Choose Your Own Adventure in Data Analysis: Clustering

Yee Whye Teh
Gatsby Computational Neuroscience Unit
UCL

Adaptive Modelling of Complex Data
UCL

Outline

3: Choose Your Own Adventure

You are a novice learning the ropes in analyzing and modelling complex data. This morning you have just added clustering techniques into your arsenal of weapons to dissect and analyze data.

This afternoon you would like to master them well by implementing and applying them to simple and real world datasets.

Code and data package [▶4].

The Components of your adventures:

- Implement the clustering algorithms [▶5];
- Prepare your datasets [▶5];
- Example Adventures applying clustering algorithms to your datasets [▶6].

You do not need to do these in linear order, and you are not expected to do everything due to time constraints. You are encouraged to band together in your endeavours to help each other. Lloyd (a tutor) and I will be on hand to answer any questions. Good luck!

4: Code and Data Package

The code and data package can be obtained from my teaching webpage:

- `http://www.gatsby.ucl.ac.uk/~ywteh/teaching/teaching.html`

Run `initpath` in the main directory in MATLAB. It will add the subdirectories `utilities` and `datasets` into your MATLAB path.

5: Components of Adventures

Your weapons of mass analysis:

- K-means [▶16];
- Spectral clustering [▶24];
- Linkage algorithms [▶28];
- Mixture models [▶40].

Your targets of analysis:

- Create your own 2D dataset [▶7];
- Cancer of Golub [▶9];
- You are encouraged to try tackling your own dataset!

6: Example Adventures

- Seeing K-means getting stuck in local optima [▶31];
- Varying the number of clusters in K-means [▶32];
- Applying K-means to Cancer of Golub [▶33];
- Exploring different linkage algorithms [▶37];
- Try out spectral clustering on small datasets [▶39];
- Explore overfitting with mixtures of Gaussians [▶40].

7: Practice Datasets

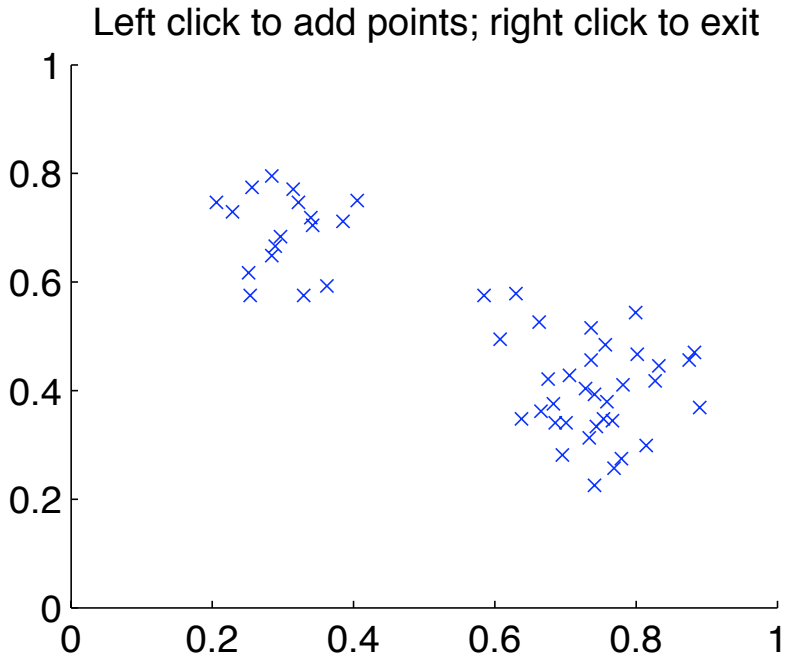
Rather than starting with large and complicated datasets, it is better to start with small and simple datasets for which you know the true answer and can debug your code easily.

You can create such a dataset using the `getpoints` command. Try:

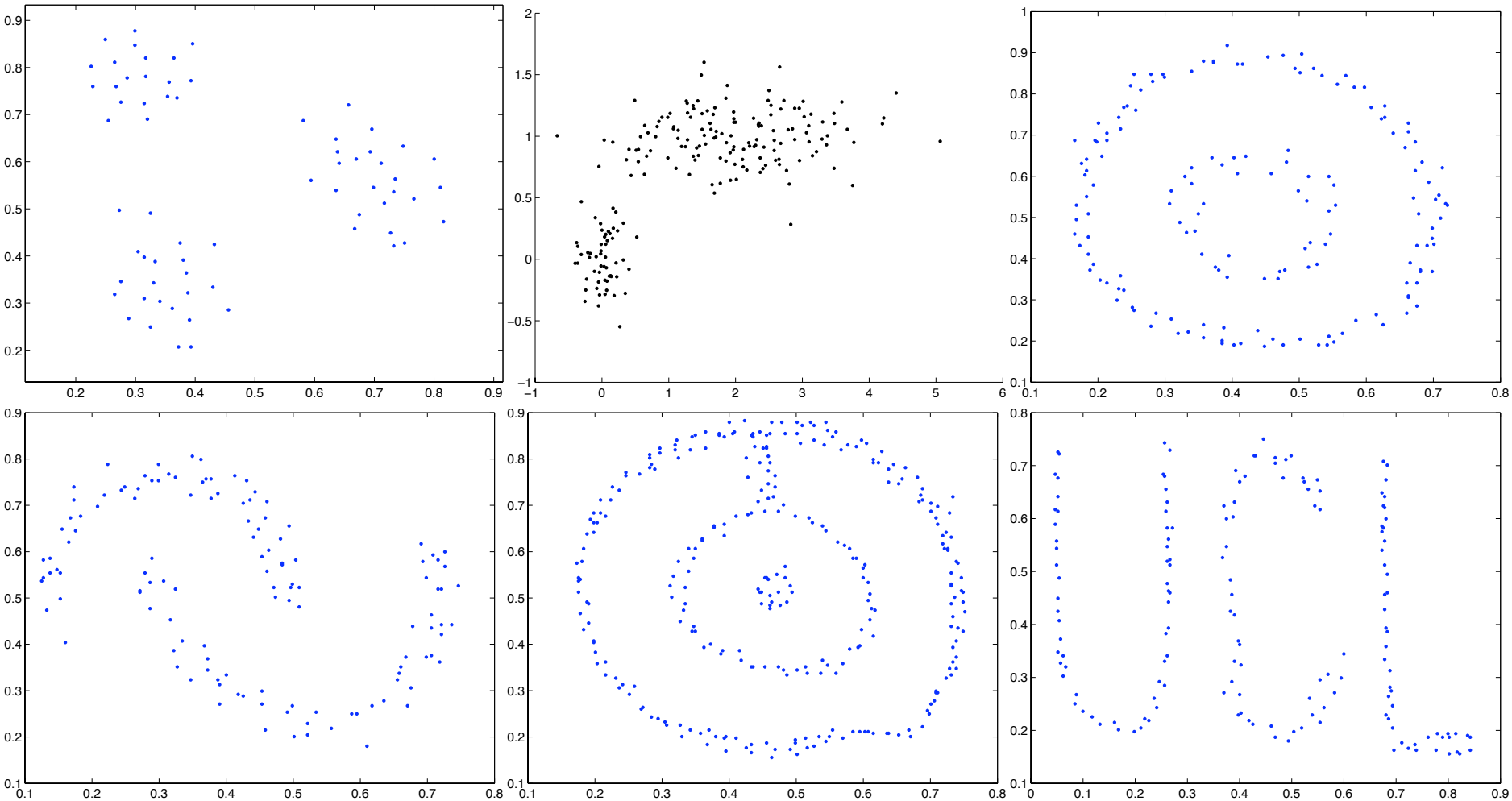
```
X = getpoints;
```

Returned `X` stores a 2D point in each row.

Some examples [▶8] of practice datasets.



8: Example Practice Datasets



9: Cancer of Golub

The Cancer of Golub is an old microarray dataset of gene expression levels among 72 leukemia patients. The paper, which was published in Science Magazine on 15 Oct 1999, proposed a number of data analysis techniques to discover and predict cancer subtypes using the expression levels of many genes. Alas times change, and the proposed data analysis tools are a little naïve by today's standards.

Your goal is to use clustering techniques to discover cancer subtypes, and to use logistic regression and classification techniques to predict cancer subtypes.

- The Golub et al site [▶`http://www.broad.mit.edu/cgi-bin/cancer/publications/pub_paper.cgi?mode=view&paper_id=43`];
- The Golub data [▶10];
- Preparing [▶11] the data;

10: Cancer of Golub: Data

In the software package, under `datasets` directory you'll find the `golub.mat` file contain the data. The variables in the file are:

`numgenes`	Number of genes (7129)
`numtrain`	Number of patients in training set (38)
`numtest`	Number of patients in test set (34)
`traindata`	Gene expression levels in training set (38x7129)
`testdata`	Gene expression levels in test set (34x7129)
`genedesciption`	Descriptions of genes (7129 cell)
`geneid`	Unique ID for genes (7129 cell)

You need to preprocess [▶11] the data first.

11: Cancer of Golub: Data Preparation

Microarray datasets are well-known to require quite a bit of normalization before they can be used due to the technology involved. Normalization cleans up noise specific to microarray data, conditions data so that analyses are not dominated by a few (outlying) data points, and sometimes extracts parts of the data that is informative to the task at hand.

We will perform some simple normalizations:

1. Subtract the median expression level for each patient; details [▶12]
2. Normalize for differential binds of probes for each gene; details [▶13]
3. Take the logarithm of expression levels; details [▶13]
4. Normalize again so that we have mean 0 and variance 1. details [▶14]

Want to skip this step? Normalization code [▶15]

12: Cancer of Golub: Data Preparation 1

Microarray uses an imaging technology to determine expression levels of genes. Roughly speaking, a microchip is tiled with a large number of probes, each probe attracts a small chunk of cDNA or cRNA corresponding to a gene, these chunks have a fluorescent dye attached, and an imaging technology is used to determine the intensity of the dye at each location (pixel) of the microarray. The more highly expressed a gene is, the more cDNA or cRNA chunks are available to attach to probes, and the brighter the corresponding pixel.

The gene expression levels of different patients are imaged on different microarrays. The overall intensity of each microarray varies due to different experimental conditions and different optical exposure lengths.

The first step is to normalize intensity levels across patients. We subtract median as this is more stably estimated than the mean.

```
data = [traindata;testdata];
mm   = median(data,2);
data = data-mm(:,ones(1,numgenes));
```

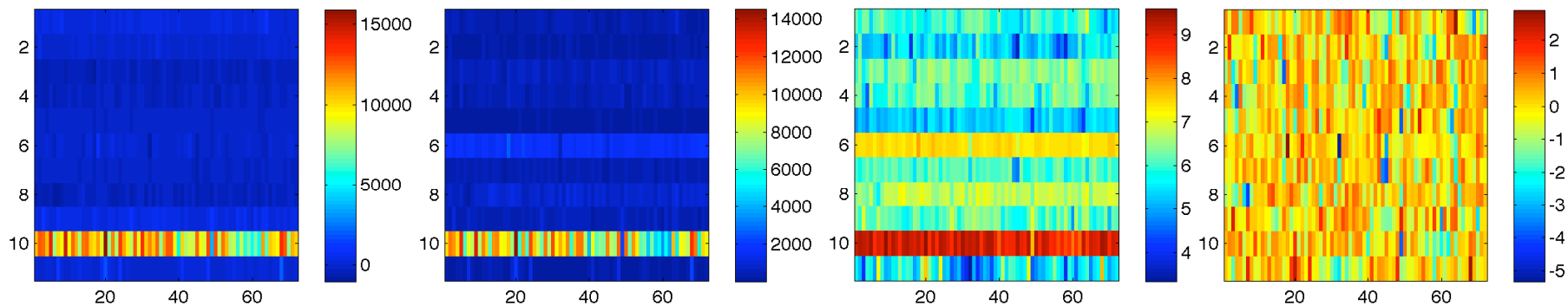
13: Cancer of Golub: Data Preparation 2

Different probes have widely varying binding affinities. The second step is to normalize for this. First we subtract the minimum values of intensities (to make the intensities positive again):

```
nn   = numtrain+numtest;
mm   = min(data,[],1);
data = data-mm(ones(1,nn),:);
```

Then we add in a little fudge factor to get rid of zeroes, and take logs:

```
ss   = sort(data);
pp   = ss(4,:);
data = pp(ones(1,nn),:)+data;
data = log(data);
```



14: Cancer of Golub: Data Preparation 3

Finally we have to normalize each gene to have zero mean and variance 1 across patients:

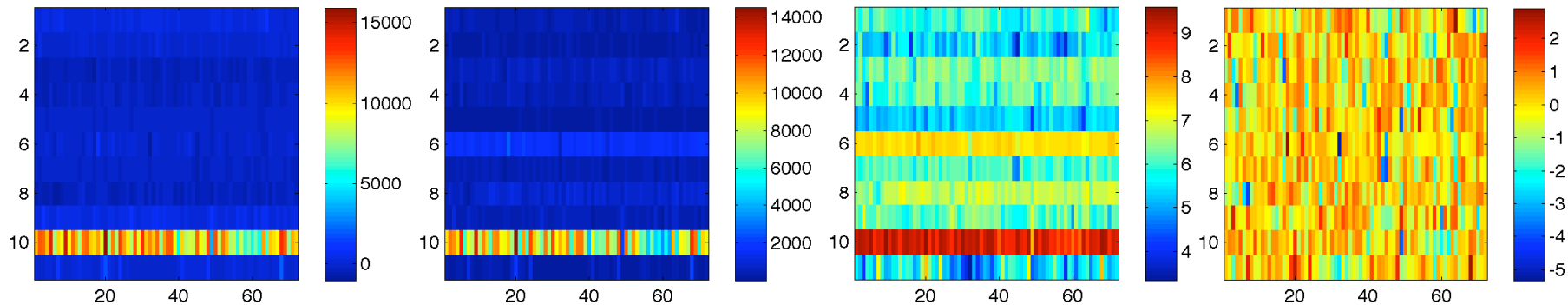
```
mm   = mean(data,1);
ss   = std(data,[],1);
data = (data - mm(ones(1,nn),:))./ss(ones(1,nn),:);
```

The normalized training and test sets are:

```
trainnormd = data(1:numtrain,:);
testnormd  = data(numtrain+(1:numtest),:);
```

Figures are shown with command

```
imagesc(data(:,1000:1010));colorbar;
```



15: Cancer of Golub: Normalization Code

```
% normalize data
data = [traindata;testdata];     % full dataset
nn   = numtrain+numtest;
mm   = median(data,2);           % subtract patient median
data = data-mm(:,ones(1,numgenes));
mm   = min(data,[],1);           % subtract gene minimum
data = data-mm(ones(1,nn),:);
ss   = sort(data);
pp   = ss(4,:);                  % use gene specific scale
data = pp(ones(1,nn),:)+data;
data = log(data);                % log expression levels
mm   = mean(data,1);             % mean 0 variance 1
ss   = std(data,[],1);
data = (data - mm(ones(1,nn),:))./ss(ones(1,nn),:);
trainnormd = data(1:numtrain,:);
testnormd  = data(numtrain+(1:numtest),:);
```

Back to Cancer of Golub main page [▶9]

16: K-means

K-means is an iterative algorithm to minimize the distortion function:

$$J(R, \boldsymbol{\mu}) = \sum_{i=1}^{m} \sum_{c=1}^{K} r_{ic} \|\mathbf{x}_i - \mu_c\|^2$$

It iterates between updating R and updating $\boldsymbol{\mu}$:

$$R \Leftarrow \operatorname*{argmin}_{R} J(R, \boldsymbol{\mu})$$

$$\boldsymbol{\mu} \Leftarrow \operatorname*{argmin}_{\boldsymbol{\mu}} J(R, \boldsymbol{\mu})$$

Write a MATLAB function implementing K-means. Such a function will have to contain 4 parts: Details [▶17]

- Initialization for R and $\boldsymbol{\mu}$;
- Iterative updates for R and $\boldsymbol{\mu}$;
- Test for convergence of algorithm;
- Special clause for when a cluster becomes empty due to an update.

17: K-means: Implementation Details

- Initialization for R and μ:
  Easiest approach is to randomly assign each data item one of the K clusters (thus getting R), and computing the mean of each cluster based on this initial assignment. Details [▶18]
- Iterative updates for R and μ:
  Each iteration first runs through the data items, assigning it to the cluster whose mean is closest, then runs through the clusters, computing a new mean based on the data items assigned to it. Details [▶19]
- Test for convergence of algorithm:
  K-means converges once the assignment of data items to clusters in the iteration above does not change the cluster assignments. Details [▶20]
- Special clause for when a cluster becomes empty due to an update:
  Once in a while a cluster becomes emptied because all the data items in it are assigned to other clusters. Easiest approach to dealing with this is to "steal" a data item at random and force it to be assigned to this cluster. Details [▶21]

Want to skip implementing K-means? K-means code [▶22]

18: K-means: Initialization

Easiest approach is to randomly assign each data item one of the K clusters (thus getting R), and computing the mean of each cluster based on this initial assignment.

```
mm = size(X,1);                  % number of data items
dd = size(X,2);                  % number of dimensions of data
cc = 1+rem(randperm(mm),K); % assign randomly to clusters
mu = zeros(K,dd);                % initialize means of clusters
for kk = 1:K
  mu(kk,:) = mean(X(cc==kk,:),1);
end
```

Why use `1+rem(randperm(mm),K)`? [▶23]

19: K-means: Updates

Each iteration first runs through the data items, finding the cluster whose mean is the closest and assigning to it, then runs through the clusters, computing a new mean based on the data items assigned to it.

```
for ii = 1:mm % iterate over data items
  dist = zeros(K,1);
  for kk = 1:K
    % squared distance
    dist(kk) = sum((X(ii,:)-mu(kk,:)).^2);
  end
  % find minimum distance cluster and assign item to it
  [dist kk] = min(dist);
  cc(ii) = kk;
end
for kk = 1:K % iterate over clusters to update means
  mu(kk,:) = mean(X(cc==kk,:),1);
end
```

20: K-means: Convergence Test

K-means converges once the assignment of data items to clusters in the iteration above does not change the cluster assignments.

At the start of each iteration, store a copy of the previous assignment of data items to clusters:

```
oldcc = cc;
```

At the end of each iteration compare the old copy to the new assignments:

```
if all(oldcc==cc)
  converged = 1;
end
```

21: K-means: Handling Empty Clusters

Once in a while a cluster becomes emptied because all the data items in it are assigned to other clusters. Easiest approach to dealing with this is to “steal” a data item at random and force it to be assigned to this cluster.

```
% make sure each cluster gets at least one data item
% using randperm guarantees at least one item per cluster
ii = randperm(mm);
for kk = 1:K
  if all(cc~=kk) % is empty
    cc(ii(kk)) = kk; % assign random item to cluster
  end
end
```

22: K-means: Code

```
function [cc,mu] = kmeans(X,K)
```

Initialization code [▶18]

```
converged = 0;
while (~converged)
  oldcc = cc; % copy of assignments
```

Update R code [▶19]

Handle Empty Clusters code [▶21]

Update μ code [▶19]

```
  if all(oldcc==cc)
    converged = 1;
  end
end
```

23: K-means: Why use `randperm`?

```
cc = 1+rem(randperm(mm),K)
```

`randperm(m)` returns a random ordering of the numbers $1, \ldots, m$. `1+rem(...,K)` then goes through the data items in the random ordering, assigning each to cluster $1, 2, \ldots, K-1, K, 1, 2, \ldots$ and so on. This process is often seen in classrooms when tutors want to break the class into equally sized but random groups: each student has to shout out $1, 2, \ldots$ in turn and is assigned to the group s/he shouted out.

We used this, rather than say randomly assigning each data item to one of the K clusters at random, because we with this each cluster will get an (almost) equal number of clusters, and we are guaranteed (so long as $K \leq m$) that none of the clusters are empty.

24: Spectral Clustering

You will need to have implemented K-means [▶16].

- We will implement the Ng et al spectral clustering method, and try it on a number of self-created datasets [▶7].
- From the lecture, the steps are:
  - Compute the W and D matrices [▶25];
  - Find the eigenvectors corresponding to the K largest eigenvalues of $D^{-\frac{1}{2}}WD^{-\frac{1}{2}}$ [▶26];
  - Form matrix Y whose columns are the eigenvectors [▶26].
  - Normalize the rows: $\tilde{Y}_{ij} = Y_{ij}/(\sum_j Y_{ij}^2)^{\frac{1}{2}}$ [▶26].
  - Perform K-means on the rows of $\tilde{Y}$ [▶26].
  - Assign $\mathbf{x}_i$ to cluster c if the i'th row of $\tilde{Y}$ is assigned to cluster c.
- Write a function implementing the above, with inputs and outputs:
  `function [cc,mu,Y,W] = speclust(X,K,sigma);`

25: Spectral Clustering: Step 1

- For simplicity we will use the RBF kernel as a similarity measure:

```
[mm dd] = size(X);
% compute distances
nx = sum(X.^2,2);
D = nx(:,ones(1,mm)) + nx(:,ones(1,mm))' - 2*X*X';
% similarity (kernel) matrix
W = exp(-D/(2*sigma.^2));
W = W - diag(diag(W)); % ignore self-similarity
```

  How did we compute the distances? [▶27]

- Compute D and normalize W by it:

```
D = diag(sum(W,2).^-.5);
N = D*W*D;
```

26: Spectral Clustering: Steps 2,3,4

- Solve the eigensystem and find K largest eigenvectors:

```
[Y Lambda] = eig(N);
[Lambda ii] = sort(diag(Lambda),1,'descend');
Y = Y(:,ii(1:K));
```

- Form the matrix of eigenvectors and normalize again:

```
ss = sum(Y.^2,2).^-.5;
Y = Y.*ss(:,ones(1,K));
```

- Finally perform K-means:

```
[cc mu] = kmeans(Y,K);
```

27: Spectral Clustering: Computing Distances

```
nx = sum(X.^2,2);
D = nx(:,ones(1,mm)) + nx(:,ones(1,mm))' - 2*X*X';
```

- The squared distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ can be written as:

$$\|\mathbf{x}_i - \mathbf{x}_j\|^2 = \|\mathbf{x}_i\|^2 + \|\mathbf{x}_j\|^2 - 2\mathbf{x}_i^\top \mathbf{x}_j$$

`nx` computes the squared lengths $\|\mathbf{x}_i\|^2$ and gives us the first two terms (notice the transpose on the second term). The third term is computed in the i,j entry of `2*X*X'`.

28: Linkage Algorithms

Linkage algorithms are hierarchical clustering algorithms that construct a hierarchy of clusters in a bottom-up fashion.

They start with each data item in its own cluster, and iteratively picks two (most similar) clusters to merge. The algorithm has a few important components:

- Compute distances between pairs of data items;
- Initialize all data items in their own clusters;
- Iteratively: find the most similar pair of clusters, and;
- Merge them.

The linkage algorithms are more involved to implement. If you feel up to the challenge, more details [▶29] and pseudocode [▶30] are provided to help you. If not, `avglink.m` in the provided software package implements average linkage. You can explore using different distance metrics or different distance combination techniques instead, or applying average linkage to datasets.

29: Linkage Algorithms: Details

The main idea in writing efficient code for linkage algorithms is to maintain a list of the current clusters, and a *priority queue of pairs of clusters* prioritized by similarity.

For efficiency, the distances between pairs of data items can be computed in an initial phase and stored in a matrix to be accessed later.

Since each iteration merges two clusters, and we start with m data items, there will be a total of $2m-1$ clusters throughout the computation history (there are $m-1$ total iterations, each iteration creates a new cluster by merging two). We can label the list of clusters as $1, \ldots, m, \ldots, 2m-1$, with $1, \ldots, m$ corresponding to the m data items and $m+1, \ldots, 2m-1$ in order of creation.

Useful outputs of the algorithm are:

- For each cluster $i = 1, \ldots, 2m-2$ except the root the index p_i of its parent;
- For each cluster $i = m+1, \ldots, 2m-1$, the indices $\{c_{i1}, c_{i2}\}$ of its children;
- For each cluster $i = m+1, \ldots, 2m-1$, the similarity s_i between its children.

30: Linkage Algorithms: Pseudocode

- Initialize list C of clusters with each data item in its own;
- Initialize queue Q with all $m(m-1)/2$ pairs of clusters, prioritized by similarity;
- While Q not empty:
  - Pick most similar pair of clusters off Q;
  - If both clusters are in C this pair will be merged:
    - Remove this pair from Q;
    - Remove both clusters from C, and add the newly merged cluster to C;
    - Add to Q pairs of clusters consisting of this merged cluster and another cluster from C;
  - Otherwise at least one cluster in pair is not in C:
    - Remove this pair from Q;

31: Local Optimal in K-means

You will need to have implemented K-means [▶16].

- Once you have K-means working, try it multiple times on the same simple dataset.
- Does it get to the same solution every time?
- Try increasing K to be more than the number of clusters you think it should use. Does it get to the same solution every time? Does it "break" some of the clusters into smaller clusters differently each time?
- You can also try getting local optima behaviour by creating a harder dataset, e.g. the strange stringy ones used for spectral clustering.

32: Varying K in K-means

You will need to have implemented K-means [▶16].

- Implement a function that computes the objective function J in K-means.
- Try K-means on the same dataset with varying numbers of clusters K.
- Plot the J values obtained at convergence as a function of K.
- Does it go down towards 0 as K becomes larger?
- Does it wriggle up and down a little? This is due to local optima in K-means. Try running K-means multiple times for each K, and using the best run (in terms of the lowest J value). Does it wriggle now? It should wriggle less.

33: K-means on Cancer of Golub

You will need to have prepared the Cancer of Golub data [▶9] and implemented K-means [▶16].

- The major subtypes of leukemia in the dataset are ALL and AML. We will first try to use K-means to discover this distinction. We extract this distinction from `classes` in `golub.mat`:

```
cc = cellfun(@(s)strcmp('ALL',s(1:3)),classes(:,2));
traincc = cc(1:numtrain);
testcc  = cc(1:numtest);
```

- Let us first try K-means with $K = 2$:

```
cc = kmeans(trainnormd,2);
```

  - Compare the output `cc` to the desired outcome `traincc`. Do they look similar?
  - Try K-means a few more times. The initialization is random so it should produce different (but undesired) answers each time.
  - Perhaps $K = 2$ is too severe, try $K = 3, 4, 5$ and see if you can get combinations of clusters to correspond to the ALL/AML distinction.
  - What is going wrong? [▶34]

34: K-means on Cancer of Golub

- What is going wrong? Recall that we have 7129 genes. Most of these genes are housekeeping genes that are functioning normally even in the leukemia samples. Only a relatively small number of genes will be affected by leukemia. The issue: the distances between patient samples are getting screwed up by the many irrelevant genes.
- One simple approach is to clean up the dataset by picking out the relevant genes, and clustering just based on them. This is called *feature selection*.
- But it seems unlikely that we will be able to pick out just the right genes to allow us to cluster properly, since there are many distinctions (further subtypes, other associated diseases, different patient characteristics like gender, weight, different laboratory conditions) that can potentially affect gene expression level. What to do? [▶35]

35: K-means on Cancer of Golub

- Since our aim is to hone your data analysis skills, let us cheat a bit so that we can proceed further with clustering. Included in `golub.mat` is a variable `inforder` which orders the genes by informativeness to the ALL/AML distinction.
  - Are you interested in how to estimate informativeness? [▶36]
- Try using the first 100 informative genes to cluster:

  ```
  X = trainnormd(:,inforder(1:100));
  cc = kmeans(X,2);
  ```

  Does it work now? You can visualize the normalized gene expression profiles with `imagesc(X,[-3 3])`.
- Try with varying numbers of genes up to the maximum (7129). When does this break down?

36: K-means on Cancer of Golub

- We can use logistic regression or some other classification techniques to measure informativeness of individual genes to the ALL/AML distinction.
- We simply learn a classifier to predict the class distinctions from the expression levels of just that individual gene.
- We do this for each gene, and measure the informativeness of a gene in terms of the accuracy of the classifier. This gives the `inforder` ordering.
- A simpler and more naïve method is to measure informativeness in terms of how "well separated" the expression levels of that gene are among patients with ALL and patients with AML. If μ_1, μ_2 are the mean expression levels of patients with ALL and AML subtypes, and σ_1, σ_2 are similarly for standard deviations, a measure of informativeness is:

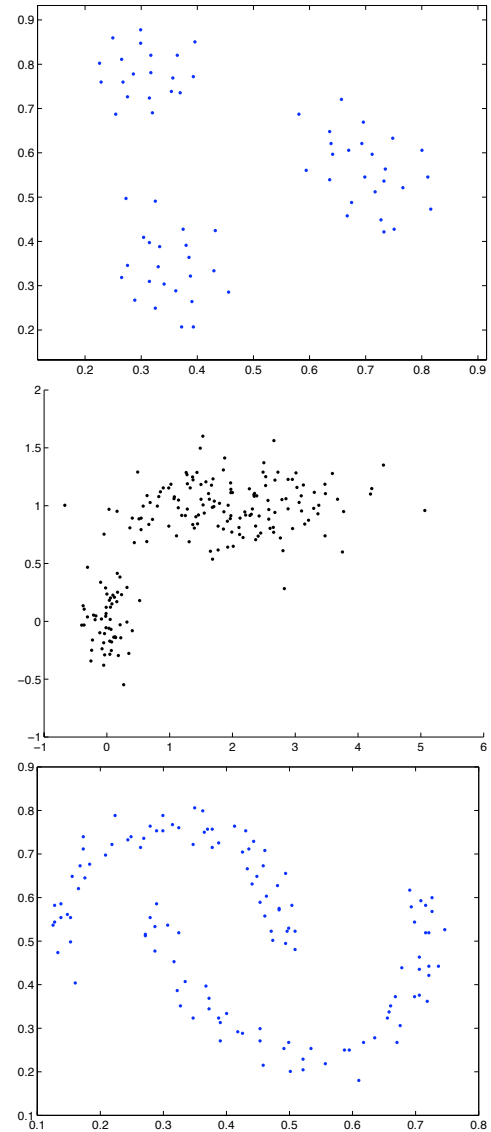
$$\frac{|\mu_1 - \mu_2|}{\sigma_1 + \sigma_2}$$

  You can try this ordering instead of `inforder`.
- An interesting thing to try is to find genes that are most informative of other distinctions (e.g. male vs female or the B-type vs T-type distinction among ALL types), and to see if these produce nice clusters as well.

37: Other Linkage Algorithms

- It is very simple to edit `avglink.m` to produce min and max linkage. Copy `avglink.m` and rename the copies `minlink.m` and `maxlink.m`, and make the desired changes.
- Try the various linkage algorithms on various datasets (some examples on right) created using `getpoints.m` [▶7].
- You can analyze the results of the linkage algorithms using `plotlinkage.m`. The command `plotlinkage(X,C,K)` shows the flat clusterings obtained with $2, \ldots, K$ clusters if we cut the tree at different levels. `X` is dataset, `C` is output of the linkage algorithms, and `K` is maximum number of clusters.
- Do you see qualitative differences between the different linkage algorithms?

More varieties of linkage algorithms [▶38].



38: Other Linkage Algorithms

Other avenues of extension to linkage algorithms are to change the distance metrics on data items and on clusters.

- Mahanalobis distance is Euclidean distance where different directions are weighted differently:

$$d(x,y) = (x-y)^\top \Sigma^{-1}(x-y)$$

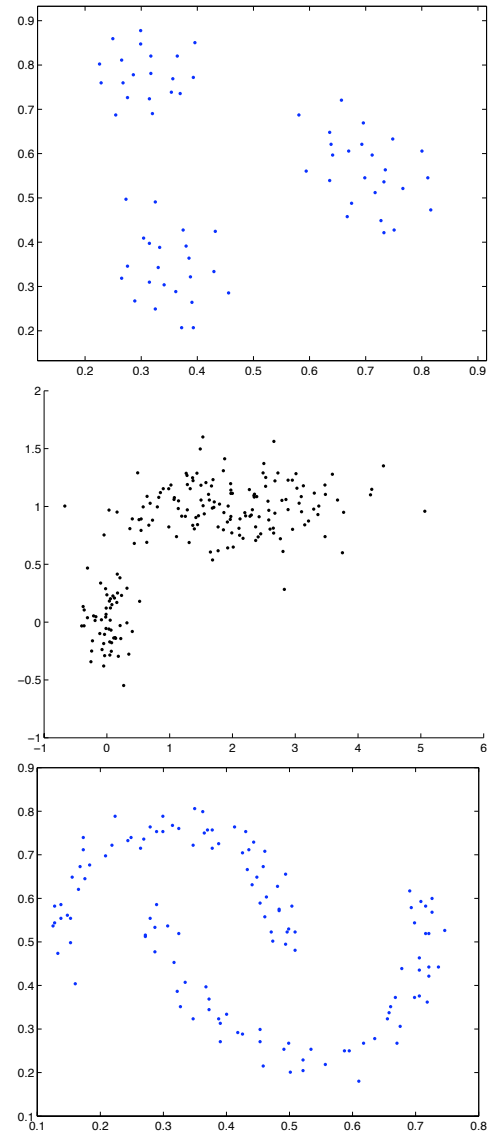
  where Σ is a positive definite matrix.

- Distances induced by L_p norms

$$d(x,y) = \left(\sum_j |x_j - y_j|^p\right)^{1/p}$$

- Distances between clusters induced by probabilistic models

$$d(C,D) = -\log \frac{P(C \cup D)}{P(C)P(D)}$$



39: Trying out Spectral Clustering

You will need to have implemented K-means [▶16] and spectral clustering [▶24].

- Try out spectral clustering on a number of datasets you create yourself. Some examples are shown on right.

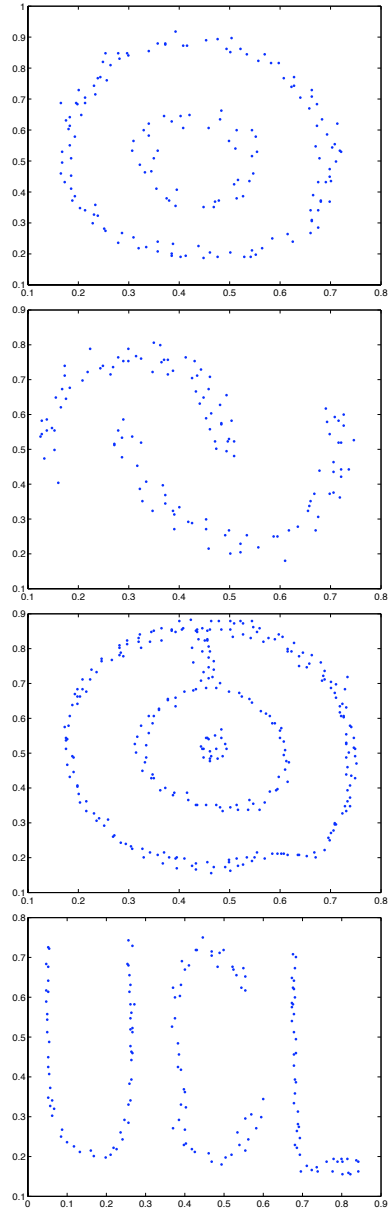
```
X = getpoints;
[cc,mu,Y,W] = speclust(X,K,sigma);
```

- You can plot results of clustering using `plotclusters`:

```
plotcluster(X,cc)
```

  K is the number of clusters, while σ is the length scale: it describes at what scale are data items considered similar.

- Try various values of K and σ. Describe how the result of the clustering algorithm depends on both.
- Unfortunately it would be way too hard to implement a spectral clustering method that can work on even small images in 3 hours! Check out [▶`http://www.cis.upenn.edu/~jshi/software`].



40: Mixture of Gaussians Demo

- You can try out mixtures of Gaussians code I included as file `mog.m`.
- There is a demo `demo_mog.m` which lets you create a dataset then runs a mixture of Gaussians on it.
- You can explore issues of overfitting in mixtures of Gaussians.
- Create a dataset, and run the demo with various values of K.
- Looks at the results of the clustering, and see if they look reasonable.
- As you increase K do you see if the log likelihood increases, even if the clustering quality (subjectively) decreases?