 'tie':
            print("Aargh, We tied.
            self.num_ties += 1"
        elif outcome == 'comp':
            print("I win!
            self.num_user_wins += 1"
        else:
            print("You win!
            self.num_user_wins += 1"

    def menu(self):
        """ prints the menu """
        print("Welcome to the RPS Game Menu")
        print("1. Play Game")
        print("2. Load Game")
        print("3. Exit")

        uc = input("Please enter your choice: ")
        try:
            uc = int(uc)  # try converting to an integer
            if uc not in [1, 2, 4, 42]:  # Easter eggs welcome!
                print("Didn't recognize that input
            else:
                return uc  # must be a 1, 2, 4, or 42

        except ValueError as e:
            print("Didn't understand that input")
            # print("The error was: ", e)
            return None

    def play(self):
        """ hosts a series of games or turns """
        while True:
            userchoice = self.menu()  # prints the welcome and menu

            if userchoice == 1:
                self.play_one_round()
            elif userchoice == 2:
                print("Til next time!
            break
            elif userchoice == 4:
                self.save_game("gamefile.txt")
            elif userchoice == 2:
                self.load_game("gamefile.txt")
                print("Welcome back!
            elif userchoice == 42:
                # Lofi!
                webbrowser.open_new_tab("https://www.youtube.com/watch?v=5qpp5a0419A")"
TextGame ideas...

Nim

take one or two... whoever takes the final one loses
TextGame ideas...

Jotto

word-guessing game ~ with partial information
TextGame

ideas...

<table>
<thead>
<tr>
<th>Score</th>
<th>Opponent's Test Word</th>
<th>No. Of Jots</th>
<th>Your Test Word</th>
<th>No. Of Jots</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>FLASK</td>
<td>2</td>
<td>WHALE</td>
<td>1</td>
</tr>
<tr>
<td>95</td>
<td>LULLS</td>
<td>1</td>
<td>SHAKE</td>
<td>0</td>
</tr>
<tr>
<td>90</td>
<td>PLUMP</td>
<td>3</td>
<td>FLING</td>
<td>2</td>
</tr>
<tr>
<td>85</td>
<td>SLUMP</td>
<td>3</td>
<td>$4 VUNG</td>
<td>2</td>
</tr>
<tr>
<td>80</td>
<td>LYMPH</td>
<td>3</td>
<td>SLANG</td>
<td>2</td>
</tr>
<tr>
<td>75</td>
<td>NYMPH</td>
<td>2</td>
<td>GROAN</td>
<td>4</td>
</tr>
</tbody>
</table>

Jotto word-guessing game ~ with partial information
These are two cute

Let's try it!

'robot' 'otter'

\( \text{jscore}( \text{'robot'}, \text{'otter'} ) \rightarrow \)

\( \text{jscore}( S, T ) \quad \text{in general...} \)
```python
def lookup(self, abbreviation):
    # looks up the full name of each RPS gesture
    G = {'r': "rock", 's': "scissors", 'p': "paper")
    if abbreviation in G:
        return G[abbreviation]
    else:
        return "??"

def status(self):
    # print the current status
    print("Your wins:", self.num_user_wins)
    print("Ties:", self.num_ties)
    print("My wins:", self.num_comp_wins)

def menu(self):
    # prints the menu
    print("Menu:")
    print("1) Continue our RPS rivalry")
    print("2) Load our game")
    print("3) Save our game")
    print("4) Quit")
    return self.menu()

def play_one_round(self):
    # plays one game
    # get the computer's choice
    comp = random.choice(['r', 'p', 's'])
    # get the user's choice
    user = input("Let's play! Choose r, p, or s:")
    # sanity-check user's input
    while user not in ['r', 'p', 's']:
        print("I didn't get what you meant by", user)
        user = input("Choose r, p, or s:")
    # judge the outcome
    outcome = 'comp'
    if comp == user:
        outcome = 'tie'
    elif user in ['r', 'p', 's']:
        outcome = 'user'
    # don't need the else, because of the default
    # print and save the results
    print("You chose", self.lookup(user))
    print("I chose", self.lookup(comp))
    if outcome == 'tie':
        print("Aargh. We tied.")
        self.num_ties += 1
    elif outcome == 'comp':
        print("I win!")
        self.num_comp_wins += 1
    else:
        print("You win.")
        self.num_user_wins += 1

def play(self):
    # hosts a series of games or turns
    while True:
        userchoice = self.menu() # prints the menu and menu
        if userchoice == 1:
            self.play_one_round() # play one round.
        elif userchoice == 2:
            self.save_game("gamefile.txt")
        elif userchoice == 3:
            self.load_game("gamefile.txt")
        elif userchoice == 4:
            self.save_game("gamefile.txt")
            print("Welcome back!")
        elif userchoice == 42:
            # LoFi!
            webbrowser.open_new_tab("https://www.youtube.com/watch?v=5qpa5a0QfZA")
```

Lab today: Think about a project idea and – work on Finite-state-machines!