Genome Sciences 373
Genome Informatics

Quiz Section 5

April 28, 2015

Bonferroni corrections

The motivation is to **minimize the probability of a single false-positive test** (Type I Error)

We often define alpha = 0.05

== *5% chance we reject the null hypothesis even when it's true*

Bonferroni corrections

The motivation is to **minimize the probability of a single false-positive test** (Type I Error)

We often define alpha = 0.05

== 5% chance we reject the null hypothesis even when it's true

The more tests we do, the probability grows quickly!

Bonferroni corrections

Bonferroni is **very conservative** – we will lose some “true signal” in order to not have false-positives

Bonferroni corrections

Input: list of p-values and alpha for one test
Output: list of p-values below a corrected threshold of significance

Procedure:
- Count ALL of the p-values
- Divide alpha by the total count
- Output each p-value if it is less than new_alpha

Some python caveats

You have two sequences (s1, s2)

You want to look for a gap (“-”) in the alignment.

```
for index in range(len(s1)):
  if s1[index] or s2[index] == “-”:
    print “found a gap!”
```

???

Some python caveats

You have two sequences (s1, s2)

You want to look for a gap (“-”) in the alignment.

```
for index in range(len(s1))
  if s1[index] or s2[index] == “-”:
    print “found a gap!”
```

XXXXX

Some python caveats

You have two sequences (s1, s2)

You want to look for a gap (“-”) in the alignment.

```
for index in range(len(s1)):
   if s1[index] == “-” or s2[index] == “-”:
      print “found a gap!”
```

Another python caveat

You are reading two sequences (s1, s2) from an alignment file

You want to check each position of the alignment for some condition

```
my_open_file = open(sys.argv[1])
s1 = my_open_file.readline()
s2 = my_open_file.readline()

for index in range(len(s1)):
   if s1[index] == s2[index]:
      number_of_matches += 1
```

Another python caveat

You are reading two sequences (s1, s2) from an alignment file

You want to check each position of the alignment for some condition

```
my_open_file = open(sys.argv[1])
s1 = my_open_file.readline()
s2 = my_open_file.readline()

for index in range(len(s1)):
   if s1[index] == s2[index]:
      number_of_matches += 1
```

XXXXX

Another python caveat

You are reading two sequences (s1, s2) from an alignment file

You want to check each position of the alignment for some condition

```
my_open_file = open(sys.argv[1])
s1 = my_open_file.readline().strip()
s2 = my_open_file.readline().strip()

for index in range(len(s1)):
   if s1[index] == s2[index]:
      number_of_matches += 1
```

Functions in Python: a brief overview

Functions are:

reusable pieces of code, that take zero or more **arguments**, perform some **actions**, and **return** one or more values

Functions in Python: a brief overview

Functions are:

reusable pieces of code, that take zero or more **arguments**, perform some **actions**, and **return** one or more values

conceptually

function "**sum**"
takes arguments a, b
adds a and b
returns sum

Functions in Python: a brief overview

Functions are:

reusable pieces of code, that take zero or more **arguments**, perform some **actions**, and **return** one or more values

conceptually

function "**sum**"
takes arguments a, b
adds a and b
returns sum

in python…

```
def sum(a, b):
    total = a + b
    return total

# later in the program
my_sum = sum(2, 5)
# my_sum is now 7
```

Functions in Python: a brief overview

Functions are:

reusable pieces of code, that take zero or more **arguments**, perform some **actions**, and **return** one or more values

stuff that happens in here is invisible outside of the function

in python…

```
def sum(a, b):
    total = a + b
    return total

# later in the program
my_sum = sum(2, 5)
print total   # this won't work!
```

```
def jc(seq1, seq2):
        # find the length of the alignment
        seqlength = len(seq1)

        # counters
        informative_pos = 0
        mismatch_pos = 0

        # progress through the sequence
        for index in range(seqlength):
                # ignore gaps
                if seq1[index] == "-" or seq2[index] == "-":
                        continue
                # look for mismatches
                if seq1[index] != seq2[index]:
                        mismatch_pos += 1
                # increment the counter of informative positions
                informative_pos += 1

        # find the raw distance: number of mismatches divided by the number
        # of informative positions
        raw_distance = float(mismatch_pos) / float(informative_pos)

        # calculate the Jukes-Cantor distance from the raw distance
        jc_distance = -0.75 * math.log(1.0 - (4.0/3.0 * raw_distance))
        return jc_distance
```

function to find Jukes-Cantor distance

In-class example:

Write a function to calculate the factorial of an integer