Quiz Section 2

Local alignments, P-values, `Python` datatypes

2019-04-11

Agenda

1. Review Dynamic Programming algorithm
2. Statistics: pvalues and multiple hypothesis testing
3. More `Python`

1. Dynamic Programming

Local alignment

- Please split into three groups
- Implement the Smith-Waterman algorithm on the board
- Report back your highest scoring local alignment

Local alignment data

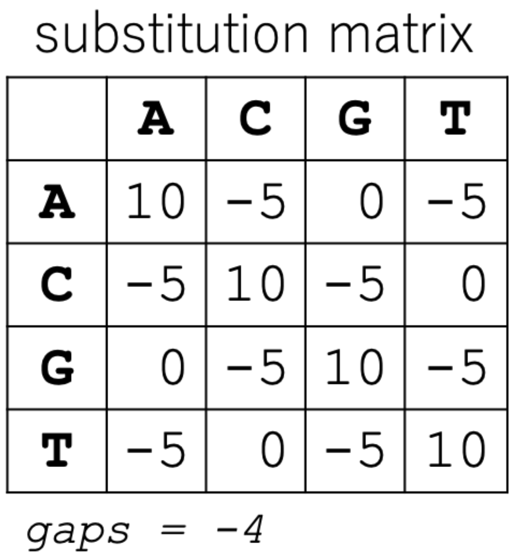
Please align `GAGTA` and `AGTTA`

Solution

2. P-values

What do P-values tell us?

- P-values tell you about expectations *under the null hypothesis*
- The null hypothesis is usually the *boring default*, the *devil's advocate position*, or *what you want to see if you can disprove.*
- Examples of null hypotheses: "there is *no* difference between treatment groups", "life expectantancy *is not* changing over time, the coin *is not* weighted, the two sequences are *unrelated.*



substitution matrix

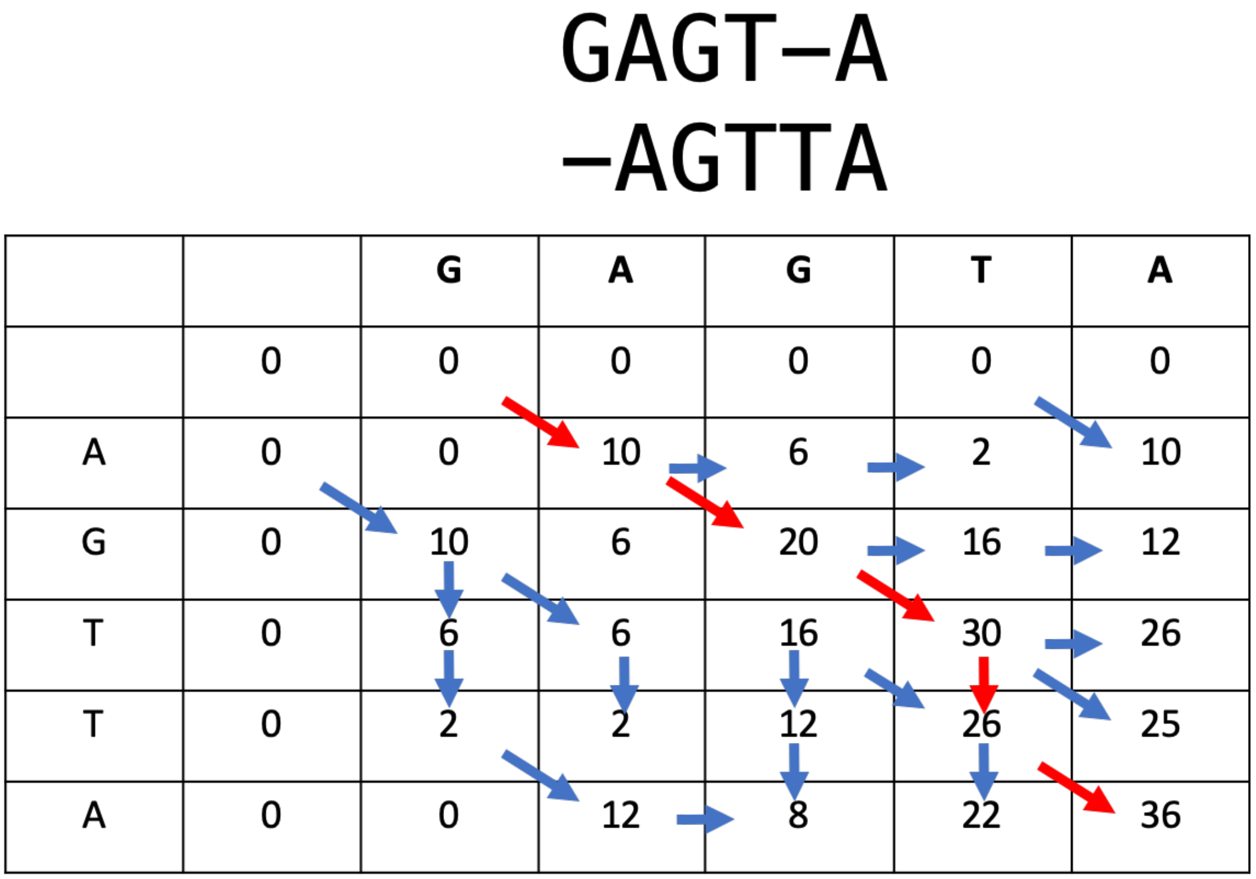
	A	C	G	T
A	10	-5	0	-5
C	-5	10	-5	0
G	0	-5	10	-5
T	-5	0	-5	10

gaps = -4

Figure 1:

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + s(x_i, y_j) \\ F(i-1,j) + d \\ F(i,j-1) + d \\ 0 \end{cases}$$

Figure 2:



```
GAGT–A
–AGTTA
```

		G	A	G	T	A
	0	0	0	0	0	0
A	0	0	10	6	2	10
G	0	10	6	20	16	12
T	0	6	6	16	30	26
T	0	2	2	12	26	25
A	0	0	12	8	22	36

Figure 3:

What *don't* P-values tell us?

- P-values say nothing about the alternative hypothesis or how probable it is.
- "We reject the null hypothesis" or "We fail to reject the null hypothesis"

"lady tasting tea" test

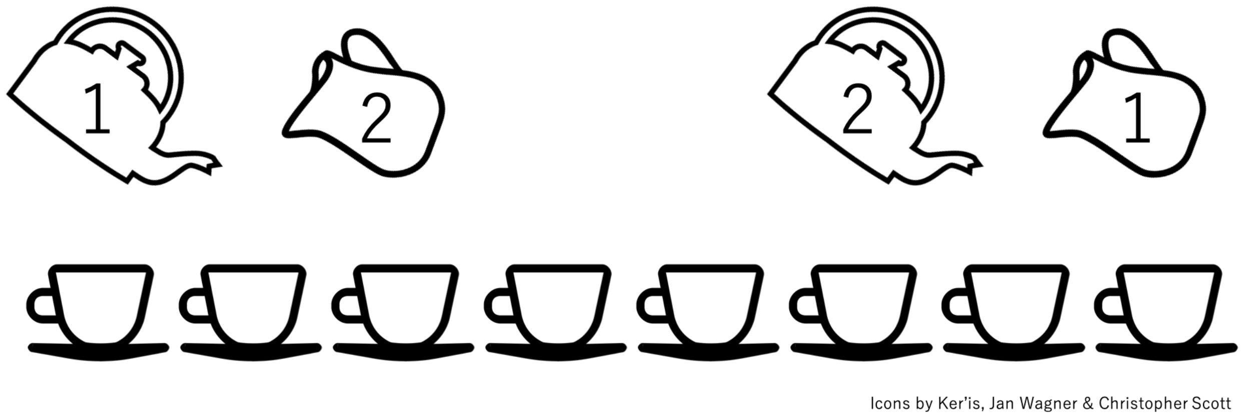


Figure 4:

Null distribution

What the data may look like if the null distribution is true. *How many guesses would be correct if the guess were random?*

(1) Parametrized probability distribution

For the lady tasting tea test, this was a hypergeometric distribution
What are some other distributions?

(2) Empirical null based on the real data

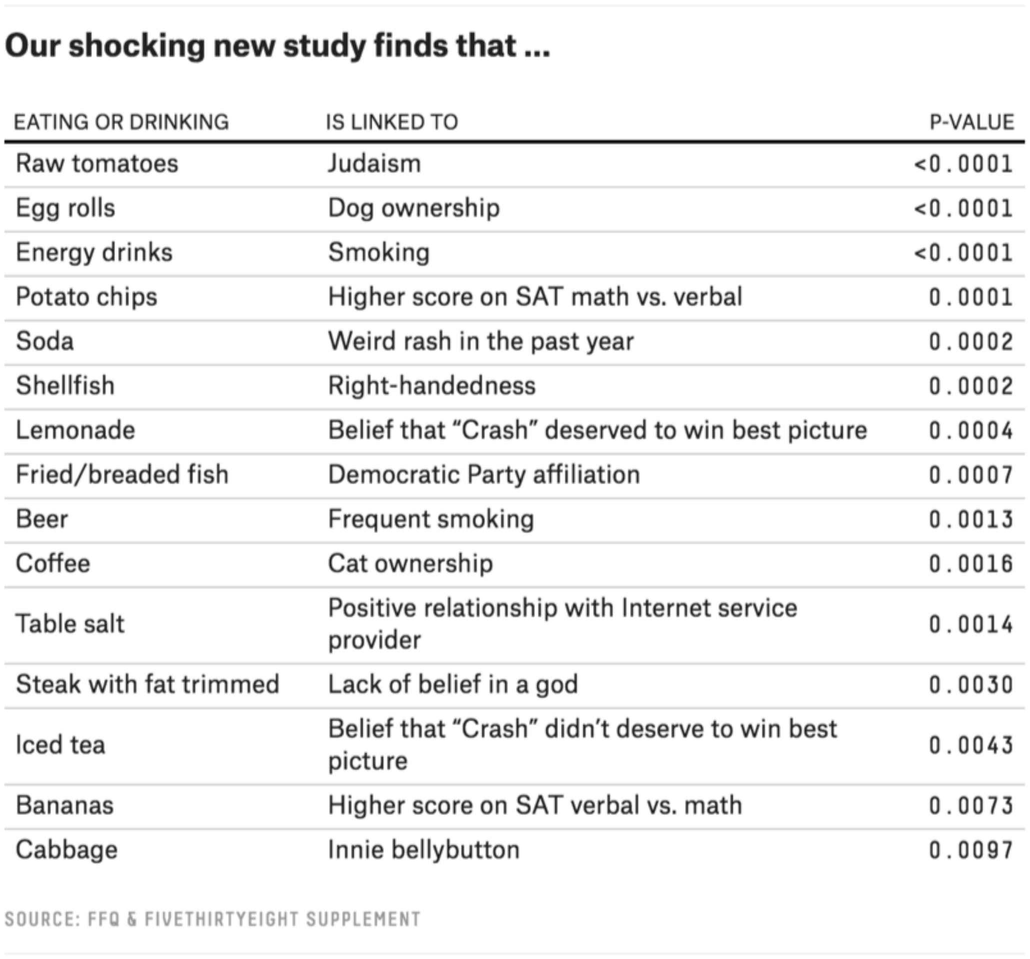
Shuffle the labels on your data

How do you define your null distribution?

Defining the most appropriate null distribution is a relevant and tough problem in a lot of computational biology!

Multiple hypothesis testing is dangerous!

FiveThirtyEight analyzed nutritional and lifestyle surveys from 54 individuals.



Our shocking new study finds that ...

EATING OR DRINKING	IS LINKED TO	P-VALUE
Raw tomatoes	Judaism	<0.0001
Egg rolls	Dog ownership	<0.0001
Energy drinks	Smoking	<0.0001
Potato chips	Higher score on SAT math vs. verbal	0.0001
Soda	Weird rash in the past year	0.0002
Shellfish	Right-handedness	0.0002
Lemonade	Belief that “Crash” deserved to win best picture	0.0004
Fried/breaded fish	Democratic Party affiliation	0.0007
Beer	Frequent smoking	0.0013
Coffee	Cat ownership	0.0016
Table salt	Positive relationship with Internet service provider	0.0014
Steak with fat trimmed	Lack of belief in a god	0.0030
Iced tea	Belief that “Crash” didn’t deserve to win best picture	0.0043
Bananas	Higher score on SAT verbal vs. math	0.0073
Cabbage	Innie bellybutton	0.0097

SOURCE: FFQ & FIVETHIRTYEIGHT SUPPLEMENT

Figure 5:

Multiple hypothesis correction

Bonferonni correction

For `n` tests, use a threshold that is `n`X stricter. (Divide your cutoff by `n`)

False discovery rate

Sometimes a Bonferonni correction is too harsh. Then, a false discovery rate correction may be more useful.

3. `Python` - datatypes.

Variables and types

Python has five built-in data types

- Number
- String
- List
- Tuple
- Dictionary

You can find an object's type using the type function.

```
x = 5
type(x)
```

Numbers

- The type number has two subtypes: integers (`int`) and floats (`float`)
- integer division is one of the changes between `Python2` and `Python3`
- To explicitly changes between types, use `int()` and `float()`. ex) `x = float(x)` will turn x into a `float`.

Operators

`Python` operator	operation	example
`**`	exponentiation	3**2 = 9
`*`	multiplication	4*2 = 6
`/`	division	4/2 = 2
`+`	addition	4+4 = 8
`-`	subtraction	3-2=1

Remember PENDAS?
*2*3**2 = ?*

Strings

- A series of characters starting or ending with single or double quotes.
- Stored a a list of characters in memory.
- `mystring = "GATTACA"`

0	1	2	3	4	5	6
G	A	T	T	A	C	A

Accessing strings: indices and splices

- You can reference a single character using its **index**

- You can reference a range of characters by creating a **splice**. The first number is **inclusive** and the second number is **exclusive**.

Accessing strings: indices and splices

command	`G`	`A`	`T`	`T`	`A`	`C`	`A`	result
s[0]	X							`G`
s[:3]	X	X	X					`GAT`
s[4:]					X	X	X	`ACA`
s[3:5]				X	X			`TA`
s[:]	X	X	X	X	X	X	X	`GATTACA`

String functionality (built-in functions)

```
len("GATTACA")  # 7

"GAT" + "TACA"  # GATTACA

"A" * 10  # AAAAAAAAAA

"GAT" in "GATTACA"  # True

"AGT" in "GATTACA"  # False
```

Methods

- In `Python`, a method is a function for a particular object type.
- The syntax is `<object>.<method>(<parameters)`

```
DNA = "AGT"
DNA.find("A")  # 0
```

String methods

```
"GATTACA".find("ATT")  # 1
"GATTACA".count("T")  # 2
"GATTACA".lower()  # 'gattaca'
"gattaca".upper()  # 'GATTACA'
"GATTACA".replace("G", "U")  # 'UATTACA'
"GATTACA".replace("C", "U")  # 'GATTAUA'
"GATTACA".replace("AT", "**")  # 'G**TACA'
"GATTACA".startswith("G")  # True
"GATTACA".startswith("g")  # False
```

Strings are immutable

String methods do not modify the string; they return a new string.

```
sequence = "ACGT"
sequence.replace("A", "G")  # 'GCGT'
print(sequence)  # "ACGT"

sequence = "ACGT"
new_sequence = sequence.replace("A", "G")
print(new_sequence)  # 'GCGT'
```

Reading input from the command line

- When you type `python sarahs_program.py 2 3`, `Python` sees a list of strings `["sarahs_program.py", "2", "3"]`
- You can access parts of this list using `sys.argv`
- `argv` is a `function` from the module `sys`
- You must `import sys` to use this function

Example of reading input from the command line

```
python sarahs_program.py 2 3
```

Let's look inside `sarahs_program.py`:

```
## Inside sarahs_program.py:
# Many functions only available via
# packages, you must import them
import sys
first_num = int(sys.argv[1])
second_num = float(sys.argv[2])
```

Sample problem

- Write a program called dna2rna.py that **reads a DNA sequence from the first command line argument, and then prints it as an RNA sequence.**

- Make sure it works for both uppercase and lowercase input.

```
> python dna2rna.py  ACTCAGT
ACUCAGU
> python dna2rna.py actcagt
acucagu
> python dna2rna.py ACTCagt
ACUCagu
```

Solution

```
import sys  # pull from command line
DNA = sys.argv[1]

# replace T with U (DNA -> RNA)
RNA = DNA.replace("T", "U")
RNA = DNA.replace("t", "u")
print(RNA)

# second solution
RNA = DNA.replace("T", "U").replace("t", "u")
```

Sample problem

- Write a program that takes a **DNA sequence as the first command line argument and prints the number of A's, T's, G's, and C's**

```
python dna-composition.py ACGTGCGTTAC
2 A's
3 C's
3 G's
3 T's
```

Solution

```
import sys

# grab the DNA sequence

DNA = sys.argv[1]

# make it uppercase
DNA = DNA.upper()

# count
A = DNA.count('A')
C = DNA.count('C')
G = DNA.count('G')
T = DNA.count('T')
print("{0} As\n{1} Cs\n{2 }Gs\n{3} Ts".format(A, C, G, T))
```

Lists

Lists are an ordered series of objects

```
list1 = ["sarah", "C", 3, 2.4]
list2 = [1, 2, 3]
list3 = [list1, list2]
list3  # [["sarah", "C", 3, 2.4], [1, 2, 3]]
```

Unlike strings, lists are mutable

```
list1 = ["sarah", "C", 3, 2.4]
list1[1] = "hilton"
list1  # ["sarah", "hilton", 3, 2.4]
```

Expanding lists

```
newlist = []
print(newlist)  # []

newlist.append(4)
print(newlist)  # [4]

newlist.extend([4,5])
print(newlist)  # [4,4,5]
```

More list methods

```
# add x to the end of L
L.append(x)
# add x and y to L
L.extend([x,y])
# count how many times x is in L
L.count(x)
 # give the location of x
L.index(x)
# remove first occurrence of x
L.remove(x)
# reverse order of elements of L
L.reverse(x)
 # sort L
L.sort()
```

Sample problem

- Write a program that takes a list of words and prints them out in sorted order

```
python sort_list.py z y x
['x', 'y', 'z']
```

Solution

```
import sys
iList = sys.argv[1:]
iList.sort()
print(iList)
```

Tuples

- Tuples are immutable lists. You can't change them in place (like strings)
- If you want to change them, you have to assign them to a new tuple

```
T = (1,2,3)
T[1] = 1  # Error

T = T + T
T  # (1,2,3,1,2,3)
```

More on conditionals

```
DNA = "AGTGGT"
if (DNA.startswith("A")):
    print("Starts with A")
```

```
if (<test evaluates to true>):
  <execute this block of code>
```

More on conditionals

- A block is a group of lines of code that belong together.

- `Python` uses indentation to keep track of blocks.
- You can use any number of spaces (or a tab) to indicate blocks, but you must be consistent.
- An unindented or blank line indicates the end of a block.
- In interactive mode, the ellipse indicates that you are inside a block.

Sample problem

- Write a program find-base.py that **takes as input a DNA sequence and a nucleotide.** The program should **print where the nucleotide occurs in the sequence, or a message saying it's not there.**

```
> python find-base.py A GTAGCTA
A occurs at position 3.
> python find-base.py A GTGCT
A does not occur at all.
```

Solution

```
import sys

base = sys.argv[1]
dna = sys.argv[2]

# solution 1
position = dna.find(base)
if position == -1:
  print("{0} does not occur at all.".format(base))
else:
  print("{0} occus in position {1}".format(base, position+1))

# solution 2
if base in dna:
  position = dna.find(base) + 1
  print("{0} occus in position {1}".format(base, position))
else:
  print("{0} does not occur at all.".format(base))
```

questions? skhilton@uw.edu

Week 2 tips + tricks

- Integer division is different between `Python2` and `Python3`
- `Python3` string formatting looks like this `"{0} {1} {0}".format("hello" "goodbye")` (`"hello goodbye hello"`)
- Strings must begin *and* end with quotes. If you forget, you may get an error like `SyntaxError: EOL while scanning string literal"`
- `Python` is 0 indexed. If `x = ["a", "b", "c"]` and you type `a[3]`, you may get an error like `IndexError: list index out of range`