COMP 424 - Artificial Intelligence Lecture 16: Learning with Missing Values

Instructor: Jackie CK Cheung (jcheung@cs.mcgill.ca)

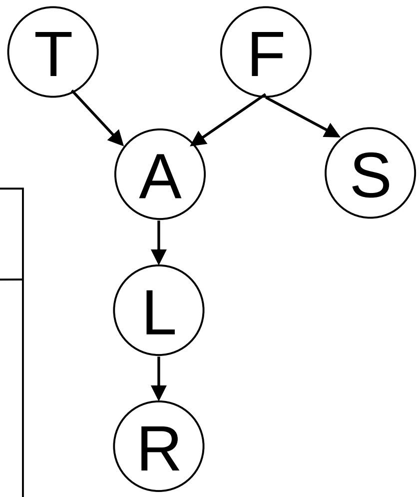
Readings: R&N Ch 20.3

Today's overview

- Topic: Parameter learning with missing values
- Expectation maximization (EM)
- K-means clustering

Learning in Bayesian networks

- Given data in the form of instances:

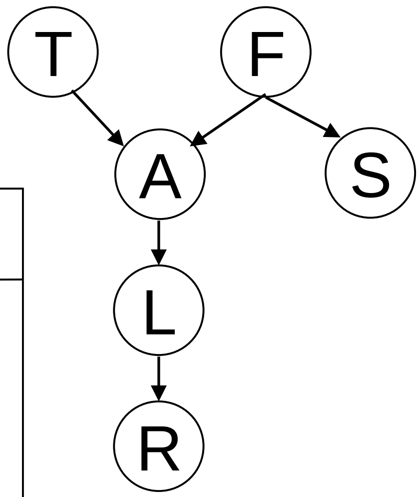


Tampering	Fire	Smoke	Alarm	Leaving	Report
No	No	No	No	No	No
No	Yes	Yes	Yes	Yes	No
…	…		…	…	…

- Goal: Find parameters of the Bayes net.
- We discussed how to do this using maximum likelihood.

Learning in Bayesian networks

- **Plot twist**: Suppose some values are missing!



Tampering	Fire	Smoke	Alarm	Leaving	Report
?	No	No	No	No	No
No	Yes	Yes	Yes	?	No
…	…		…	…	…

- Can we still use MLE?
  - How do we deal with the missing data?

Why do we get incomplete data?

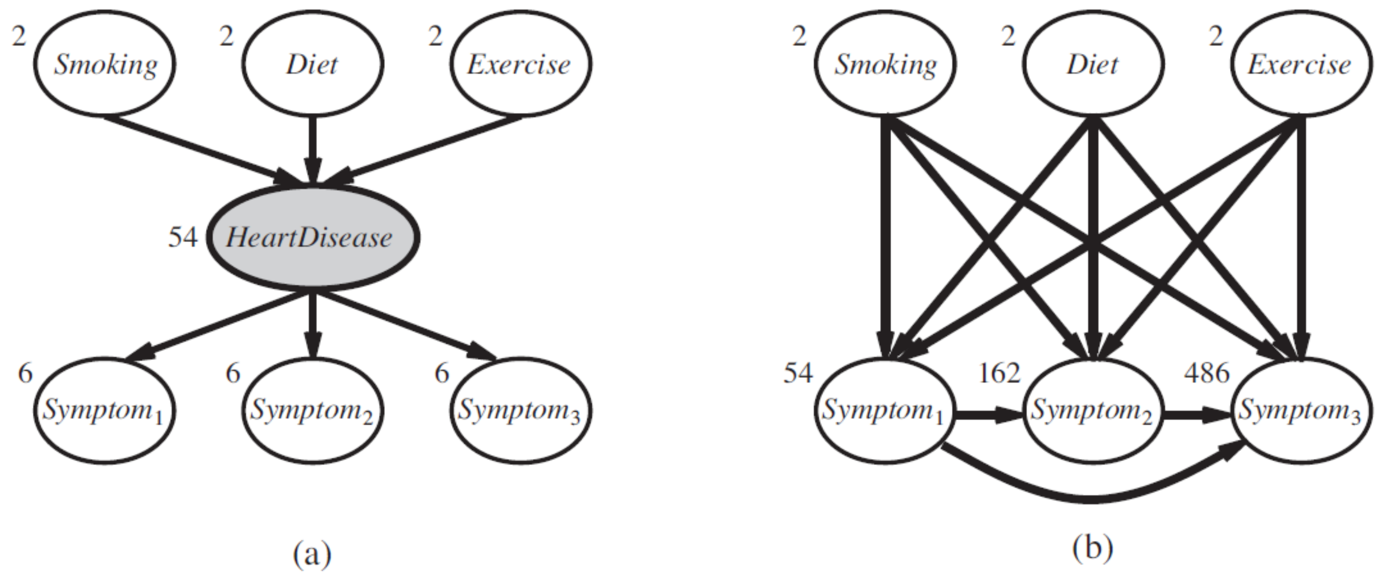
- Some variables may not be assigned values in *some* instances.
    - e.g., not all patients undergo all medical tests.

- Some variables may not be observed in *any* of the data items.
    - e.g., viewer preferences for a show may depend on their metabolic cycle (what time they are awake) - which is not usually measured
    - These are called **latent** (or **hidden**) **variables**

Why model latent variables?

- You can imagine designing a Bayesian network that ignores latent variables
- e.g., heart disease domain
  - Factors: Smoking, diet, exercise
  - Symptoms: s1, s2, s3
  - Latent variable: whether a person has heart disease

Latent variables add perspicuity



Shaded means missing
R&N Fig 20.10

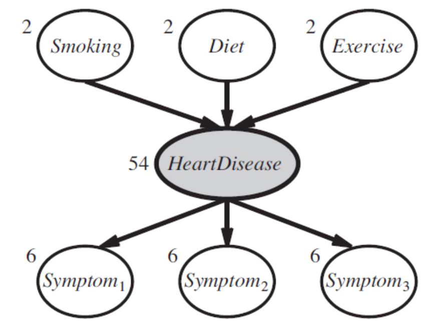
If each variable has 3 possible values:

78 parameters

708 parameters!

Learning with missing data

- Our problem is to estimate the parameters of the network
- If we had the parameters, we could get P(HeartDisease|OtherVariables), then we would have complete data for doing supervised learning
- A chicken and egg conundrum



Idea:

- Let's pretend we know the parameters of the model
- Then we can do inference to figure out the value of HeartDisease
- Then we can re-estimate the parameters!
- *This seems like magic! But it works (kind of)!*

Expectation Maximization (EM)

- General purpose method for learning from incomplete data (not only Bayes nets), **whenever an underlying distribution is assumed**.
- Main idea: Alternate between two steps
  1. (**E-step**): For all the instances of missing data, we will "fantasize" how the data should look based on the current parameter setting.
     - This means we compute **expected sufficient statistics**.
  2. (**M-step**): Then **maximize parameter setting**, based on these statistics.

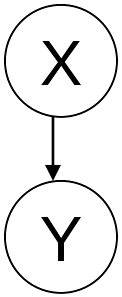
Outline of EM

- **Initialization**:
  - Start with some initial parameter setting (e.g. *P(T), P(F), P(A|F,T)*, etc.).
  - These can be estimated from all complete data instances.
- **Repeat**:
  1. <u>Expectation (E-step)</u>: Complete the data by assigning "values" to the missing items based on current parameter setting.
  2. <u>Maximization (M-step)</u>: Compute the maximum likelihood parameter setting based on the completed data. This is what we did last class.
- **Convergence**:
  - Nothing changes in E-step or M-step between 2 consecutive rounds.

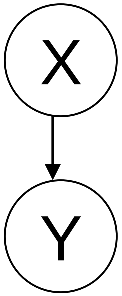
Example

- Consider a simple network $X \rightarrow Y$, and suppose we want to learn its parameters from samples $\langle x_1, y_1 \rangle, \ldots, \langle x_m, y_m \rangle$.

- Suppose that x_1 is missing and $y_1=1$. What can we do?



EM in our example



- To start, guess the parameters of the network θ (using the known data):

  e.g. $\theta_X = N_{x=1}(2{:}m) / (m-1)$

  $\theta_{Y|x=0} = N_{Y=1,X=0}(2{:}m) / N_{X=0}(2{:}m)$

  $\theta_{Y|x=1} = N_{Y=1,X=1}(2{:}m) / N_{X=1}(2{:}m)$

- **E-step**: Using initial θ, compute: $P(x_1=0 \mid y_1), P(x_1=1 \mid y_1)$

  (Note that this step requires *exact inference* - so not cheap!)

  Complete dataset with most likely value of x_1.

  Call new dataset D*.

- **M-step**: Compute new parameter vector θ, which maximizes the likelihood given the completed data: $L(\theta|D^*) = P(D^* \mid \theta)$

  e.g. $\theta_X = N_{x=1}/m$

  $\theta_{Y|x=0} = N_{Y=1,X=0} / N_{X=0}$

  $\theta_{Y|x=1} = N_{Y=1,X=1} / N_{X=1}$

- **Repeat E-step and M-step until the parameter vector converges.**

Two version of the algorithm

- **Hard EM**: for each missing data point, assign the value that is most likely.

  (This is the version we just saw.)

- **Soft EM**: for each missing data point, put a weight on each value, equal to its probability, and use the weights as counts.

  (This is the most common version.)

Then these numbers are used as real counts, to provide a maximum likelihood estimate for θ.

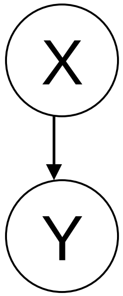
Soft EM in our example

- To start, guess the parameters of the network θ (using the known data):

  E.g. $\theta_X = N_{x=1}(2{:}m) / (m-1)$

  $\theta_{Y|x=0} = N_{Y=1,X=0}(2{:}m) / N_{X=0}(2{:}m)$

  $\theta_{Y|x=1} = N_{Y=1,X=1}(2{:}m) / N_{X=1}(2{:}m)$



- **<u>E-step</u>**: Using initial θ, compute: $w_0 = P(x_1=0 \mid y_1)$ $w_1 = P(x_1=1 \mid y_1)$

  Now hypothesize two datasets: $D_0 = <w_0, y_1>, <x_2, y_2>, \ldots, <x_m, y_m>$

  $D_1 = <w_1, y_1>, <x_2, y_2>, \ldots, <x_m, y_m>$

- **<u>M-step</u>**: Compute new parameter vector θ, which maximizes the <u>expected likelihood</u> given the completed data: $L(\theta|D^*) = w_0 P(D_0 \mid \theta) + w_1 P(D_1 \mid \theta)$

  E.g. $\theta_X = (N_{X=1}(2{:}m) + w_1) / m$

  $\theta_{Y|x=0} = (N_{Y=1,X=0}(2{:}m) + w_0) / (N_{X=0}(2{:}m) + w_0)$

  $\theta_{Y|x=1} = (N_{Y=1,X=1}(2{:}m) + w_1) / (N_{X=1}(2{:}m) + w_1)$

Repeat E and M steps!

Comparison of hard EM and soft EM

- Soft EM does not commit to specific value for the missing item.
  - Instead, it considers all possible values, with some probability.
  - This is a pleasing property, given the uncertainty in the value.
- Complexity:
  - Hard EM requires computing most probable values.
  - Soft EM requires computing conditional probabilities for completing the missing values.
  - Same complexity: both require full probabilistic inference - which can be expensive!

Properties of EM

- Likelihood function is guaranteed to improve (or stay the same) with each iteration.
  - Algorithm can be stopped when no more improvement is achieved between iterations.

- EM is guaranteed to converge to a *local* optimum of the likelihood function.
  - Starting with different values of initial parameters is necessary
  - Or find an informed way to initialize the parameters (e.g., with a small labelled dataset)

- EM is a widely used algorithm in practice!

A harder example

Suppose we have the simple Bayes net $A \to B \to C$, where each node is associated with a Bernoulli random variable. Further suppose we have the following sample data:

(i) A=1,B=?,C=1

(ii) A=0,B=1,C=0

(iii) A=1,B=0,C=0

(iv) A=1,B=1,C=0

(v) A=1,B=1,C=0

(vi) A=0,B=0,C=?

A harder example

Suppose we have the simple Bayes net $A \to B \to C$, where each node is associated with a Bernoulli random variable. Further suppose we have the following sample data:

(i) A=1,B=?,C=1

(ii) A=0,B=1,C=0

(iii) A=1,B=0,C=0

(iv) A=1,B=1,C=0

(v) A=1,B=1,C=0

(vi) A=0,B=0,C=?

E-step

$$\begin{aligned}
w_{B_1=1} &= P(B_1 = 1 | A_1, C_1) \\
&= P(B = 1 | A = 1, C = 1) \\
&= \frac{P(A = 1, B = 1, C = 1)}{P(A = 1, C = 1)} \\
&= \frac{P(A = 1, B = 1, C = 1)}{\sum_{b \in 0,1} P(A = 1, C = 1, B = b)} \\
&= \frac{\theta_A \theta_{B|A=1} \theta_{C|B=1}}{\theta_A \theta_{B|A=1} \theta_{C|B=1} + \theta_A (1 - \theta_{B|A=1}) \theta_{C|B=0}} \\
&= \frac{(0.5)(0.5)(0.5)}{(0.5)(0.5)(0.5) + (0.5)(0.5)(0.5)} \\
&= 0.5
\end{aligned}$$

$$\begin{aligned}
w_{B_1=0} &= P(B_1 = 0 | A_1, C_1) \\
&= P(B = 0 | A = 1, C = 1) \\
&= (1 - P(B = 1 | A = 1, C = 1)) \\
&= (1 - w_{B_1=1}) \\
&= 0.5
\end{aligned}$$

$$\begin{aligned}
w_{C_6=1} &= P(C_6 = 1 | A_6, B_6) \\
&= P(C_6 = 1 | B_6) \qquad \text{by conditional independence} \\
&= P(C = 1 | B = 0) \\
&= 0.5
\end{aligned}$$

$$\begin{aligned}
w_{C_6=0} &= (1 - w_{C_6=1}) \qquad \text{same reasoning as } w_{B_1=0} = (1 - w_{B_1=1}) \\
&= 0.5
\end{aligned}$$

M-step

$$\theta_A^{ML} = \frac{N_{A=1}(1{:}6)}{6} = \frac{4}{6} \approx 0.667$$

$$\theta_{B|A=1}^{ML} = \frac{N_{B=1|A=1)}(2{:}6) + w_{B_1=1}}{4} = \frac{2+0.5}{4} = 0.625$$

$$\theta_{B|A=0}^{ML} = \frac{N_{B=1|A=1)}(2{:}6)}{2} = \frac{1}{2} = 0.5$$

$$\theta_{C|B=1}^{ML} = \frac{N_{C=1|B=1}(2{:}4) + w_{B_1=1}}{3 + w_{B_1=1}} = \frac{0.5}{3.5} \approx 0.143$$

$$\theta_{C|B=0}^{ML} = \frac{N_{C=1|B=0}(2{:}4) + w_{B_1=0} + w_{C_6=1}}{2 + w_{B_1=0}} = \frac{0.5+0.5}{2.5} = 0.4$$

And repeat…

A seemingly harder problem

- What if one of your variables is <u>always</u> missing?

Data = <x_1, ?>, <x_2, ?>, …, <x_m, ?>

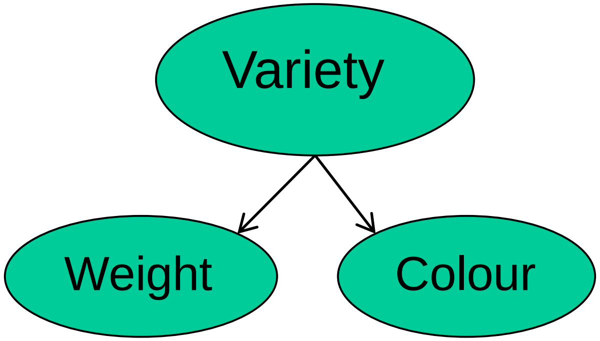
- Can you estimate a maximum-likelihood parameter for this case?

- We call this **learning from unlabeled data**, or **unsupervised learning**

An example

- A fruit merchant approaches you, with a set of apples to classify according to their variety.
  - Tells you there are five varieties of apples in the dataset.
  - Tells you the weight and colour of each apple in the dataset.

- Can you label each apple with the correct variety?
  - What would you need to know / assume?

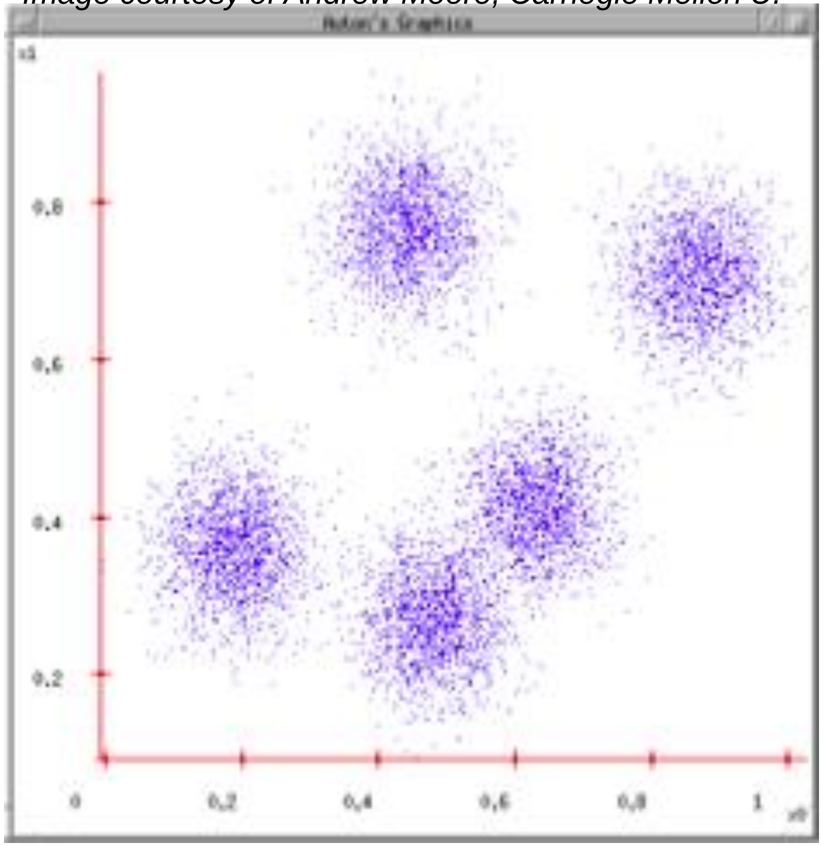


Plotting the data

What if you observe that the data looks like this?

(One axis is weight, the other is colour.)

Image courtesy of Andrew Moore, Carnegie Mellon U.



Reminder: Gaussian distribution

- a.k.a., normal distribution
- **Probability density function**:

$$f(x|\mu,\sigma^2) = N(\mu,\sigma^2) = \frac{1}{\sqrt{2\sigma^2\pi}}\exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

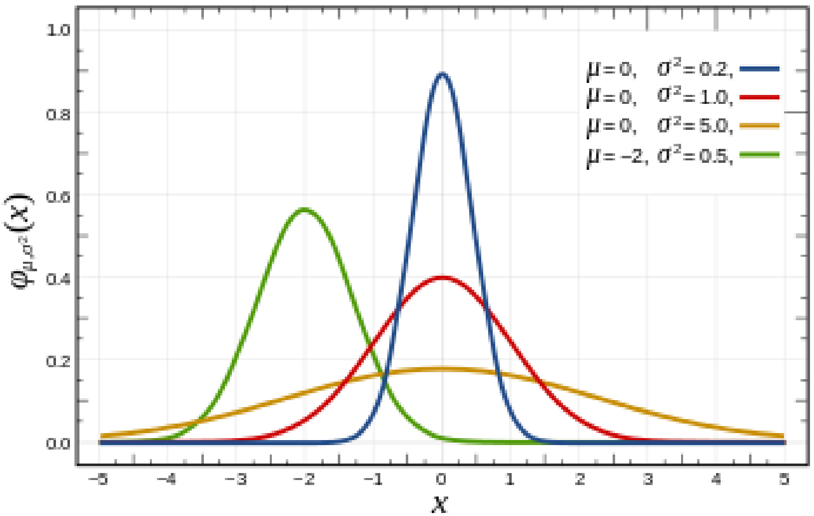


Image source: Wikipedia

  - Outcomes are continuous values (e.g., height, weight, size, …)
  - Parameterized by **mean** and **variance**

K-means clustering

- K-means clustering: cluster instances into K distinct classes.
- Need to know K in advance.
- Need to assume a parametric distribution for each class.

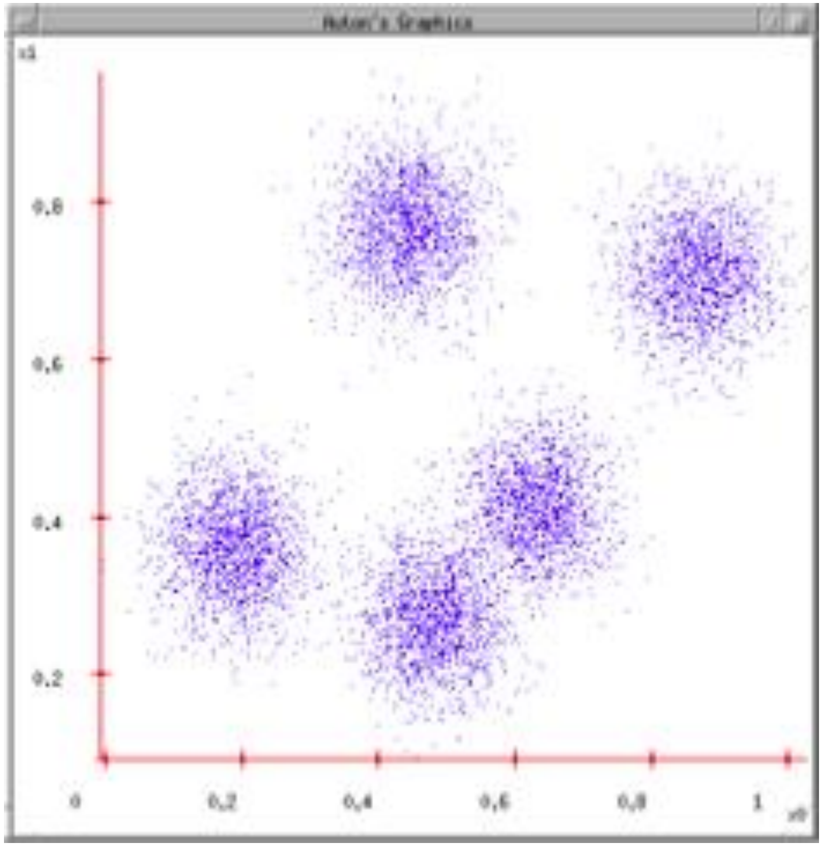
- *Back to our apples…*
  - You know there are 5 varieties.
  - Assume each variety generates apples according to a (variety-specific) 2-D Gaussian distribution, $N(\mu_i, \sigma_i^2)$
  - **If** you know μ_i, σ_i^2 for each class, it's easy to classify the apples!
  - **If** you know the class of each apple, it's easy to estimate μ_i, σ_i^2!

    What if we know neither?

K-means algorithm

This data could easily be modeled by Gaussians.

1. Ask user how many clusters.



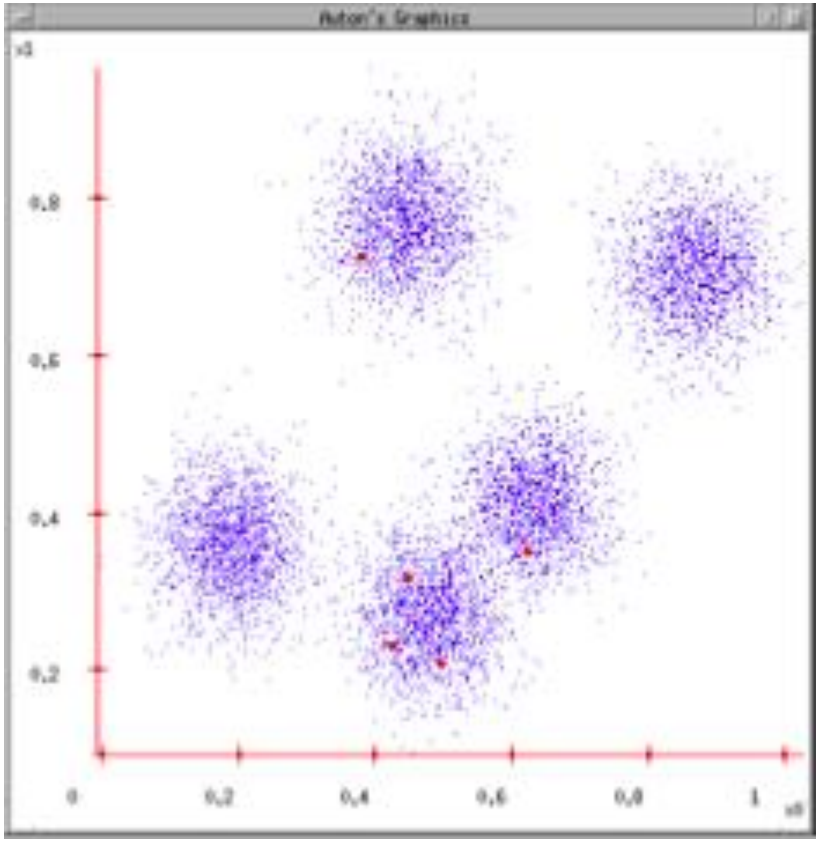
Images courtesy of Andrew Moore, Carnegie Mellon U.

K-means algorithm

This data could easily be modeled by Gaussians.

1. Ask user how many clusters.
2. Randomly guess k centers:

   $\{ \mu_1, \ldots, \mu_k \}$ (assume σ^2 is known).

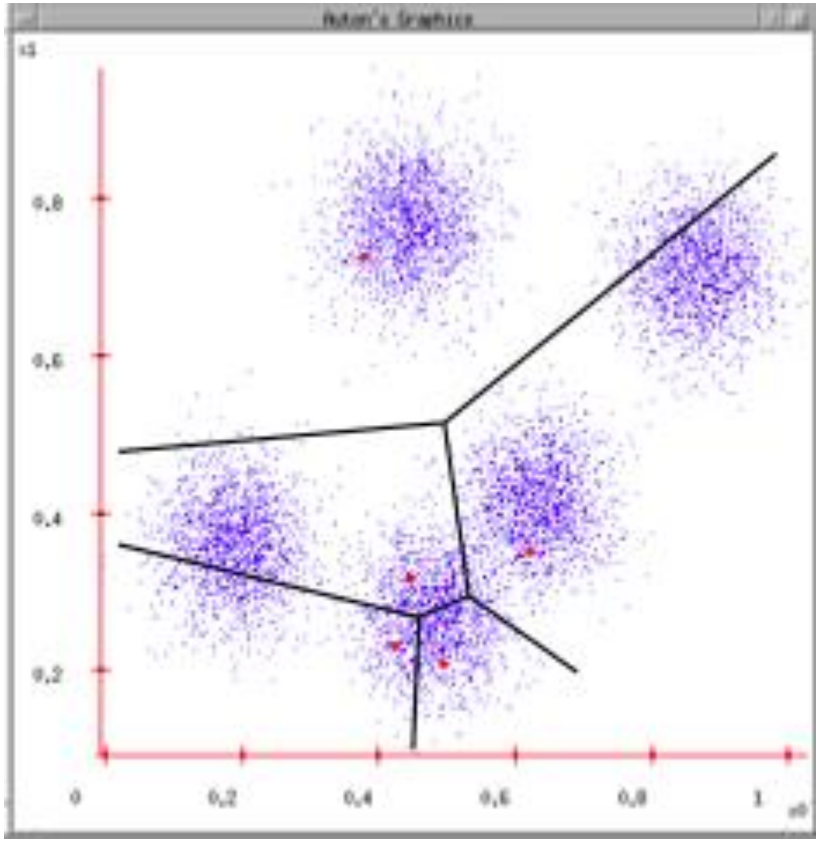


K-means algorithm

This data could easily be modeled by Gaussians.

1. Ask user how many clusters.
2. Randomly guess k centers:

   $\{ \mu_1, \ldots, \mu_k \}$ (assume σ^2 is known).
3. Assign each data point to closest center.

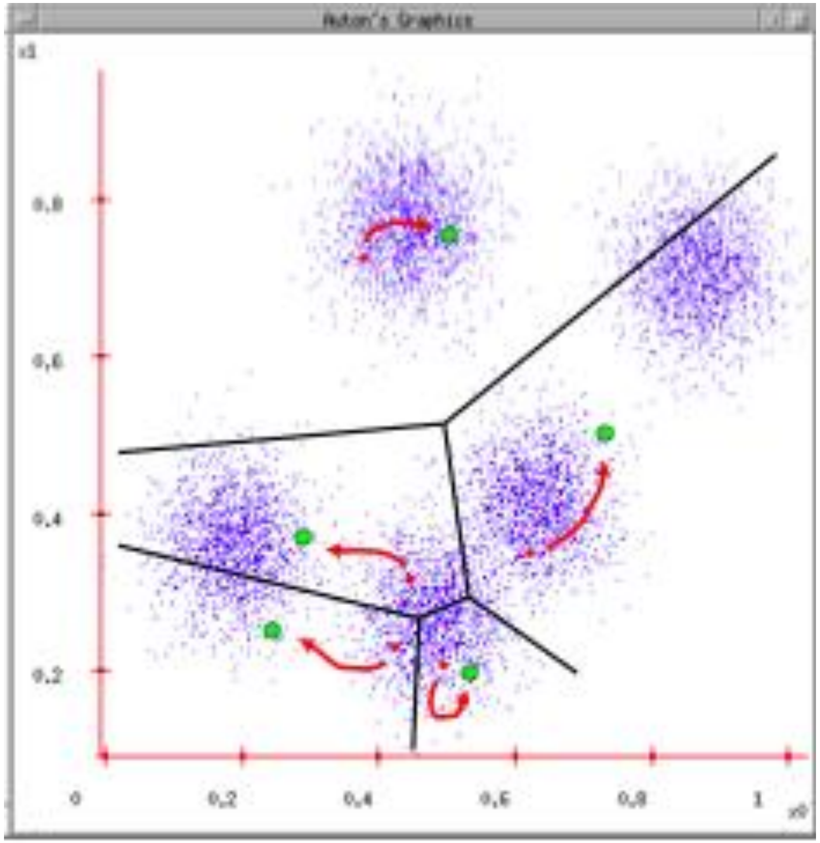


K-means algorithm

This data could easily be modeled by Gaussians.

1. Ask user how many clusters.
2. Randomly guess k centers:

   $\{ \mu_1, \ldots, \mu_k \}$ (assume σ^2 is known).
3. Assign each data point to the closest center.
4. Each center finds the centroid of the points it owns.

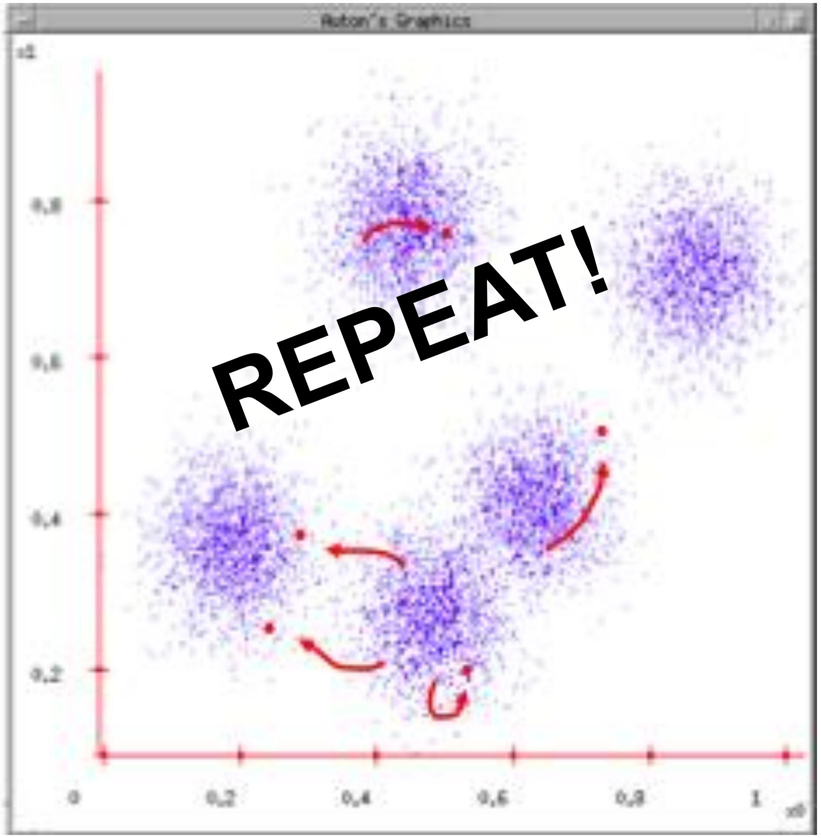


K-means algorithm

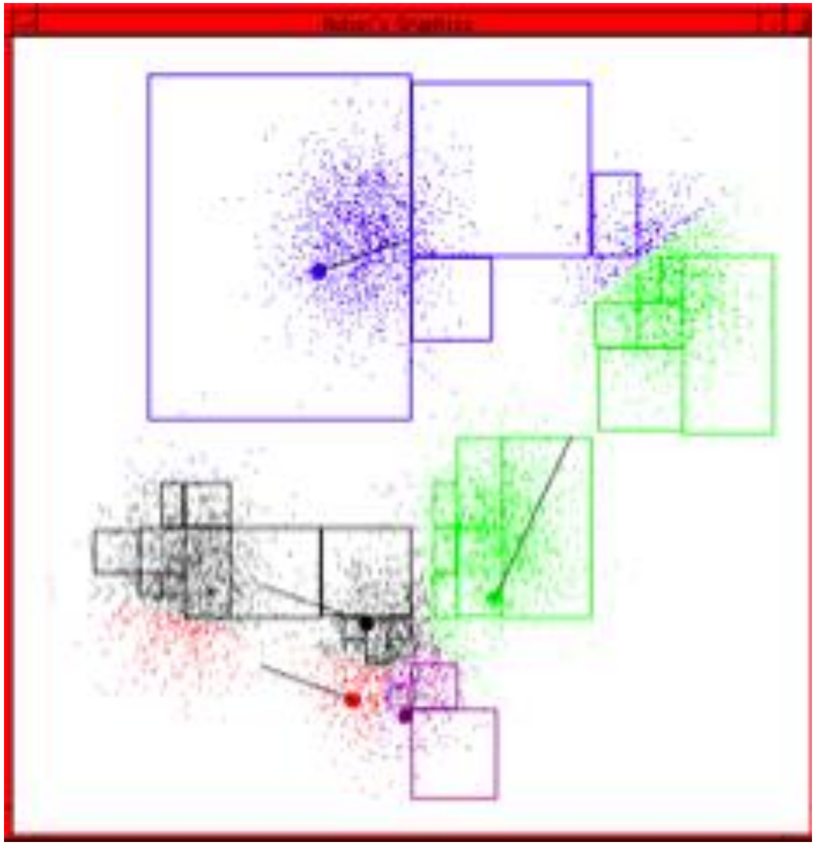
This data could easily be modeled by Gaussians.

1. Ask user how many clusters.
2. Randomly guess k centers:

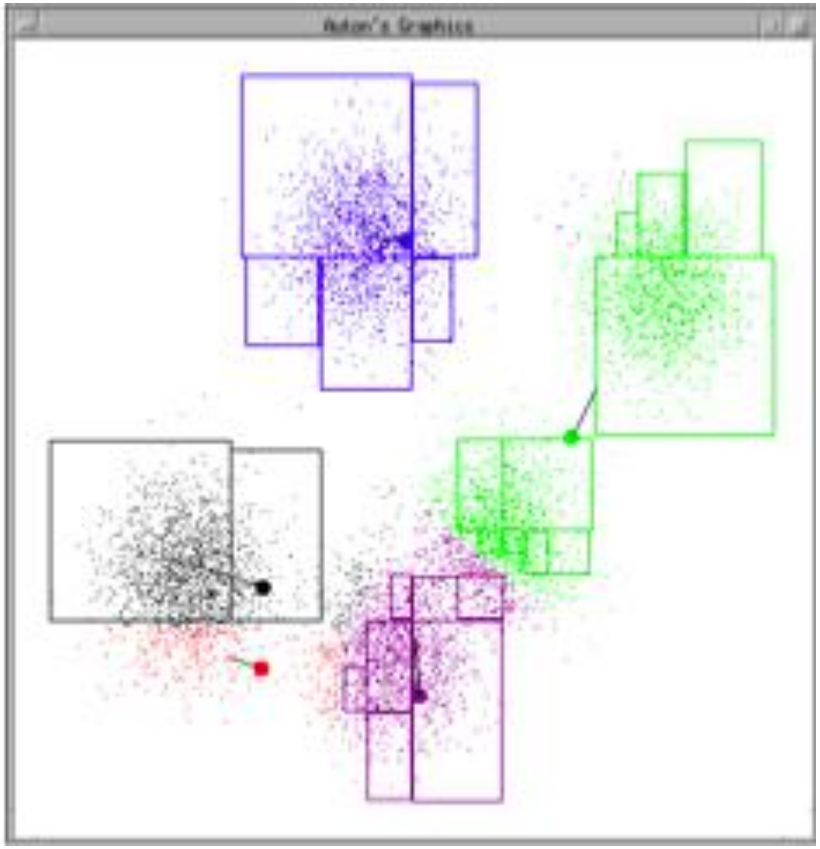
   $\{ \mu_1, \ldots, \mu_k \}$ (assume σ^2 is known).
3. Assign each data point to the closest center.
4. Each center finds the centroid of the points it owns.
5. Repeat steps 3-4



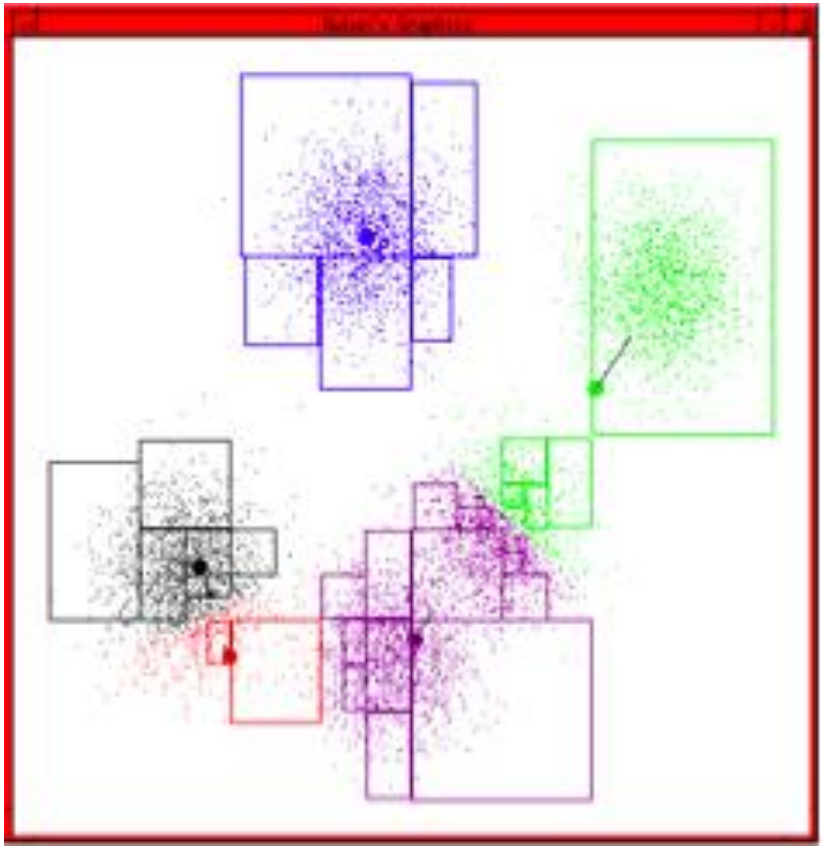
K-means algorithm starts



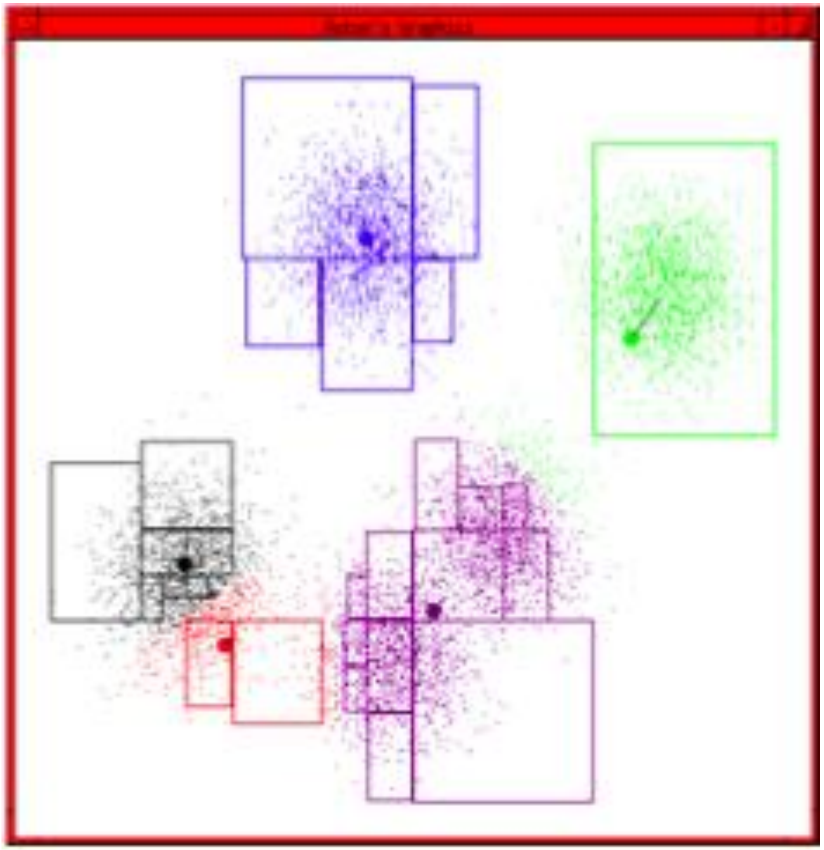
K-means algorithm continues (2)



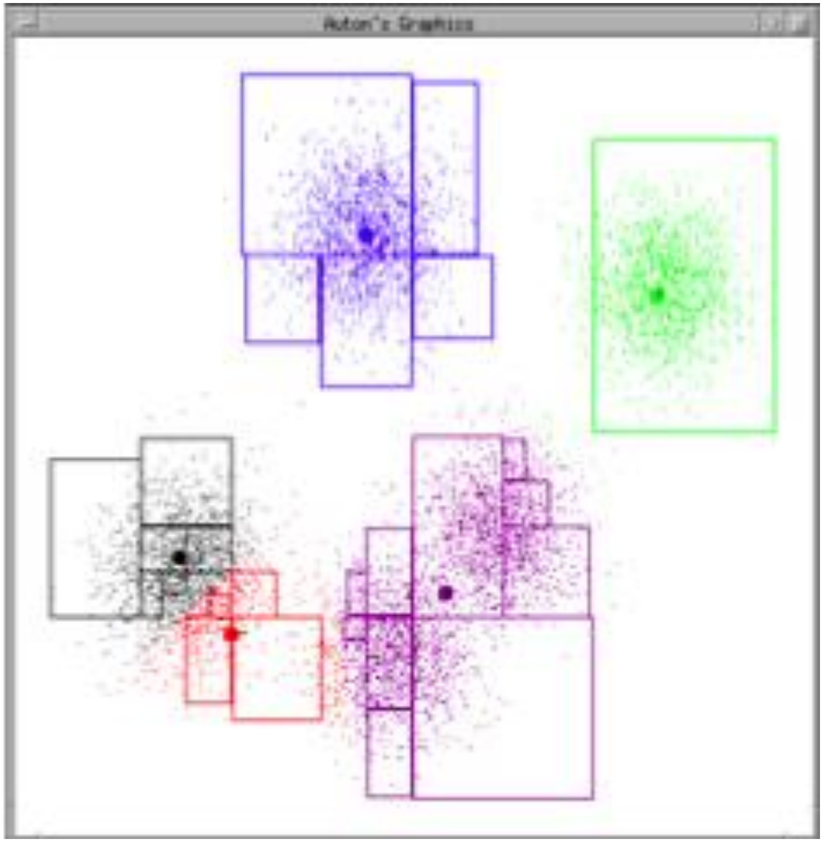
K-means algorithm continues (3)



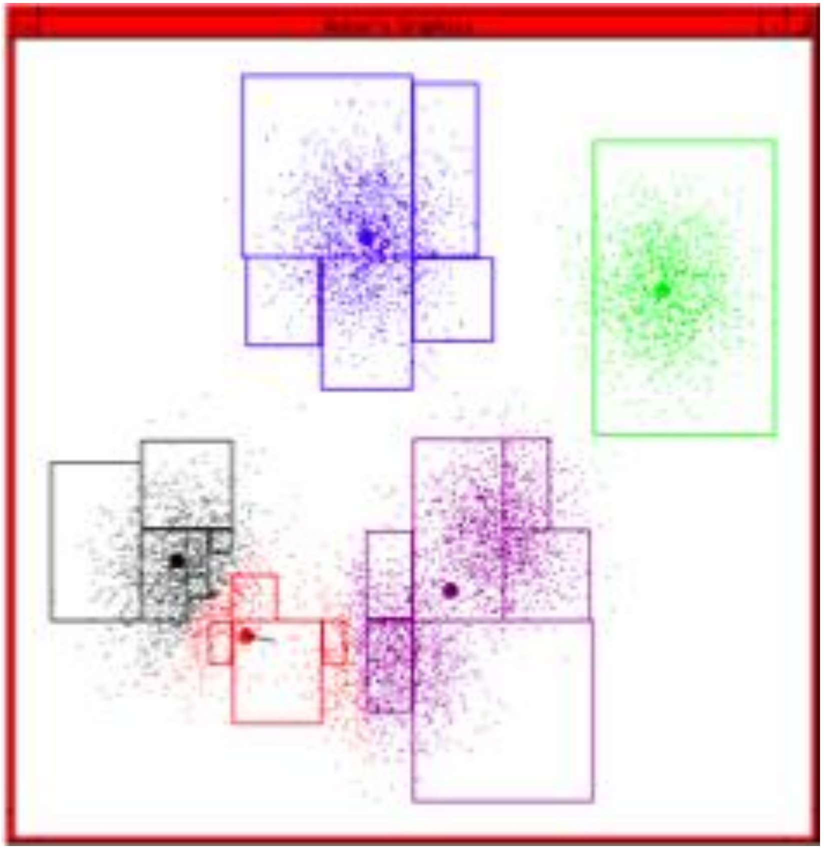
K-means algorithm continues (4)



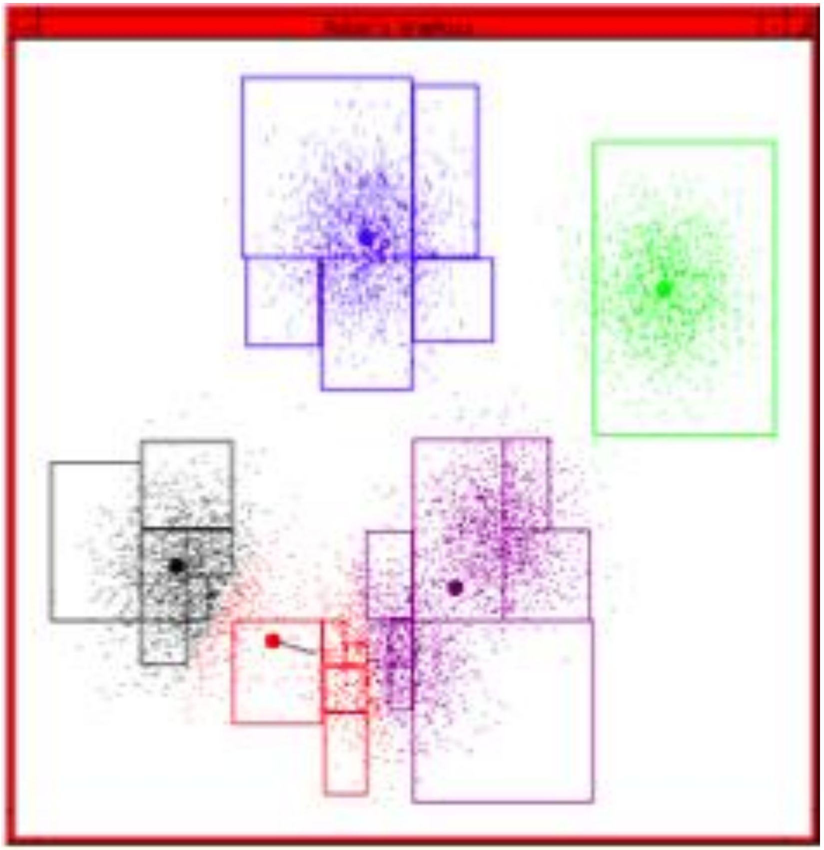
K-means algorithm continues (5)



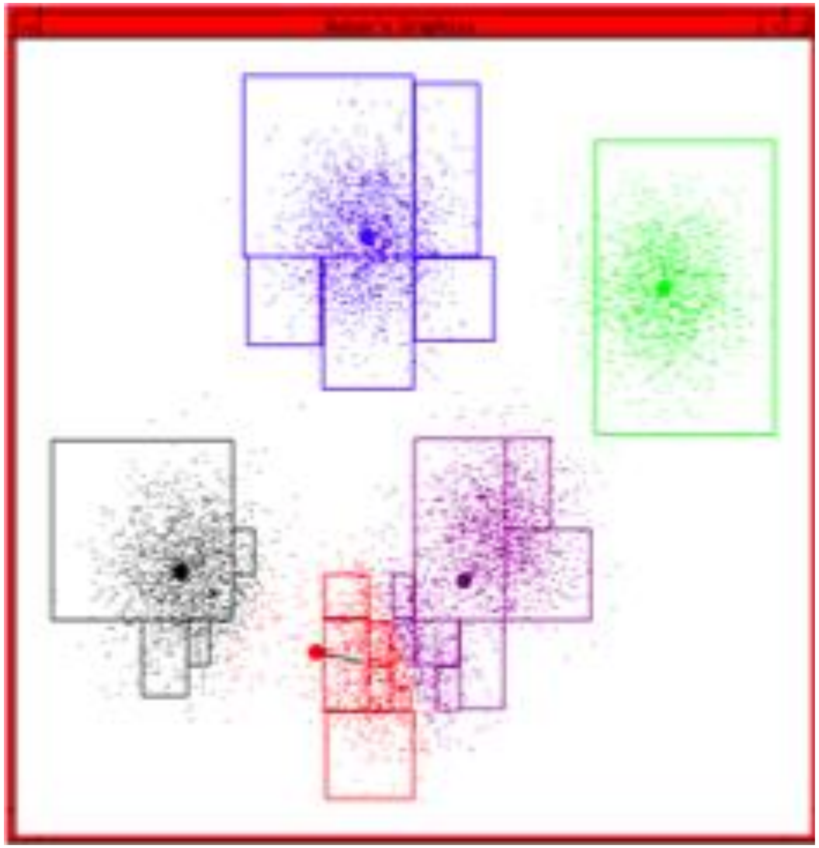
K-means algorithm continues (6)



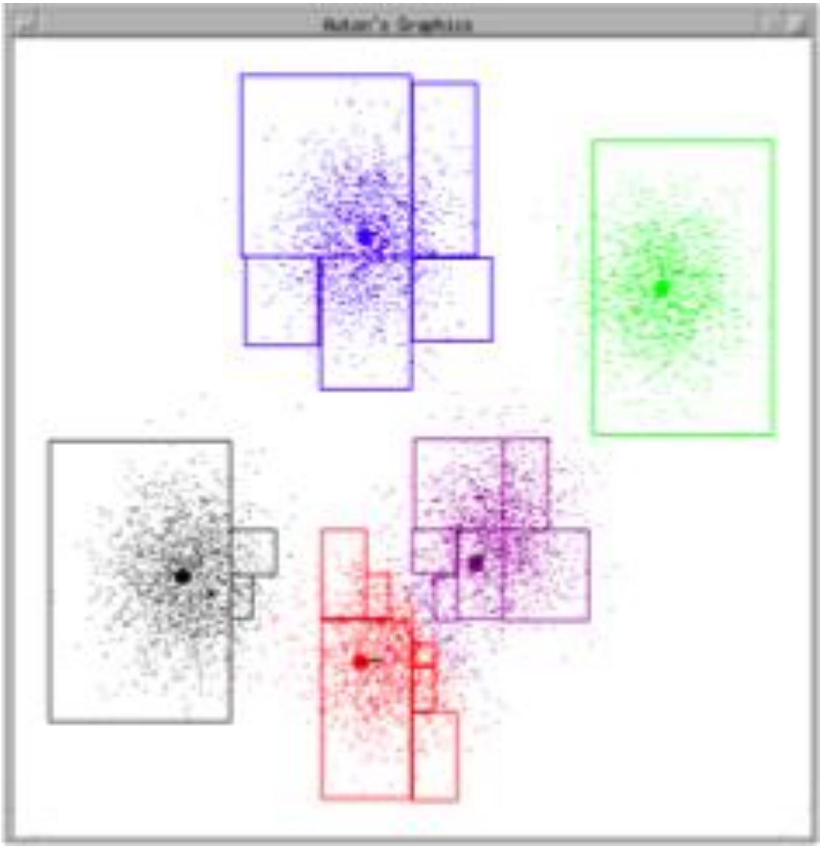
K-means algorithm continues (7)



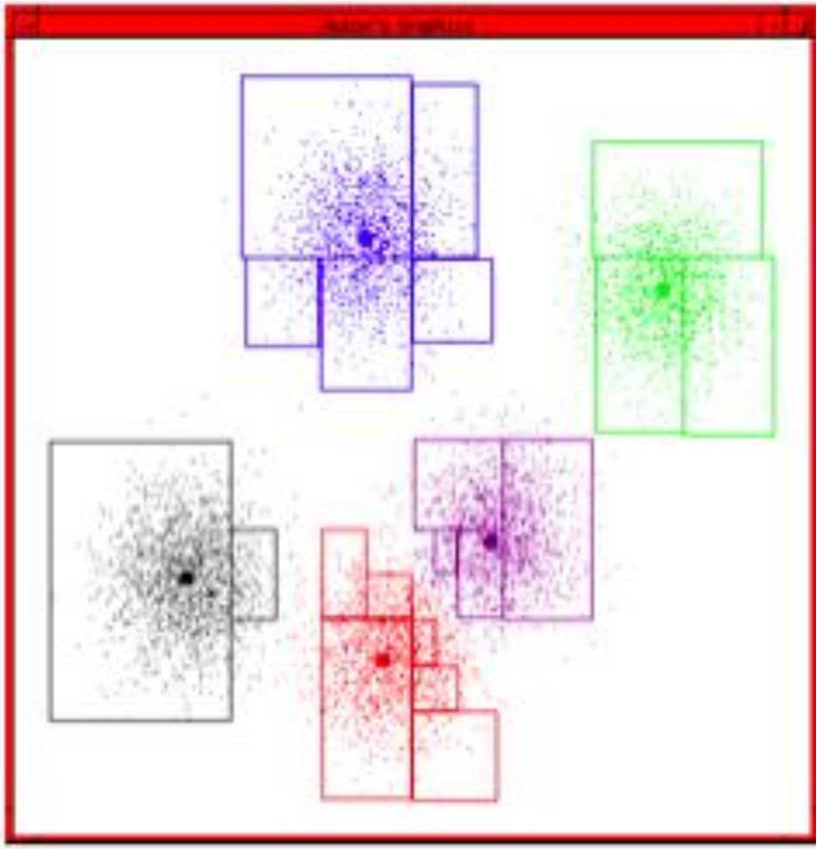
K-means algorithm continues (8)



K-means algorithm continues (9)



K-means algorithm terminates



Starting and Stopping K-means

- How to decide on value of K?
  - User must specify – not always easy to do!
  - Could try for different values of K, and inspect output
  - There are more sophisticated methods which attempt to learn a good value of K, given assumptions about cluster coherence and shape
- When to stop?
  - Usually stop when label of datapoints / cluster centres stop changing

K-means and EM

- K-means as shown above is just the hard EM algorithm, where the underlying model that generated the data is a **mixture of Gaussian distributions**.
  - i.e., each cluster (each type of apples) is generated by a different Gaussian distribution with its own mean and variance.
- There exists an analogous algorithm for the standard, soft EM algorithm for a mixture of Gaussians.
  - In E-step, must assign a **responsibility** score for each data point being generated by each Gaussian distribution in the mixture.
  - M-step update must then use these responsibility scores to weight the samples when computing the centroids.

Properties of K-means

- Time complexity? *O(nkd)* where k = #centers
  n = #datapoints
  d = dimensionality of data
- Optimality?
  - Converges to a local optimum.
  - Can use random re-starts to get better local optimum.
  - Alternately, can choose your initial centers carefully:
    - Place μ_1 on top of a randomly chosen datapoint.
    - Place μ_2 on top of datapoint that is furthest from μ_1.
    - Place μ_3 on top of datapoint that is furthest from both μ_1 and μ_2.

What you should know

- Learning maximum-likelihood parameters with missing data.
  - EM algorithm in general case.
  - Properties of EM (complexity, convergence).

- Clustering of data using K-means algorithm
  - Basic procedure.
  - Properties of K-means (complexity, convergence).

- Relation between EM and K-means.