Quiz Section Week 4
April 19, 2016

Parsimony Algorithm
K-means clustering algorithm
Fitch algorithm, bottom up phase: what is the parsimony score?

1. Initialization: \( R_i = \{s_i\} \)
2. Traverse the tree from leaves to root ("post-order")
3. Determine \( R_i \) of internal node \( i \) with children \( j, k \):
   \[
   R_i = \begin{cases} 
   R_j \cap R_k \neq \emptyset & \rightarrow R_j \cap R_k \\
   \text{otherwise} & \rightarrow R_j \cup R_k 
   \end{cases}
   \]

1 change parsimony score of 1
What is a possible pair of sequences consistent with the score?

Total: $1 + 2 + 1 = 4$
2. Fitch’s algorithm: Top-down phase

*(Pick a state for each internal node)*

1. Pick arbitrary state in $R_{\text{root}}$ to be the state of the root, $s_{\text{root}}$
2. Traverse the tree from root to leaves ("pre-order")
3. Determine $s_i$ of internal node $i$ with parent $j$:

$$s_i = \begin{cases} 
    s_j & \text{if } s_j \in R_i \\
    \text{otherwise } \rightarrow \text{arbitrary state } \in R_i 
\end{cases}$$
K-mean clustering algorithm

- The number of centers, \( k \), has to be specified a-priori

**Algorithm:**

1. Arbitrarily select \( k \) initial centers
2. Assign each element to the closest center
3. Re-calculate centers (mean position of the assigned elements)
4. Repeat 2 and 3 until one of the following termination conditions is reached:
   i. The clusters are the same as in the previous iteration
   ii. The difference between two iterations is smaller than a specified threshold
   iii. The maximum number of iterations has been reached

How can we do this efficiently?
How can we represent 2D points?
Tuples are like lists but immutable

```python
# Define points in 2D space
point_a = (1, 2)
print(point_a)
(1, 2)
print(point_a[0])
1
print(point_a[1])
2
point_b = (3, 4)
points = [point_a, point_b]  # You can put them in lists
print(points)
[(1, 2), (3, 4)]
```
How can we store the points and their corresponding assigned clusters?
How can we store the points and their corresponding assigned clusters?

```python
points = []  # Contains a list of points (tuples)
clusters = []  # A list of strings that will contain the cluster assigned to each point
# For instance, the point denoted by points[3] is assigned to clusters[3].
points.append((3,1))
clusters.append('A')
points.append((5,2))
clusters.append('B')
print(points, clusters)
```

```
[(3, 1), (5, 2)] ['A', 'B']
```
K-mean clustering algorithm

- The number of centers, $k$, has to be specified a-priori

**Algorithm:**

1. Arbitrarily select $k$ initial centers
2. Assign each element to the closest center
3. Re-calculate centers (mean position of the assigned elements)
4. Repeat 2 and 3 until one of the following termination conditions is reached:
   
   i. The clusters are the same as in the previous iteration
   ii. The difference between two iterations is smaller than a specified threshold
   iii. The maximum number of iterations has been reached

How can we do this efficiently?
Write a function to compute the distance between two points

The distance between two points \((x_1, y_1)\) and \((x_2, y_2)\) is

\[
\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}
\]

# to compute \(a^b\), use the function \(\text{pow}(a, b)\)
print \text{pow}( 4, 0.5 )
2.0

\(a = (1,4)\)
\(b = (2,6)\)
def distance( a, b ):
    # YOUR CODE HERE
    return \text{pow}( \text{pow}(a[0]-b[0],2) + \text{pow}(a[1]-b[1],2), 0.5 ) \)
K-mean clustering algorithm

The number of centers, $k$, has to be specified a-priori

Algorithm:

1. Arbitrarily select $k$ initial centers
2. Assign each element to the closest center
3. Re-calculate centers (mean position of the assigned elements)
4. Repeat 2 and 3 until one of the following termination conditions is reached:
   i. The clusters are the same as in the previous iteration
   ii. The difference between two iterations is smaller than a specified threshold
   iii. The maximum number of iterations has been reached

How can we do this efficiently?
Exercise: given a list of 2D points, compute their center

points = [ (1,2), (3,4), (5,6), (7,8) ]
# center point is ( mean_x, mean_y )
mean_x = 0.0
mean_y = 0.0
for i in range(0,len(points)):
    mean_x += points[i][0]
    mean_y += points[i][1]

center = ( mean_x/len(points), mean_y/len(points) )
print center
(4.0, 5.0)
Split a string into its constituent words

```
s = 'Wherefore art thou Romeo?'
words = s.split()  # Returns a list of substrings
print words

['Wherefore', 'art', 'thou', 'Romeo?']

# split() can use any arbitrary string to split by
words = s.split('r')
print words

['Whe', 'efo', 'e a', 't thou Romeo?']
```
Breaking out of a for loop

```python
ts = ‘tacocat’
# Print ‘Not palindrome!’ if not
# Do nothing if it is

middle = int(len(s))/2 # int(7/2)=3
for i in range(0,middle):
    reverse_index = -1-i
    if s[i] == s[reverse_index]:
        continue
    else:
        print ‘Not palindrome!’
        break
```