Practical Bioinformatics

Mark Voorhies

6/20/2010

Clustering exercises – Scripting Cluster

Modify the clustering protocol script to run Cluster3 multiple times on the same input, varying distance metric and/or clustering method. Be sure to give the output files distinct names.

Clustering exercises – Scripting Cluster

Modify the clustering protocol script to run Cluster3 multiple times on the same input, varying distance metric and/or clustering method. Be sure to give the output files distinct names.

```
metrics = ("None",
           "Uncentered",
           "Pearson",
           "UncenteredAbs",
           "PearsonAbs",
           "Spearman",
           "Kendall",
           "Euclidean",
           "City")
linkage = (("Complete","m"),
           ("Single","s"),
           ("Centroid","c"),
           ("Average","a"))

# Loop over all 32 possible methods
print "Starting hierarchical clustering runs..."
from subprocess import check_call
for metric in xrange(1,len(metrics)):
    print "    ",metrics[metric],"..."
    for (linkname, link) in linkage:
        print "        ",linkname
        check_call(("cluster","-f","shuffled.txt",
                    "-u",".".join(("shuffled",
                                   metrics[metric],
                                   linkname)),
                    "-m",link,"-g",str(metric)))
```

Clustering exercises – Negative controls

Add functions to TdtRatios to reproduce the shuffling controls in figure 3 of the Eisen paper (removing correlations among genes and/or arrays).

Clustering exercises – Negative controls

Add functions to TdtRatios to reproduce the shuffling controls in figure 3 of the Eisen paper (removing correlations among genes and/or arrays).

```
def shuffleRows(self, seed = None):
    """Permute ratio values within rows."""
    import random
    if(seed != None):
        random.seed(seed)
    for i in self.ratios:
        random.shuffle(i)
```

Clustering exercises – Negative controls

Add functions to TdtRatios to reproduce the shuffling controls in figure 3 of the Eisen paper (removing correlations among genes and/or arrays).

```
def shuffleRows(self, seed = None):
    """Permute ratio values within rows."""
    import random
    if(seed != None):
        random.seed(seed)
    for i in self.ratios:
        random.shuffle(i)


def shuffleCols(self, seed = None):
    """Permute ratio values within columns."""
    import random
    if(seed != None):
        random.seed(seed)
    # Transpose the expression matrix
    cols = []
    for col in xrange(len(self.ratios[0])):
        cols.append([row[col] for row in self.ratios])
    # Shuffle
    for i in cols:
        random.shuffle(i)
    # Transpose back to original orientation
    self.ratios = []
    for row in xrange(len(cols)):
        self.ratios.append([col[row] for col in row])
```

Clustering exercises – JavaTreeView

Cluster supp2data.tdt and explore the results in JavaTreeView. Can you identify the clusters from figure 2 of the Eisen paper. Click on gene names to open the corresponding SGD annotations in your web browser. Are the current annotations consistent with those in supp2data.tdt? Are they consistent with the clustering pattern?

Sets

```
s1 = set((1,2,3,4,5))
s2 = set((1,3,5,7))
s1.union(s2) == set((1,2,3,4,5,7))
s1.difference(s2) == set((2,4))
s1.intersection(s2) == set((1,3,5))
```

Sets

```
cluster1 = set(i.strip() for i in open("cluster1.uids"))
cluster2 = set(i.strip() for i in open("cluster2.uids"))
cluster3 = set(i.strip() for i in open("cluster3.uids"))

cluster1.intersection(cluster2)
cluster1.intersection(cluster2).intersection(cluster3)
cluster1.intersection(cluster2).difference(cluster3)
```

Sets

```
cluster1 = set(i.strip() for i in open("cluster1.uids"))
cluster2 = set(i.strip() for i in open("cluster2.uids"))
cluster3 = set(i.strip() for i in open("cluster3.uids"))

cluster1.intersection(cluster2)
cluster1.intersection(cluster2).intersection(cluster3)
cluster1.intersection(cluster2).difference(cluster3)
```

1. Export several overlapping UID lists from Java TreeView and use sets to find their intersections.
2. Export UID lists for similar clusters from two different CDT files and use sets to compare them. Explore the non-intersecting elements in Java TreeView.
3. Use sets of gene names to compare your clusters to the annotated clusters in figure 2 of the Eisen paper.

Clustering exercises – Visualizing the distance matrix

Write your pairwise distance matrix to a CDT file (in this case, the rows and columns are *both* genes) and visualize it in JavaTreeView.

Clustering exercises – Visualizing the distance matrix

Write your pairwise distance matrix to a CDT file (in this case, the rows and columns are *both* genes) and visualize it in JavaTreeView.

```
class DistanceMatrix:
    def __init__(self, genes, annotations, ratios, metric):
        self.genes = genes
        self.annotations = annotations
        self.distances = []
        for i in self.ratios:
            self.distances.append([])
            for j in self.ratios:
                self.distances[-1].append(metric(i,j))

    def writeCDT(self, filename):
        fp = open(filename,"w")
        fp.write("\t".join(["GID","UNIQID","NAME"]+
                           ["GWEIGHT"]+self.genes)+"\n")
        fp.write("\t".join(["EWEIGHT"]+[""]*3+
                           ["1.0"]*len(self.genes))+"\n")
        for i in range(len(self.genes)):
            fp.write("GENE%4dX" % (i+1))
            fp.write("\t"+self.genes[i])
            fp.write("\t"+self.annotations[i])
            fp.write("\t1.0")
            for j in self.distances[i]:
                if(j == None):
                    fp.write("\t"+"")
                else:
                    fp.write("\t%f" % j)
            fp.write("\n")
```

Clustering exercises – Visualizing the distance matrix

Write your pairwise distance matrix to a CDT file (in this case, the rows and columns are *both* genes) and visualize it in JavaTreeView.

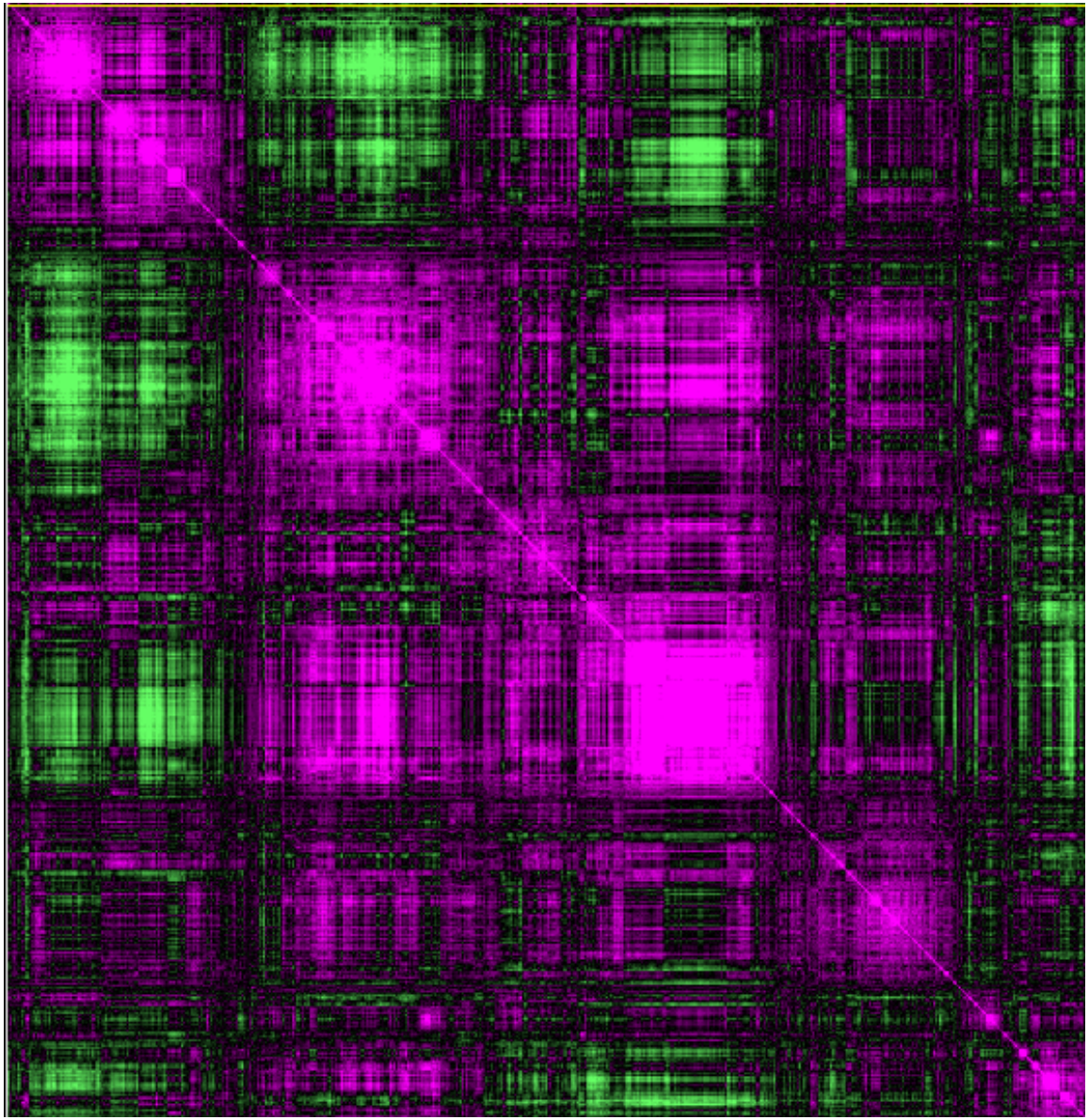
```
class DistanceMatrix:
    def __init__(self, genes, annotations, ratios, metric):
        self.genes = genes
        self.annotations = annotations
        self.distances = []
        for i in self.ratios:
            self.distances.append([])
            for j in self.ratios:
                self.distances[-1].append(metric(i,j))

    def writeCDT(self, filename):
        fp = open(filename,"w")
        fp.write("\t".join(["GID","UNIQID","NAME"]+
                           ["GWEIGHT"]+self.genes)+"\n")
        fp.write("\t".join(["EWEIGHT"]+[""]*3+
                           ["1.0"]*len(self.genes))+"\n")
        for i in range(len(self.genes)):
            fp.write("GENE%4dX" % (i+1))
            fp.write("\t"+self.genes[i])
            fp.write("\t"+self.annotations[i])
            fp.write("\t1.0")
            for j in self.distances[i]:
                if(j == None):
                    fp.write("\t"+"")
                else:
                    fp.write("\t%f" % j)
            fp.write("\n")
```

Left as exercises for the reader:

1. Rewrite `__init__` to avoid redundant calls for (i,j) and (j,i).
2. Add an option to load a matrix from a CDT file rather than calculating it.
3. Rewrite the class to store only the upper triangle of the matrix. Can you provide an interface that mimics storing the full matrix?

Clustering exercises – Visualizing the distance matrix



Dictionaries

```
dictionary = {"A":"T","T":"A","G":"C","C":"G"}
dictionary["G"]
dictionary["N"] = "N"
dictionary.has_key("C")
```

Dictionaries

```
geneticCode = {"TTT":"F","TTC":"F","TTA":"L","TTG":"L",
               "CTT":"L","CTC":"L","CTA":"L","CTG":"L",
               "ATT":"I","ATC":"I","ATA":"I","ATG":"M",
               "GTT":"V","GTC":"V","GTA":"V","GTG":"V",

               "TCT":"S","TCC":"S","TCA":"S","TCG":"S",
               "CCT":"P","CCC":"P","CCA":"P","CCG":"P",
               "ACT":"T","ACC":"T","ACA":"T","ACG":"T",
               "GCT":"A","GCC":"A","GCA":"A","GCG":"A",

               "TAT":"Y","TAC":"Y","TAA":"*","TAG":"*",
               "CAT":"H","CAC":"H","CAA":"Q","CAG":"Q",
               "AAT":"N","AAC":"N","AAA":"K","AAG":"K",
               "GAT":"D","GAC":"D","GAA":"E","GAG":"E",

               "TGT":"C","TGC":"C","TGA":"*","TGG":"W",
               "CGT":"R","CGC":"R","CGA":"R","CGG":"R",
               "AGT":"S","AGC":"S","AGA":"R","AGG":"R",
               "GGT":"G","GGC":"G","GGA":"G","GGG":"G"}
```

Exercise: Transforming sequences

1. Write a function to return the antisense strand of a DNA sequence in 3'→5' orientation.
2. Write a function to return the compliment of a DNA sequence in 5'→3' orientation.
3. Write a function to translate a DNA sequence

Exercise: Scoring an ungapped alignment

$$S(x, y) = \sum_{i}^{N} s(x_i, y_i) \tag{1}$$

1. Given two equal length sequences and a scoring matrix, return the alignment score for a full length, ungapped alignment.

Exercise: Scoring a gapped alignment

$$S_{gapped}(x, y) = S(x, y) + \sum_{i}^{gaps} G + E * len(i) \quad (2)$$

1. Given two equal length gapped sequences (where "-" represents a gap) and a scoring matrix, calculate an alignment score with a -1 penalty for each base aligned to a gap.
2. Write a new scoring function with separate penalties for opening a zero length gap (*e.g.*, G = -11) and extending an open gap by one base (*e.g.*, E = -1).