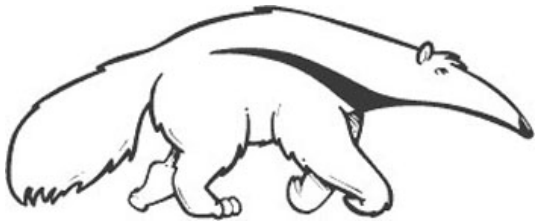
+

CS178: Machine Learning and Data Mining

Final Review

Prof. Alexander Ihler



Final Format

Friday, December 14 from 8:00-10:00am:

- 2 hour exam, here (SSLH 100)
- Arrive early, find seat listed on exam

What you may bring:

- One (double-sided) 8.5x11-inch sheet of handwritten notes
- Pencil and/or pen
- Scratch sheets will be part of the exam booklet
- *Electronic devices are not allowed*

Machine Learning

Pre-Midterm Topics (supervised learning, see previous review)

VC Dimension

Clustering Methods

Decision Trees

Dimensionality Reduction

Ensemble Methods

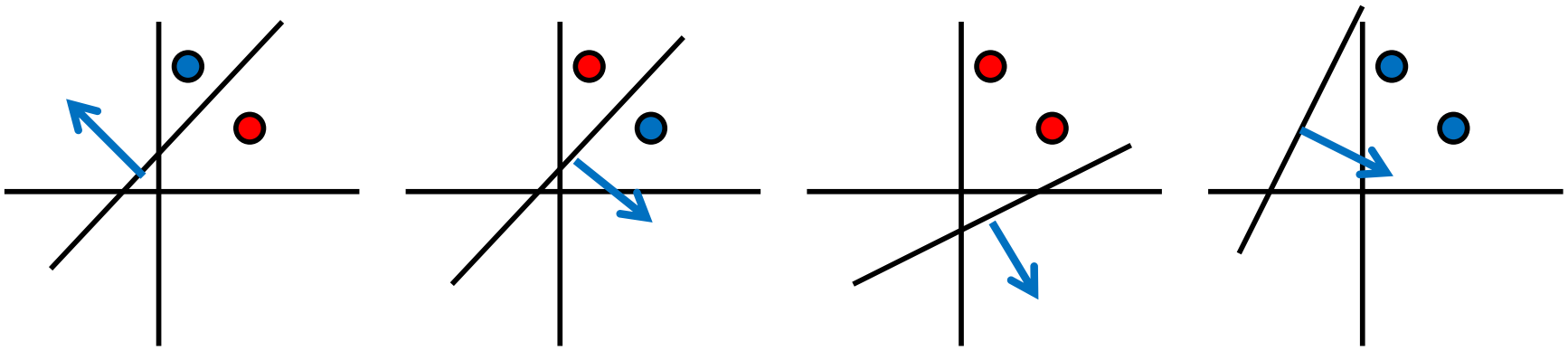
Reinforcement Learning

Shattering

- We say a classifier f(x) can *shatter* points $x^{(1)}…x^{(h)}$ iff
  For *all* $y^{(1)}…y^{(h)}$, f(x) can achieve zero error on training data $(x^{(1)},y^{(1)})$, $(x^{(2)},y^{(2)})$, … $(x^{(h)},y^{(h)})$

  (i.e., there exists some θ that gets zero error)

- Can $f(x;\theta) = \text{sign}(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$ shatter these points?
- Yes: there are 4 possible training sets…



VC Dimension

- The VC dimension H is defined as:

  The maximum number of points h that *can be arranged* so that f(x) can shatter them

- A game:
  - Fix the definition of f(x;θ)
  - Player 1: choose locations $x^{(1)}$…$x^{(h)}$
  - Player 2: choose target labels $y^{(1)}$…$y^{(h)}$
  - Player 1: choose value of θ
  - If f(x;θ) can reproduce the target labels, P1 wins

$$\exists\{x^{(1)}\ldots x^{(h)}\}\ s.t.\ \ \forall\{y^{(1)}\ldots y^{(h)}\}\ \ \exists\theta\ s.t.\ \ \forall i\ f(x^{(i)};\theta)=y^{(i)}$$

VC Dimension and Risk

- Given some classifier, let H be its VC dimension
  - Represents "representational power" of classifier

$$R(\theta) = \text{TestError} = \mathbb{E}[\mathbb{1}[c \neq \hat{c}(x\,;\,\theta)]]$$

$$R^{\text{emp}}(\theta) = \text{TrainError} = \frac{1}{m}\sum_i \mathbb{1}[c^{(i)} \neq \hat{c}(x^{(i)}\,;\,\theta)]$$

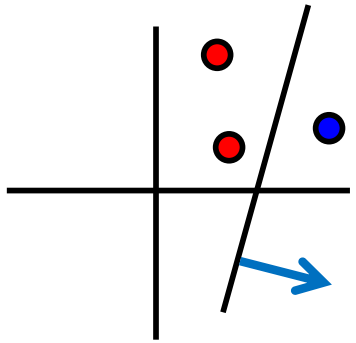
- With "high probability" (1-η), Vapnik showed

$$\text{TestError} \leq \text{TrainError} + \sqrt{\frac{H\log(2m/H) + H - \log(\eta/4)}{m}}$$

VC Dimension

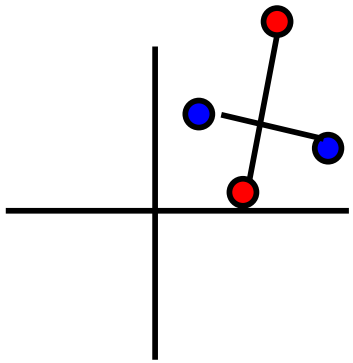
- Example: what's the VC dimension of the two-dimensional line, $f(x;\theta) = \text{sign}(\theta_1 x_1 + \theta_2 x_2 + \theta_0)$?

- VC dim >= 3? Yes



- VC dim >= 4? No…

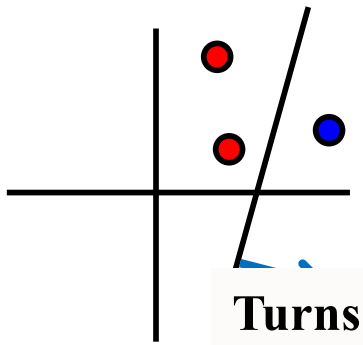
  Any line through these points must split one pair (by crossing one of the lines)



VC Dimension

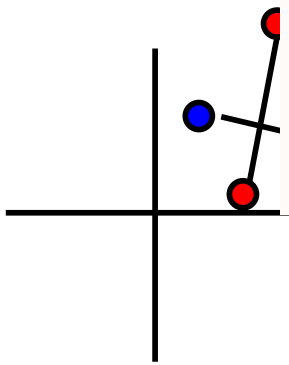
- Example: what's the VC dimension of the two-dimensional line, $f(x;\theta) = \text{sign}(\theta_1 x_1 + \theta_2 x_2 + \theta_0)$?

- VC dim >= 3? Yes



- VC dim >= 4? No…

  Any line through these points must split one pair (by crossing one of the lines)



Turns out:
For a general , linear classifier (perceptron) in d dimensions with a constant term:

VC dim = d+1

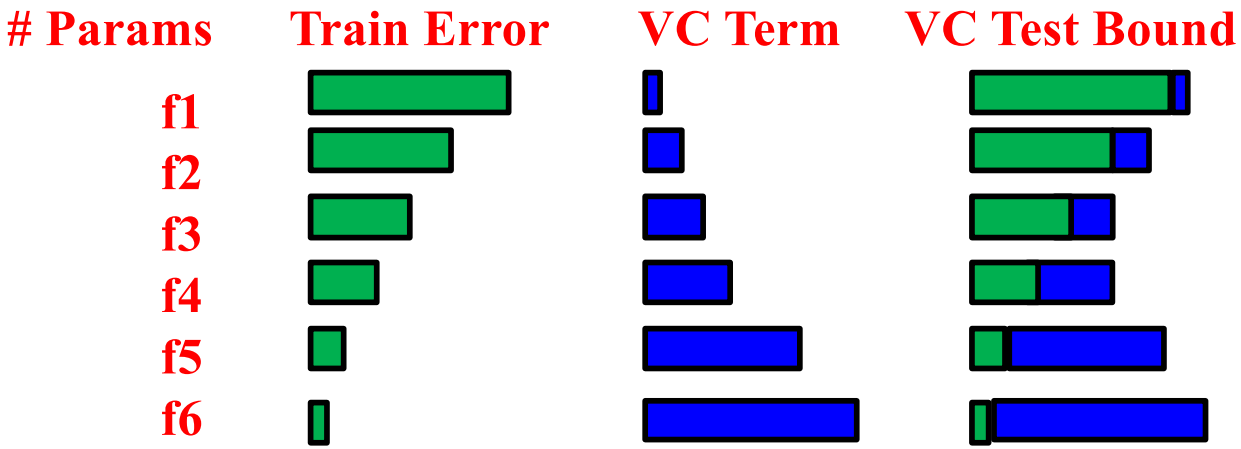
Using VC dimension

- Used validation / cross-validation to select complexity

Using VC dimension

- Used validation / cross-validation to select complexity
- Use VC dimension based bound on test error similarly

- "Structural Risk Minimization" (SRM)



Machine Learning

Pre-Midterm Topics (supervised learning, see previous review)

VC Dimension

Clustering Methods

Decision Trees

Dimensionality Reduction

Ensemble Methods

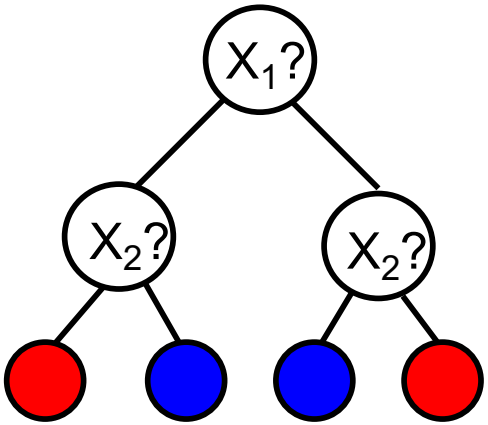
Reinforcement Learning

Decision trees

- Functional form f(x;θ): nested "if-then-else" statements
  - Discrete features: fully expressive (any function)
- Structure:
  - Internal nodes: check feature, branch on value
  - Leaf nodes: output prediction

"XOR"

x_1	x_2	y
0	0	1
0	1	-1
1	0	-1
1	1	1



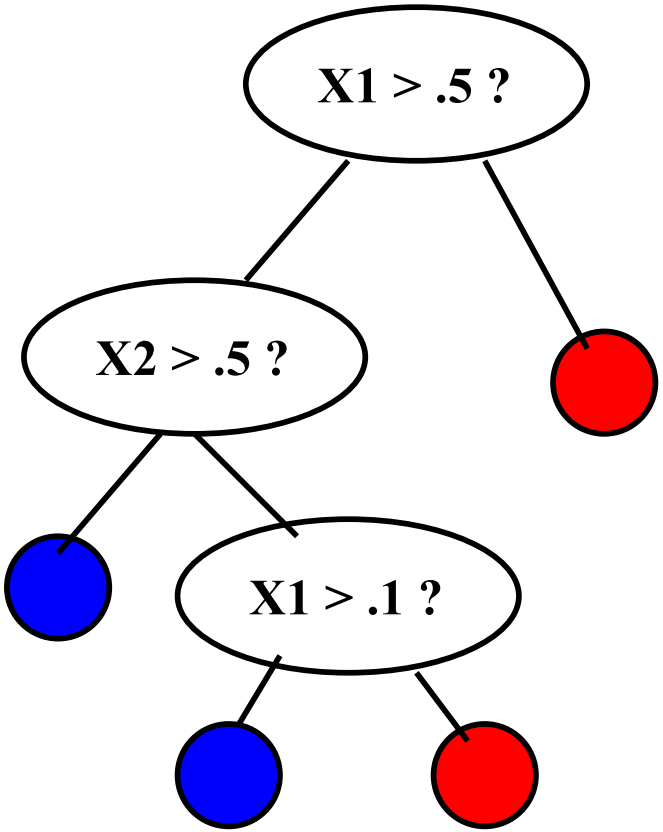

```
if X1:                      # branch on feature at root
   if X2:  return +1        # if true, branch on right child feature
   else:   return -1        # & return leaf value
else:  # left branch:
   if X2:  return -1        # branch on left child feature
   else:   return +1        # & return leaf value
```

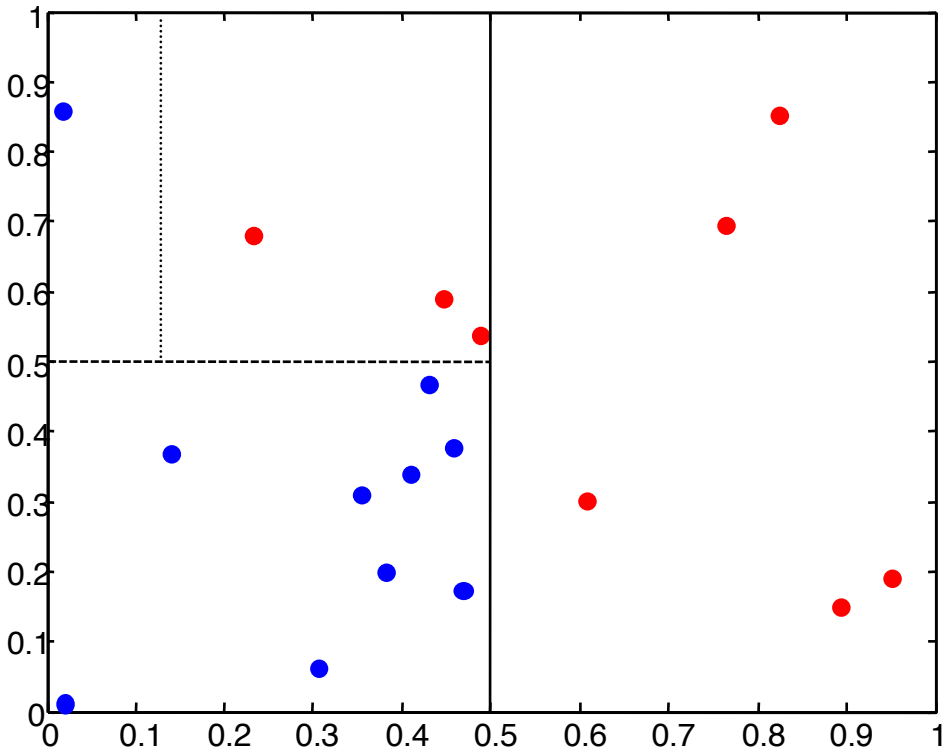
Parameters?

Tree structure, features, and leaf outputs

Decision trees

- Real-valued features
  - Compare feature value to some threshold

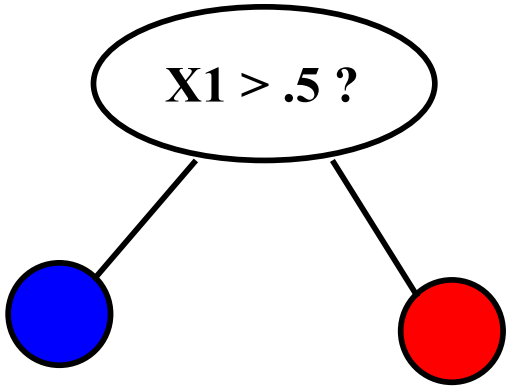


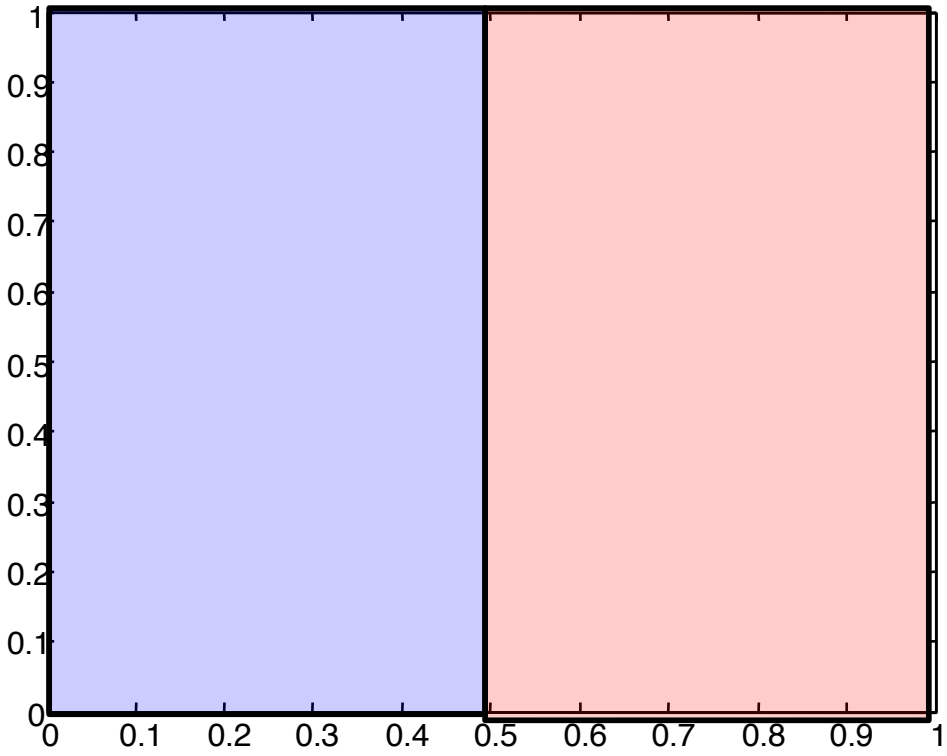


Decision trees

- "Complexity" of function depends on the depth

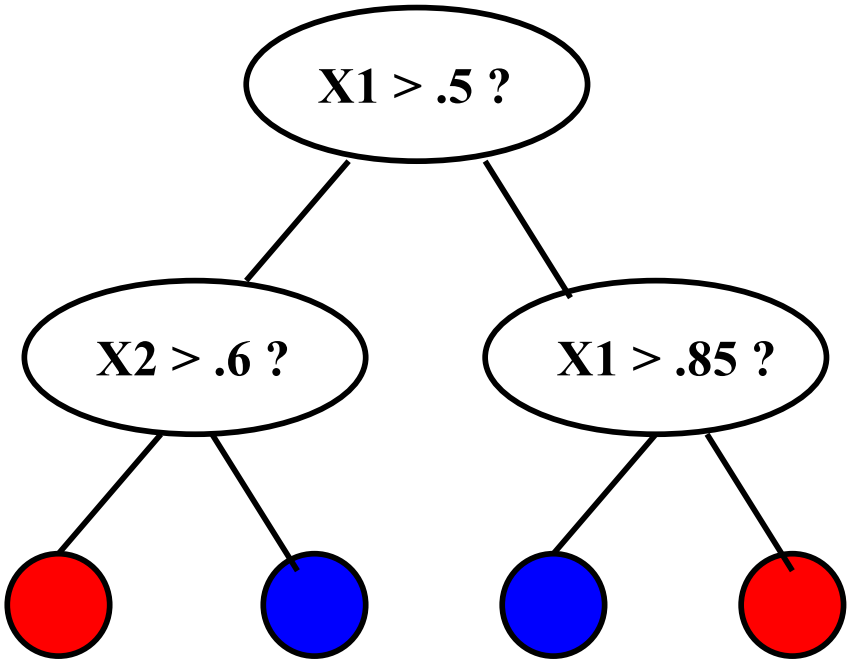
- A depth-1 decision tree is called a decision "stump"
  - Simpler than a linear classifier!

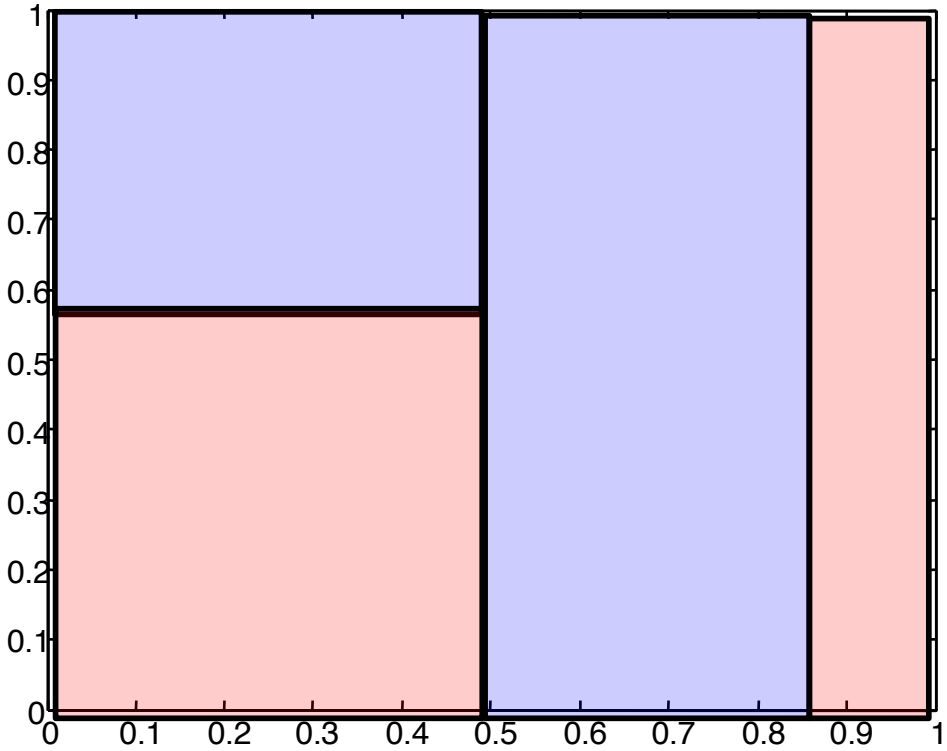




Decision trees

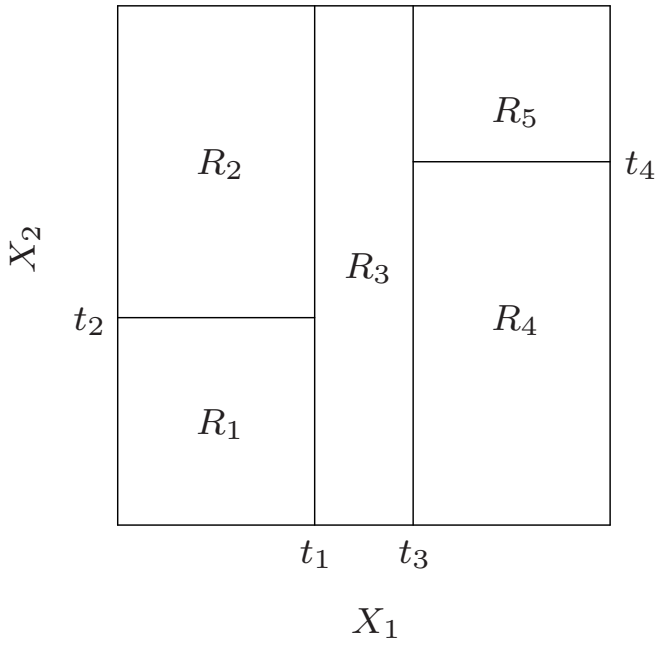
- "Complexity" of function depends on the depth
- More splits provide a finer-grained partitioning

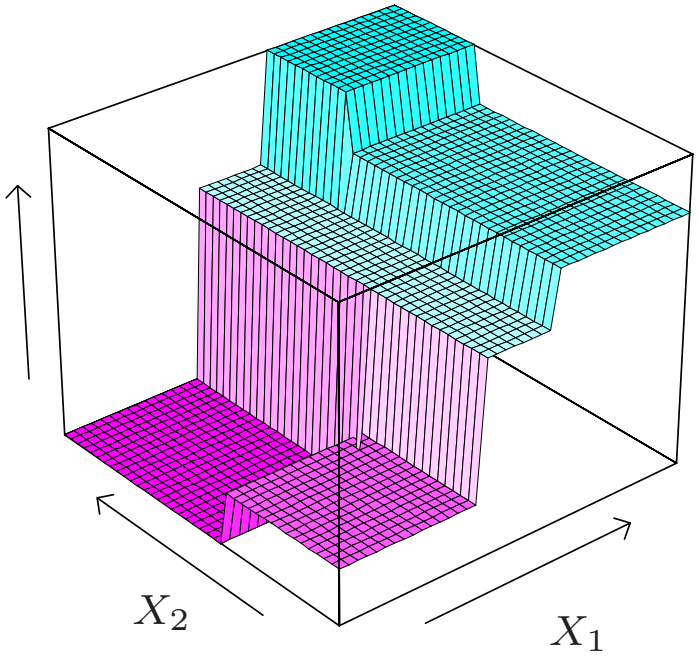


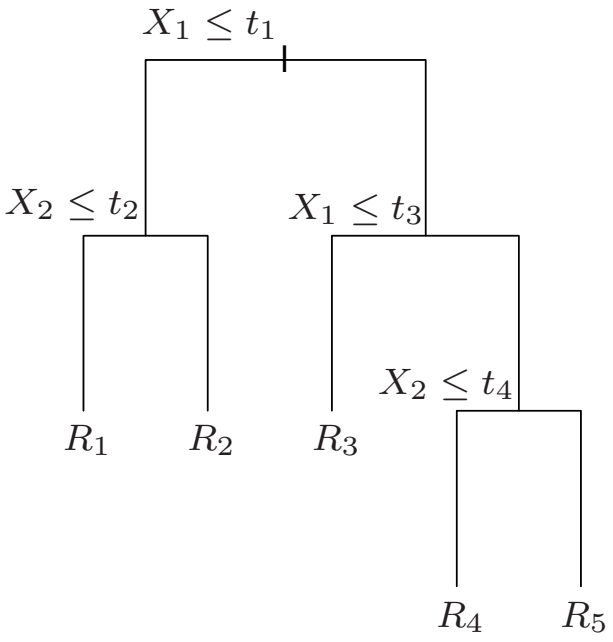


Depth d = up to 2^d regions & predictions

Decision Trees for 2D Regression



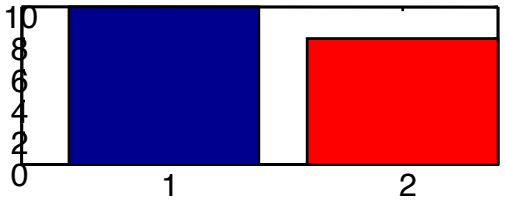




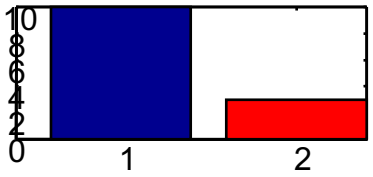
- **Each node in tree splits examples according to a single feature**
- **Leaves predict mean of training data whose path through tree ends there**

Entropy and information

- Information gain
  - How much is entropy reduced by measurement?
- Information: expected information gain

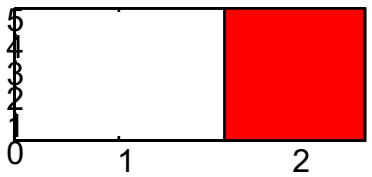


H = .99 bits



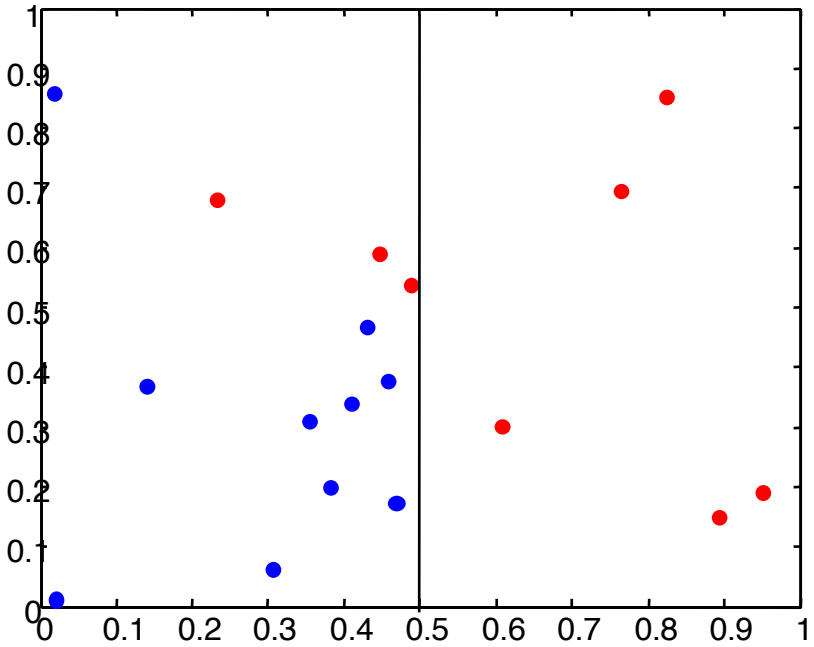
H = .77 bits

Prob = 13/18



H=0

Prob = 5/18

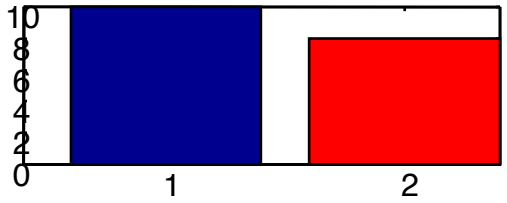


Information = 13/18 * (.99-.77) + 5/18 * (.99 – 0)

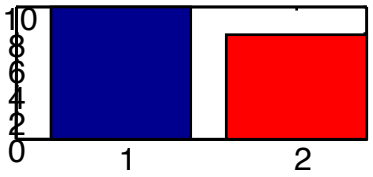
Equivalent: $\sum$ p(s,c) log [p(s,c) / p(s) p(c)]
= 10/18 log[(10/18) / (13/18) (10/18)] + 3/18 log[(3/18)/(13/18)(8/18) + …

Entropy and information

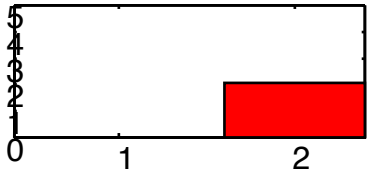
- Information gain
  - How much is entropy reduced by measurement?
- Information: expected information gain



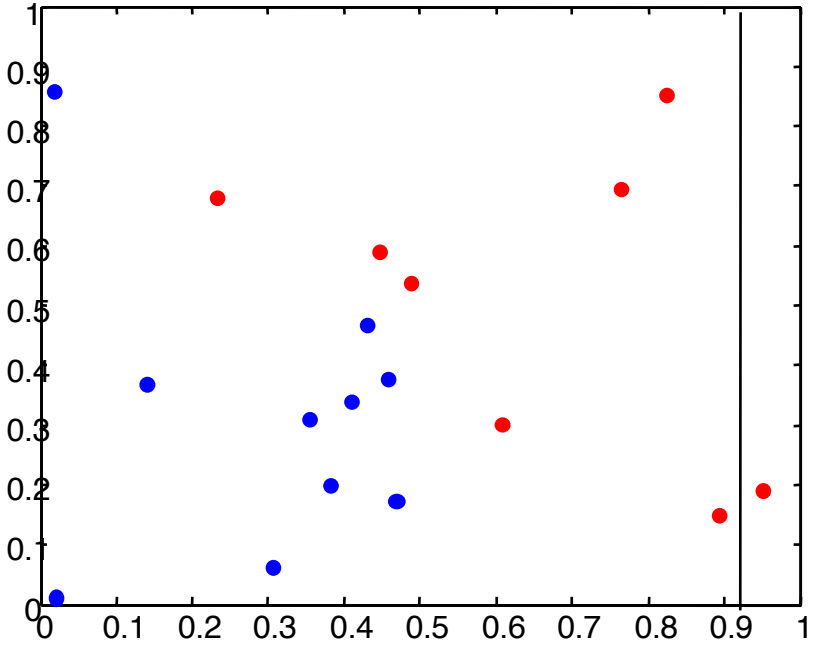
H = . 99 bits



H = .97 bits
Prob = 17/18



H=0
Prob = 1/18



Information = 17/18 * (.99-.97) + 1/18 * (.99 – 0)

Less information reduction – a less desirable split of the data

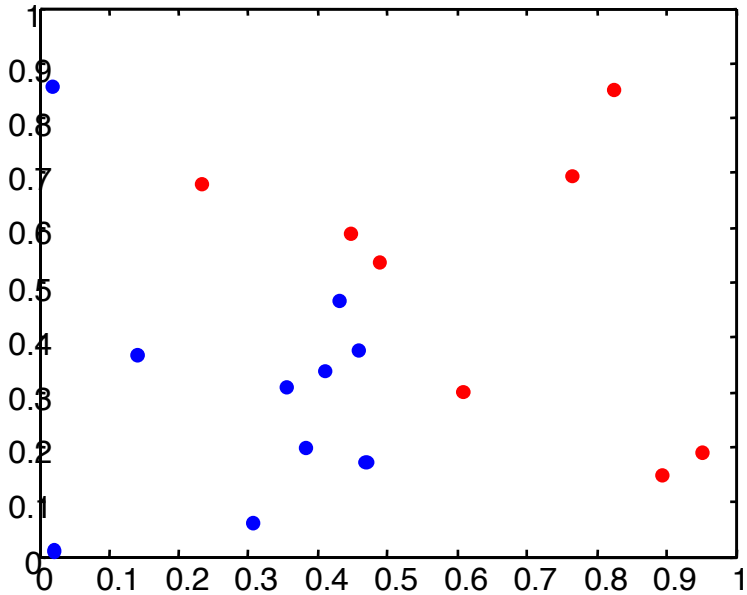
Scoring decision tree splits

Algorithm 1 FindBestSplit(D)

Input: A data set $D = (X, Y)$ of size m;
impurity function $H(\cdot)$.

Output: A split j^*, t^* minimizing impurity H

Initialize $H^* = 0$
for each feature j **do**
  Sort $\{x_j^{(i)}\}$ in order of increasing value
  for each i such that $x^{(i)} < x^{(i+1)}$ **do**
    Compute $p_c^L = \frac{1}{i} \sum_{k \le i} \mathbb{1}[y^{(k)} = c]$
      and $p_c^R = \frac{1}{k-i} \sum_{k > i} \mathbb{1}[y^{(k)} = c]$
    Set $H' = \frac{i}{m} H(p^L) + \frac{m-i}{m} H(p^R)$
    if $H' < H^*$ **then**
      Set $j^* = j$, $t^* = (x^{(i)} - x^{(i+1)})/2$, $H^* = H'$
    end if
  end for
end for
Return j^*, t^*



Building a decision tree

Algorithm 1 BuildTree(D): Greedy training of a decision tree

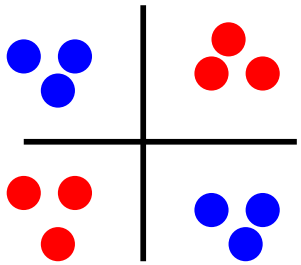
Input: A data set $D = (X, Y)$.

Output: A decision tree.

if LeafCondition(D) **then**
  $f_n = \text{FindBestPrediction}(D)$
else
  $j_n, t_n = \text{FindBestSplit}(D)$

  $D_L = \{(x^{(i)}, y^{(i)}) : x^{(i)}_{j_n} < t_n\}$ and
  $D_R = \{(x^{(i)}, y^{(i)}) : x^{(i)}_{j_n} \geq t_n\}$

  $\text{leftChild} = \text{BuildTree}(D_L)$
  $\text{rightChild} = \text{BuildTree}(D_R)$
end if

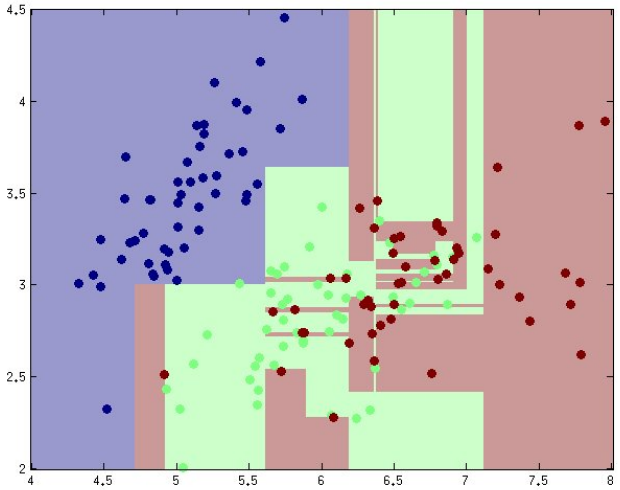


Stopping conditions:
* # of data < K
* Depth > D
* All data indistinguishable (discrete features)
* Prediction sufficiently accurate

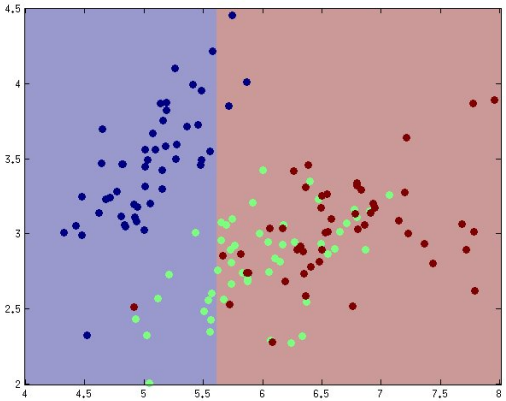
* Information gain threshold?
  Often not a good idea!
  No single split improves,
    but, two splits do.
  Better: build full tree, then prune

Controlling complexity

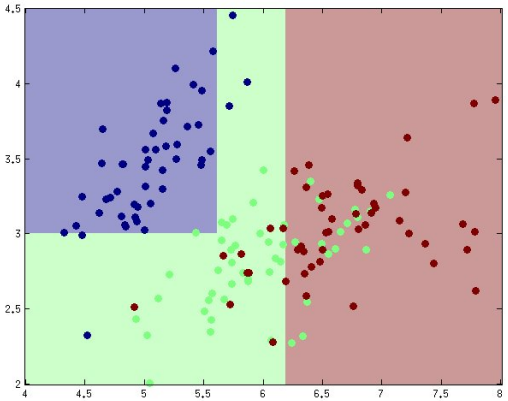
- Maximum depth cutoff



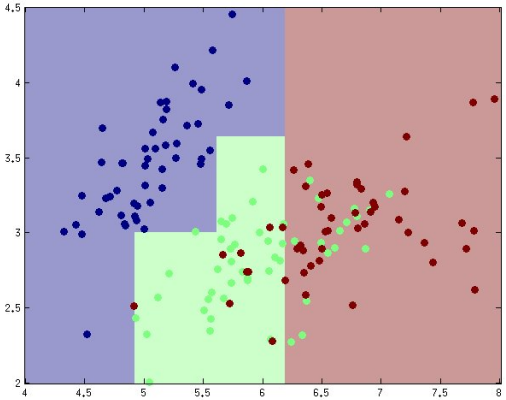
No limit



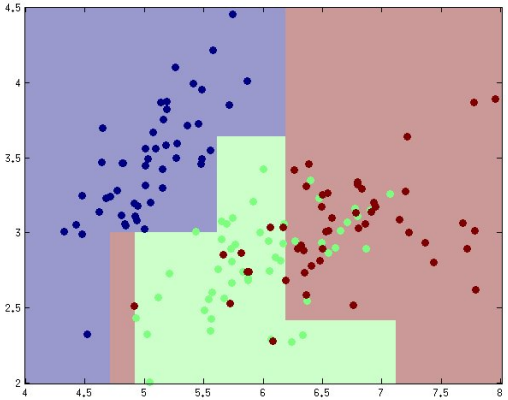
Depth 1



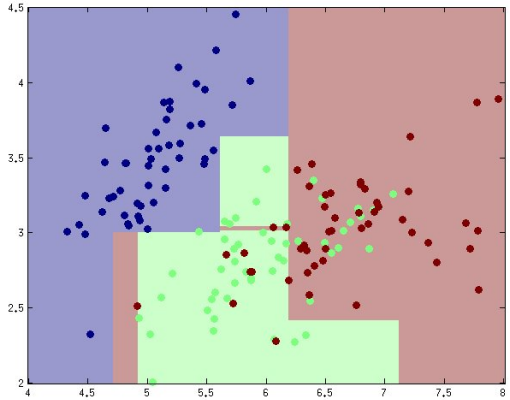
Depth 2



Depth 3



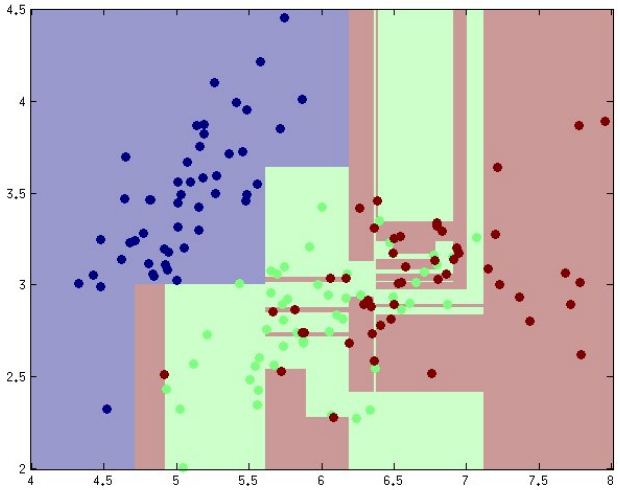
Depth 4



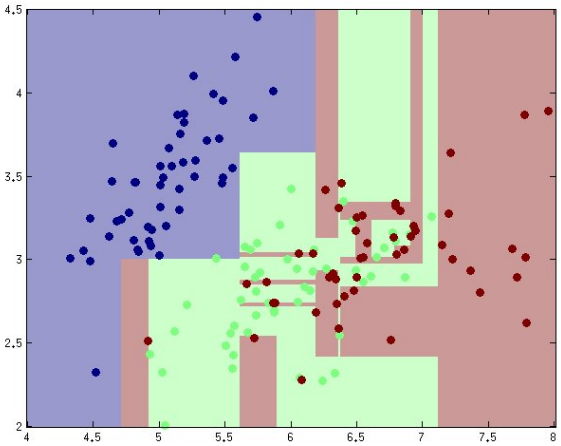
Depth 5

Controlling complexity

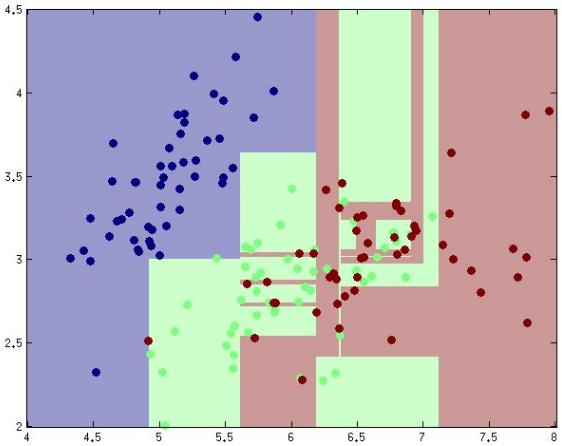
- Minimum # parent data



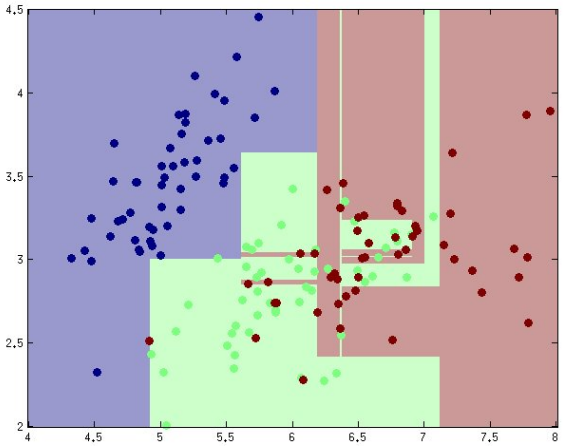
minParent 1



minParent 3



minParent 5



minParent 10

Machine Learning

Pre-Midterm Topics (supervised learning, see previous review)

VC Dimension

Clustering Methods

Decision Trees

Dimensionality Reduction

Ensemble Methods

Reinforcement Learning

“Stacked” ensembles

- Train a “predictor of predictors”
  - Treat individual predictors as features

$$\hat{y}_1 = f_1(x_1, x_2, \ldots)$$
$$\hat{y}_2 = f_2(x_1, x_2, \ldots) \quad \Rightarrow \quad \hat{y}_e = f_e(\hat{y}_1, \hat{y}_2, \ldots)$$
$$\ldots$$

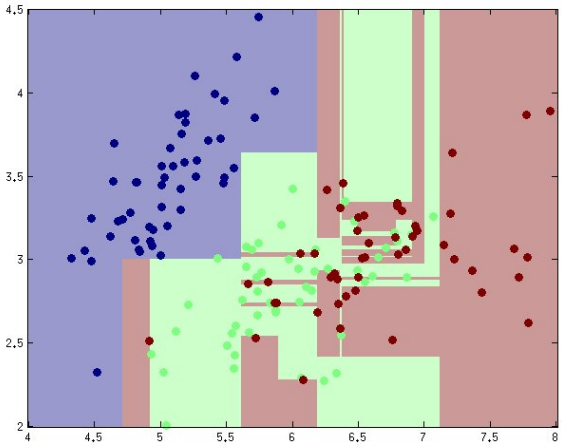
  - Similar to multi-layer perceptron idea
  - Special case: binary, f_e linear => weighted vote
  - Can train stacked learner f_e on validation data
    - Avoids giving high weight to overfit models

Bagging

- Bootstrap
  - Create a random subset of data by sampling
  - Draw m' of the m samples, with replacement (some variants w/o)
    - Some data left out; some data repeated several times
- Bagging
  - Repeat K times
    - Create a training set of $m' \leq m$ examples
    - Train a classifier on the random training set
  - To test, run each trained classifier
    - Each classifier votes on the output, take majority
    - For regression: each regressor predicts, take average
- Notes:
  - Some complexity control: harder for each to memorize data
  - Doesn't work for linear models (average of linear functions is linear function), but perceptrons OK (linear + threshold = nonlinear)

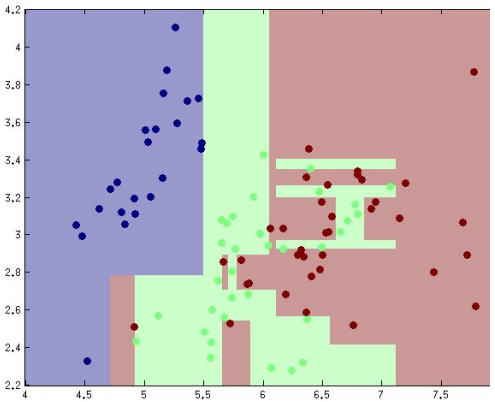
Bagged decision trees

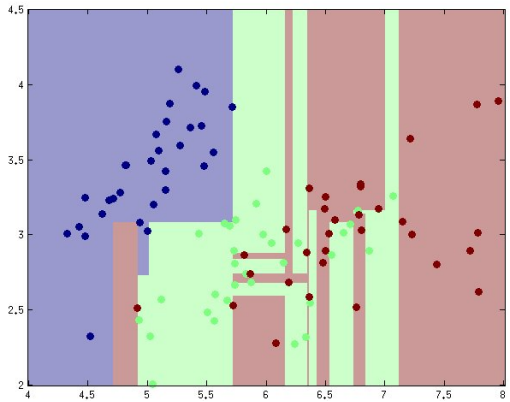
- Randomly resample data
- Learn a decision tree for each
  - No max depth = very flexible class of functions
  - Learner is low bias, but high variance

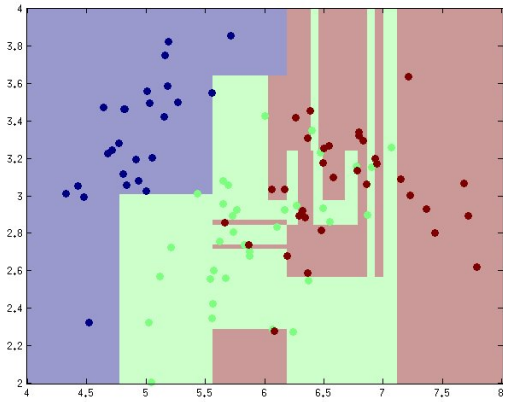


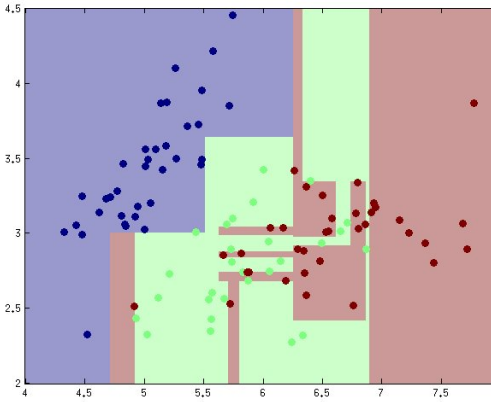
Full data set

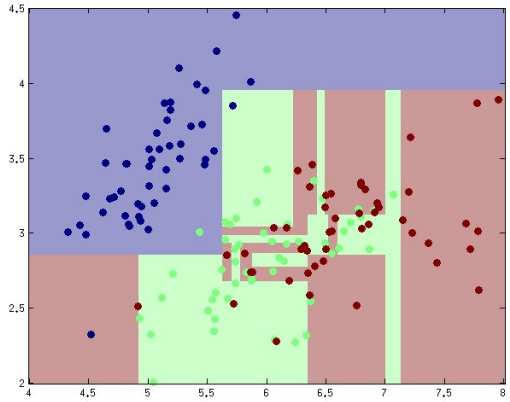
Sampling:
simulates “equally likely”
data sets we could have
observed instead, &
their classifiers





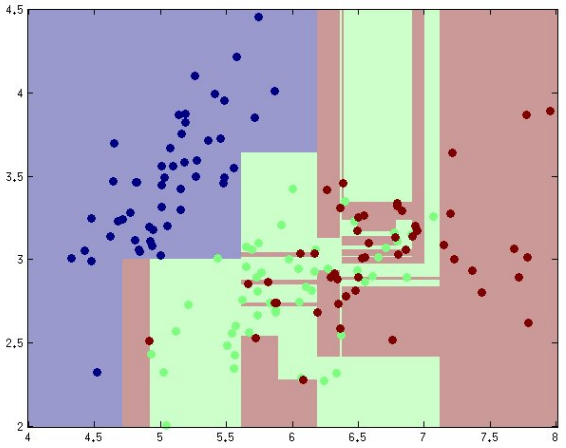




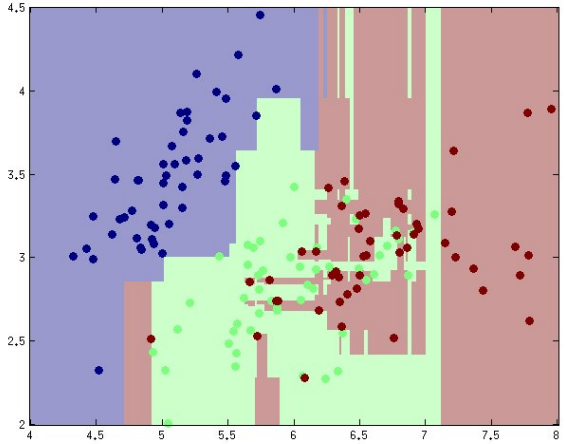


Bagged decision trees

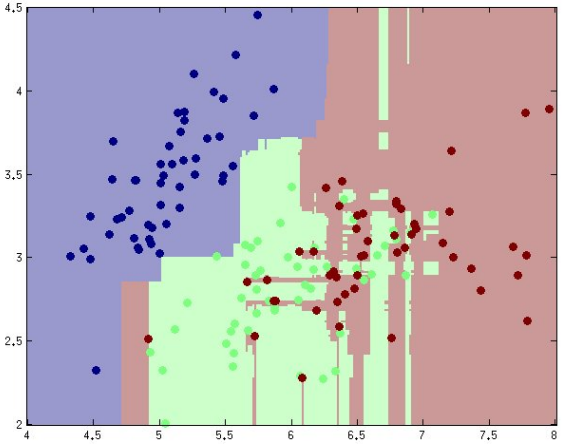
- Average over collection
  - Classification: majority vote
- Reduces memorization effect
  - Not every predictor sees each data point
  - Lowers effective “complexity” of the overall average
  - Usually, better generalization performance
  - Intuition: reduces variance while keeping bias low


Full data set

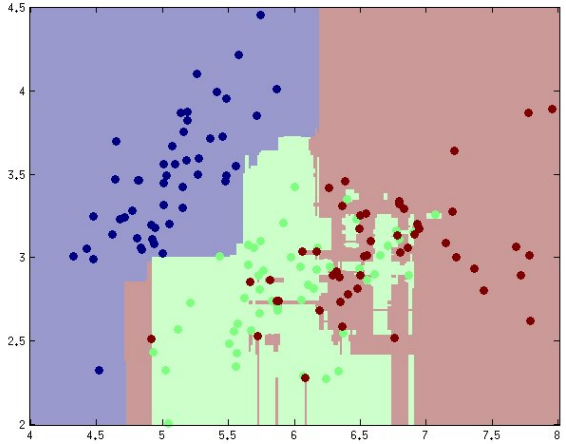
Avg of 5 trees



Avg of 25 trees



Avg of 100 trees



Random forests

- Bagging applied to decision trees
- Problem
  - With lots of data, we usually learn the same classifier
  - Averaging over these doesn't help!
- Introduce extra variation in learner
  - At each step of training, only allow a (random) subset of features
  - Enforces diversity ("best" feature not available)
  - Keeps bias low (every feature available eventually)
  - Average over these learners (majority vote)

```
# in FindBestSplit(X,Y):
    for each of a subset of features
        for each possible split
            Score the split  (e.g. information gain)
    Pick the feature & split with the best score
    Recurse on left & right splits
```

Gradient boosting

- Make a set of predictions $\hat{y}[i]$

- The "error" in our predictions is $J(y,\hat{y})$
  - For MSE: $J(.) = \sum (y[i] - \hat{y}[i])^2$

- We can "adjust" $\hat{y}$ to try to reduce the error
  - $\hat{y}[i] = \hat{y}[i] + \text{alpha } f[i]$
  - $f[i] \approx \nabla J(y, \hat{y})$ $= (y[i]-\hat{y}[i])$ for MSE

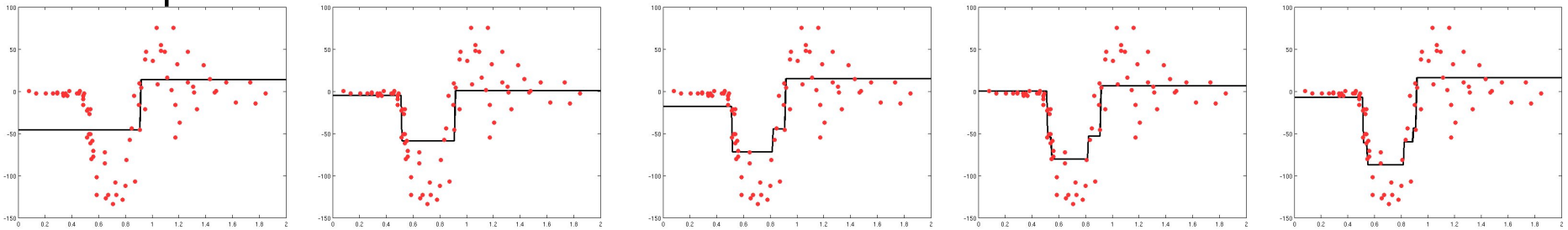
- Each learner is estimating the gradient of the loss function
- Gradient descent: take sequence of steps to reduce J
  - Sum of predictors, weighted by step size alpha

Gradient boosting

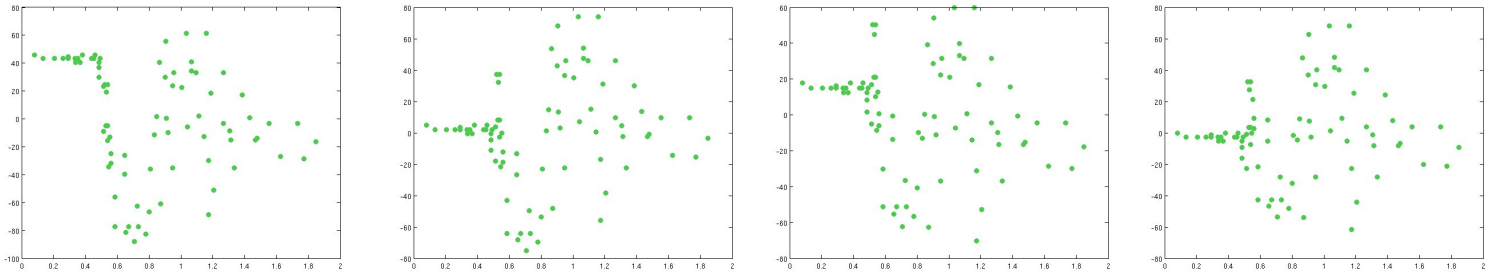
- Learn sequence of predictors
- Sum of predictions is increasingly accurate
- Predictive function is increasingly complex

$$y^{(i)} \approx \sum_z f_z(x^{(i)})$$

Data & prediction function

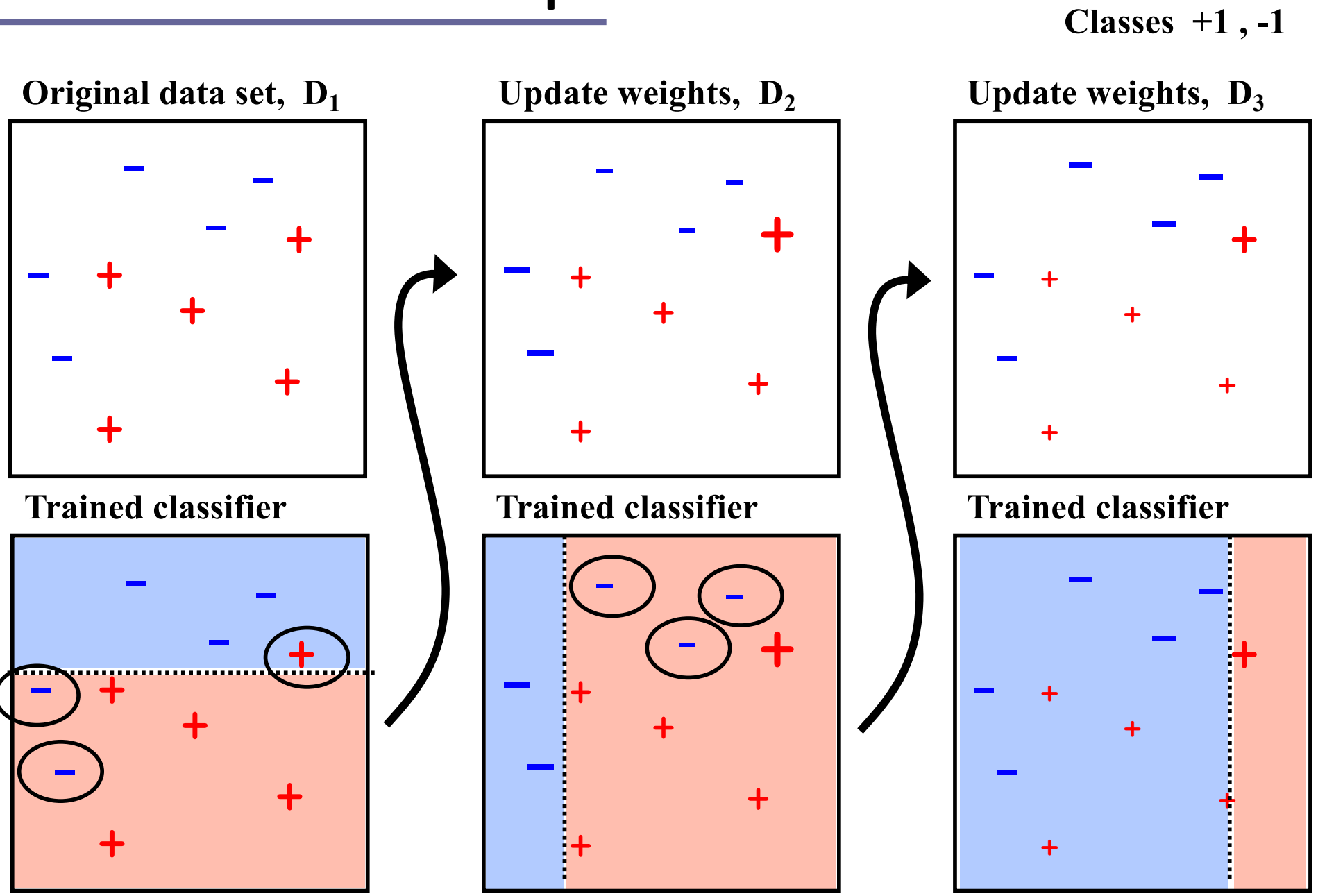


Error residual



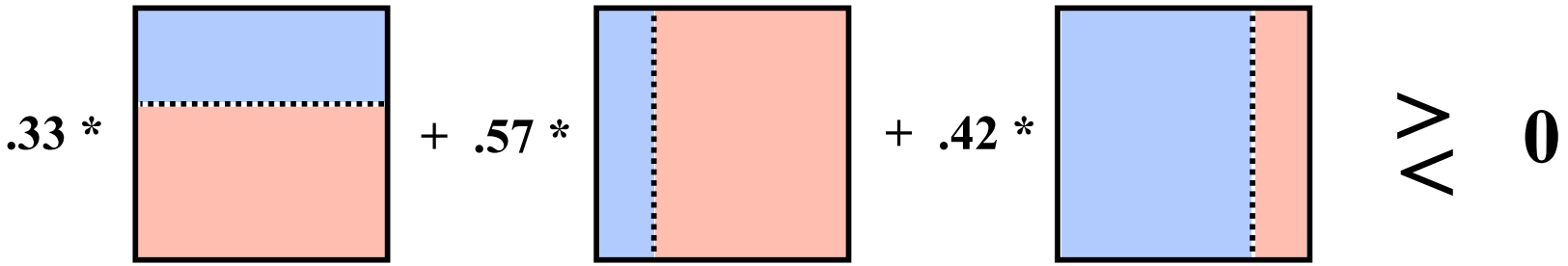
• • •

Adaboost example

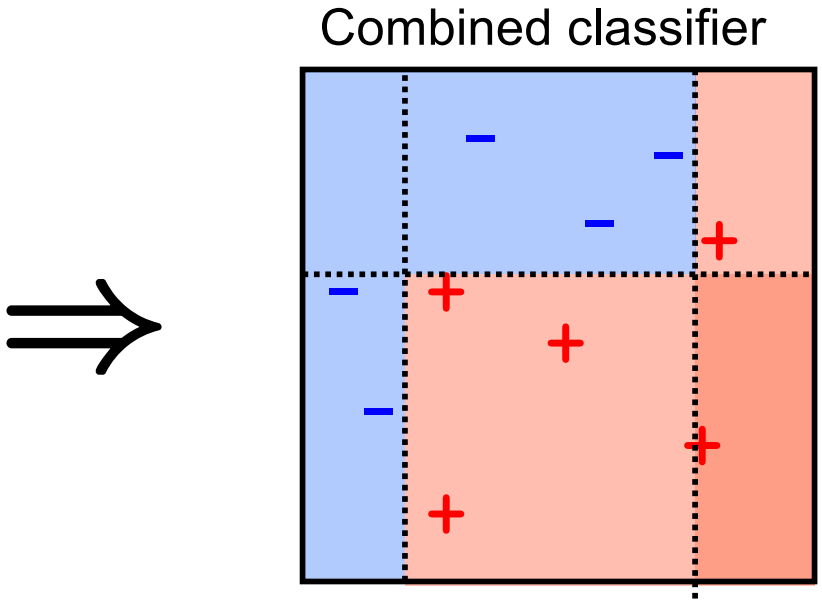


Adaboost example

Weight each classifier and combine them:



Combined classifier



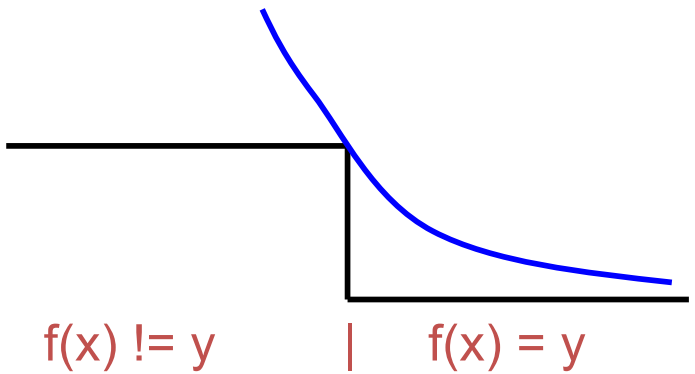
1-node decision trees
"decision stumps"
very simple classifiers

AdaBoost theory

- Minimizing classification error was difficult
  - For logistic regression, we minimized MSE or NLL instead
  - Idea: low MSE => low classification error
- Example of a surrogate loss function
- AdaBoost also corresponds to a surrogate loss function

$$C_{ada} = \sum_i \exp[-y^{(i)} f(x^i)]$$

- Prediction is yhat = sign(f(x))
  - If same as y, loss < 1; if different, loss > 1; at boundary, loss=1
- This loss function is smooth & convex (easier to optimize)



Machine Learning

Pre-Midterm Topics (supervised learning, see previous review)

VC Dimension

Clustering Methods

Decision Trees

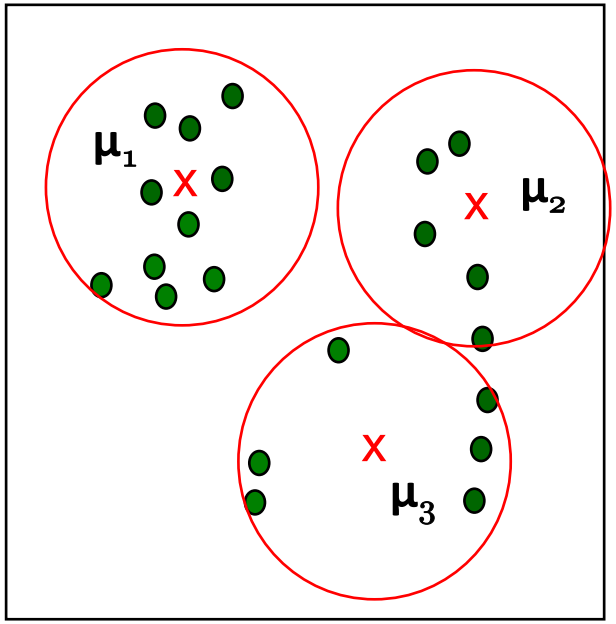
Dimensionality Reduction

Ensemble Methods

Reinforcement Learning

K-Means Clustering

- A simple clustering algorithm
- Iterate between
  - Updating the assignment of data to clusters
  - Updating the cluster's summarization



Notation:

Data example i has features x_i

Assume K clusters

Each cluster c "described" by a center μ_c

Each cluster will "claim" a set of nearby points

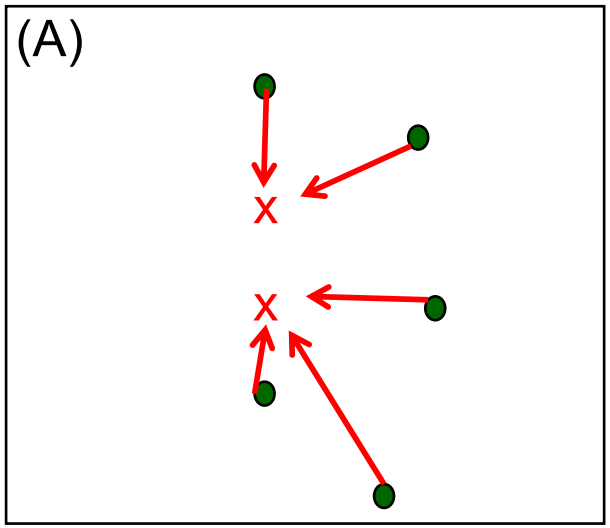
K-Means Clustering

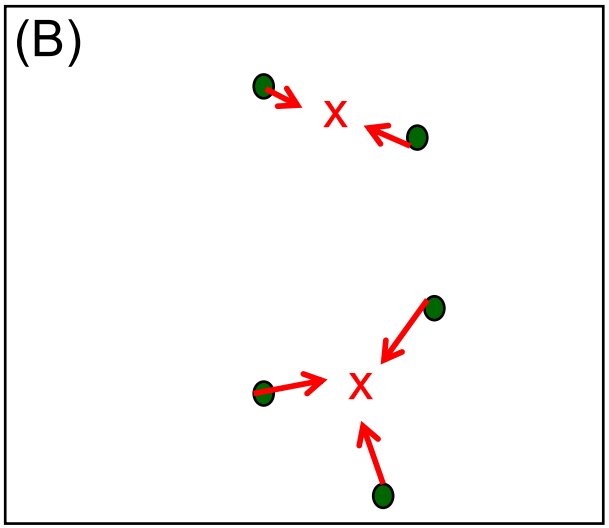
- Iterate until convergence:
  - (A) For each datum, find the closest cluster

$$z_i = \arg\min_c \|x_i - \mu_c\|^2 \qquad \forall i$$

  - (B) Set each cluster to the mean of all assigned data:

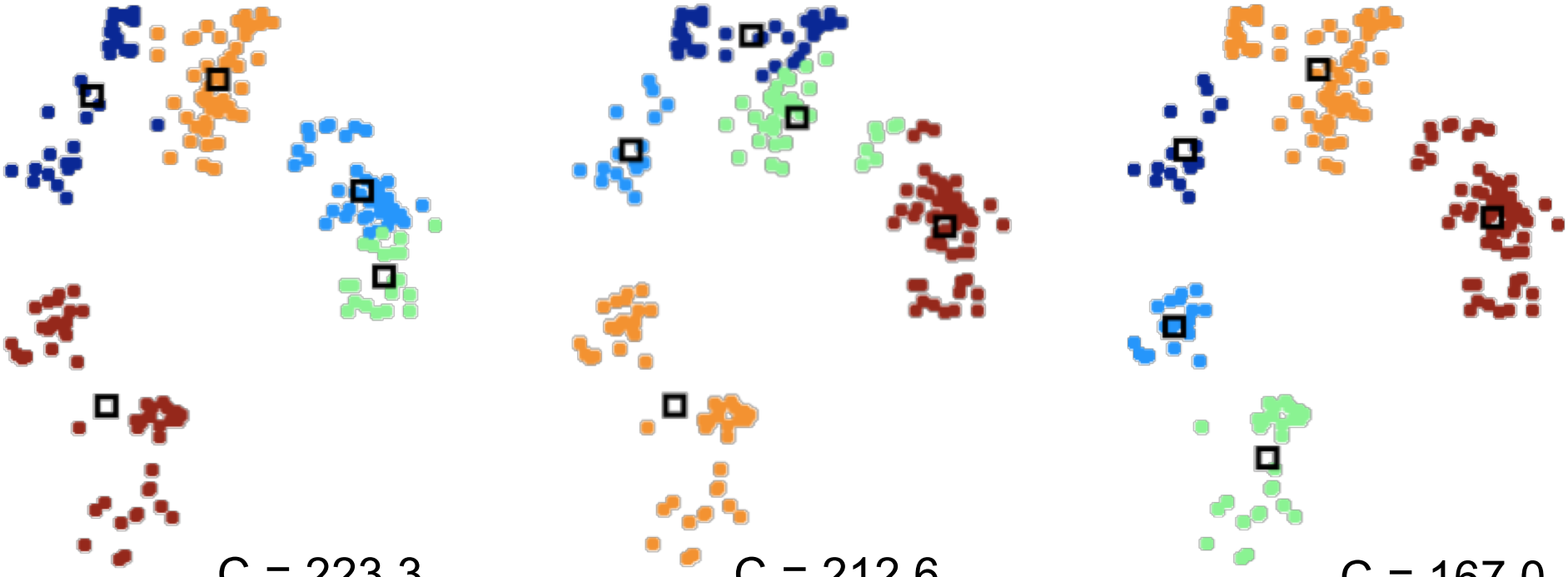
$$\forall c, \qquad \mu_c = \frac{1}{m_c} \sum_{i \in S_c} x_i \qquad S_c = \{i : z_i = c\},\ m_c = |S_c|$$





Initialization

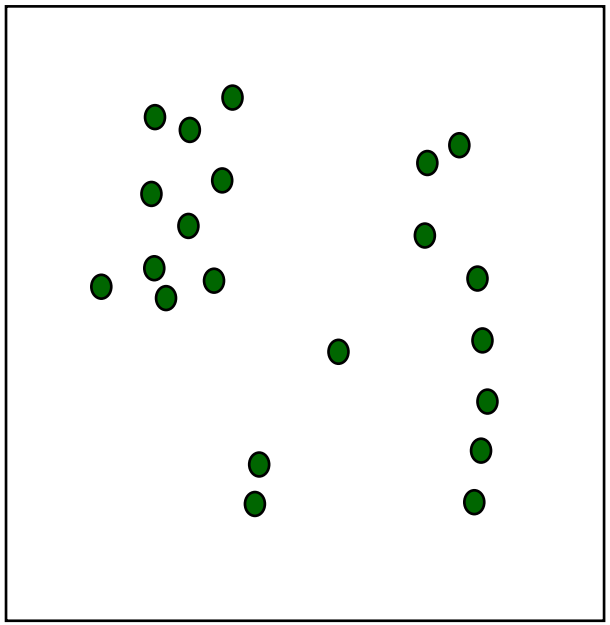
- Multiple local optima, depending on initialization
- Try different (randomized) initializations
- Can use cost C to decide which we prefer



Hierarchical Agglomerative Clustering

Initially, every datum is a cluster

Data:



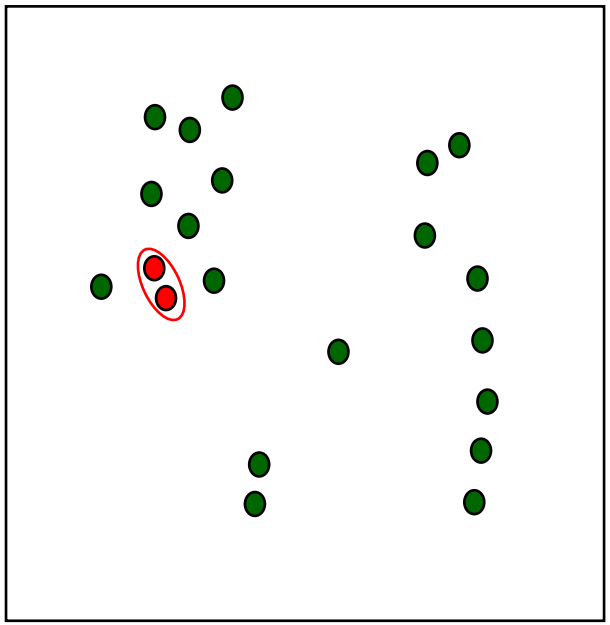
- A simple clustering algorithm
- Define a distance (or dissimilarity) between clusters (we'll return to this)
- Initialize: every example is a cluster
- Iterate:
  - Compute distances between all clusters (store for efficiency)
  - Merge two closest clusters
- Save both clustering and sequence of cluster operations
- "Dendrogram"

Algorithmic Complexity: $O(m^2 \log m) +$

Iteration 1

Builds up a sequence of clusters (“hierarchical”)

Data:



Dendrogram:

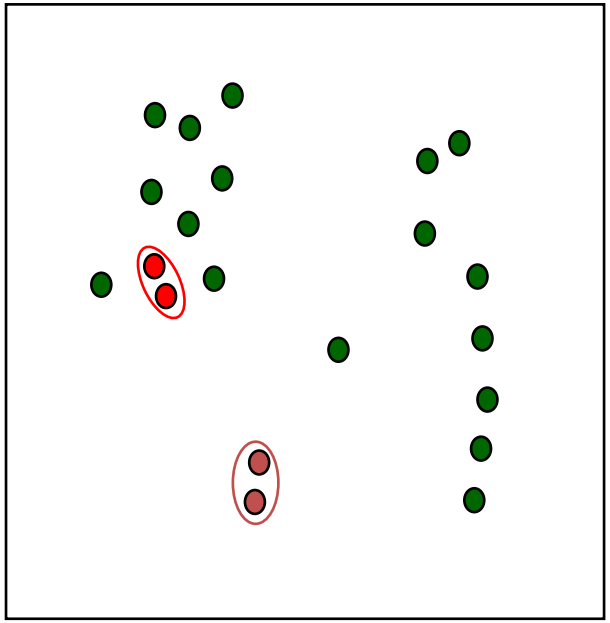
Height of the join
indicates dissimilarity

Algorithmic Complexity: $O(m^2 \log m) + O(m \log m) +$

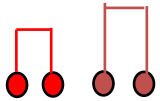
Iteration 2

Builds up a sequence of clusters (“hierarchical”)

Data:



Dendrogram:



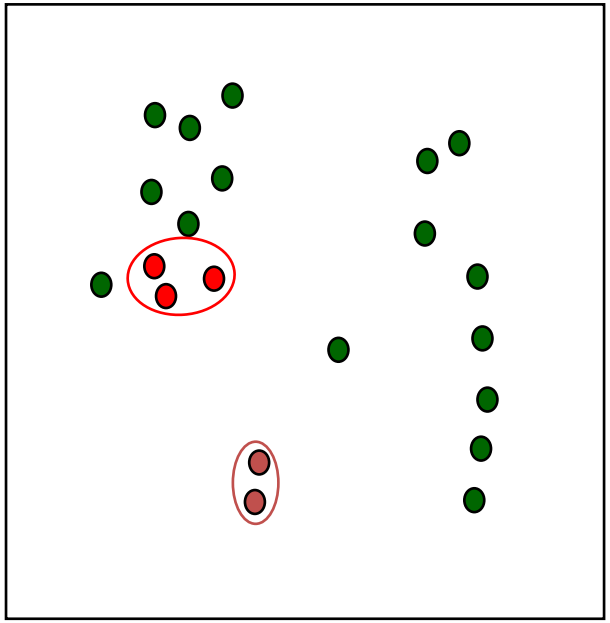
Height of the join indicates dissimilarity

Algorithmic Complexity: $O(m^2 \log m) + 2*O(m \log m) +$

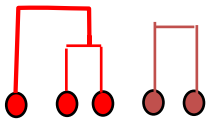
Iteration 3

Builds up a sequence of clusters (“hierarchical”)

Data:



Dendrogram:



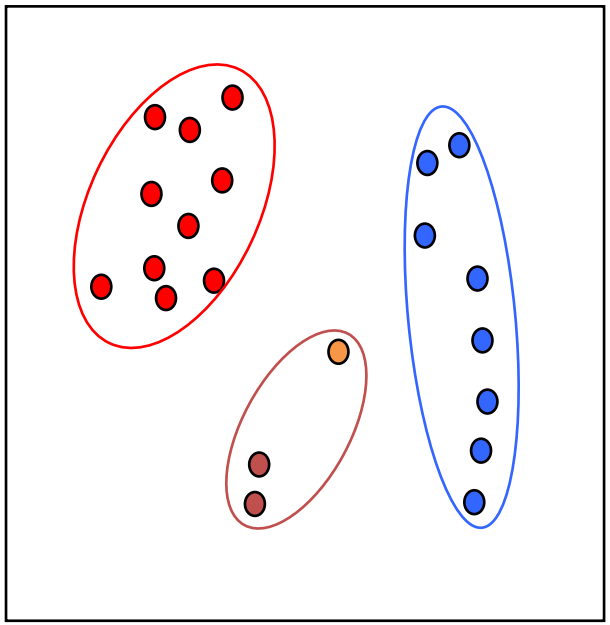
Height of the join indicates dissimilarity

Algorithmic Complexity: $O(m^2 \log m) + 3*O(m \log m) +$

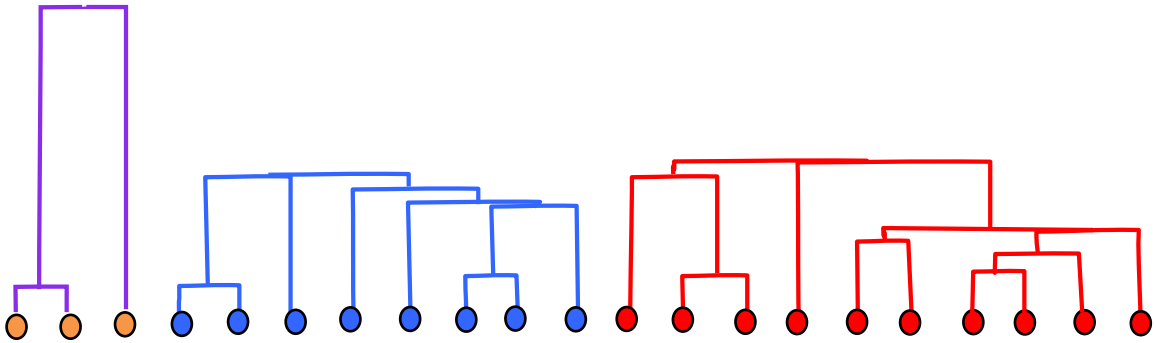
Iteration m-3

Builds up a sequence of clusters (“hierarchical”)

Data:



Dendrogram:



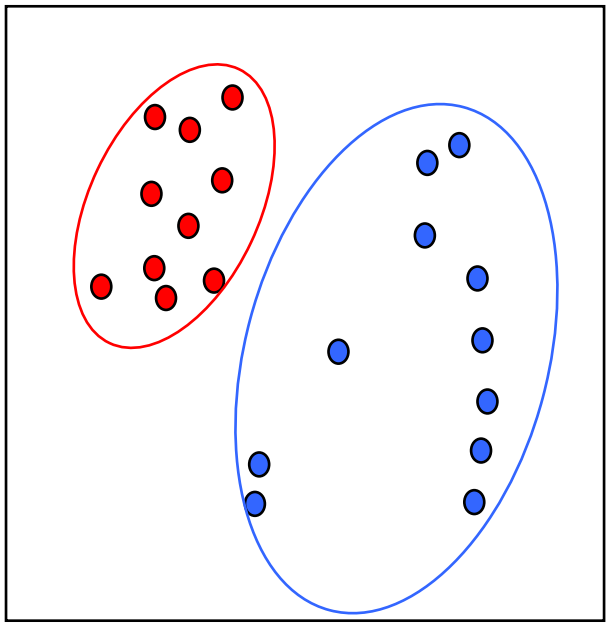
In mltools: “agglomerative”

Algorithmic Complexity: $O(m^2 \log m)$ + (m-3)*O(m log m) +

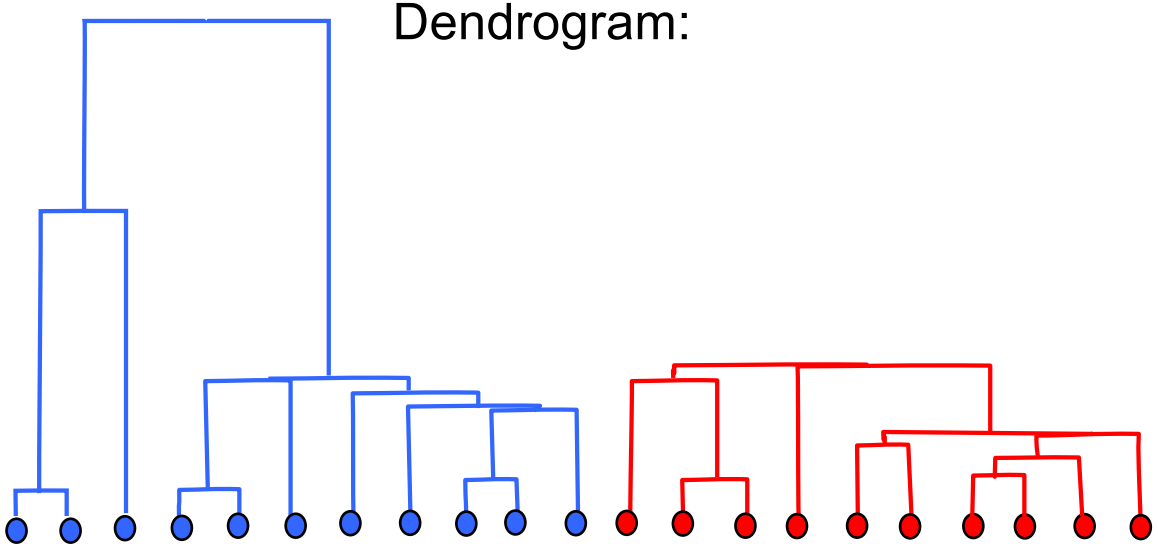
Iteration m-2

Builds up a sequence of clusters (“hierarchical”)

Data:



Dendrogram:



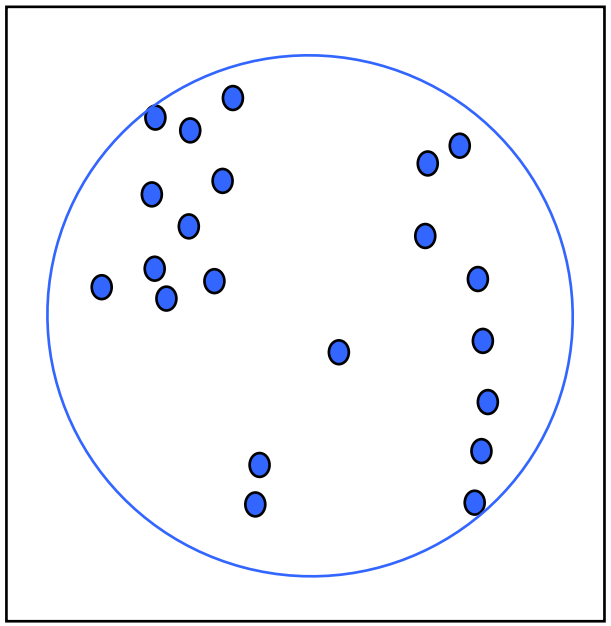
In mltools: “agglomerative”

Algorithmic Complexity: $O(m^2 \log m)$ + (m-2)*O(m log m) +

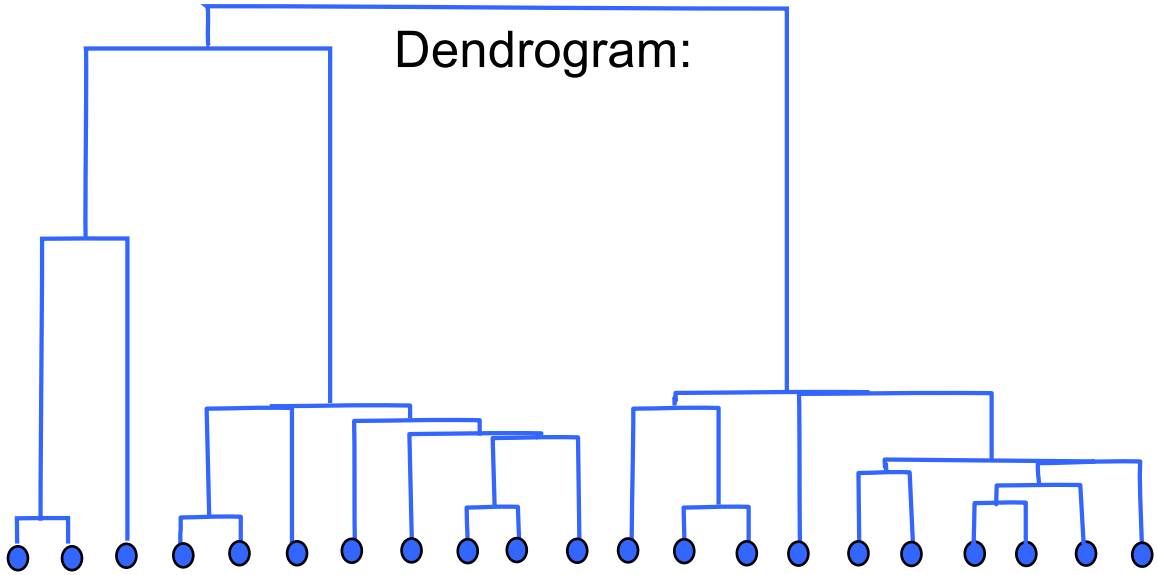
Iteration m-1

Builds up a sequence of clusters (“hierarchical”)

Data:



Dendrogram:

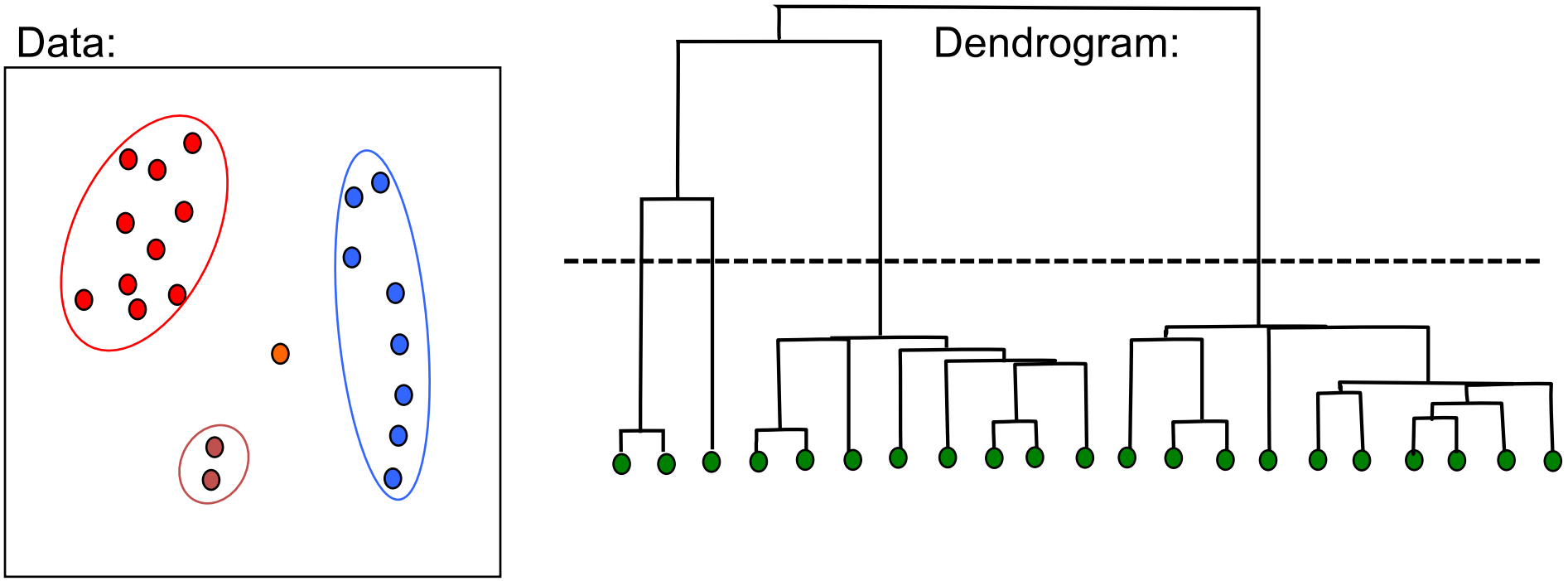


In mltools: “agglomerative”

Algorithmic Complexity: $O(m^2 \log m) + (m-1)*O(m \log m) = O(m^2 \log m)$

From dendrogram to clusters

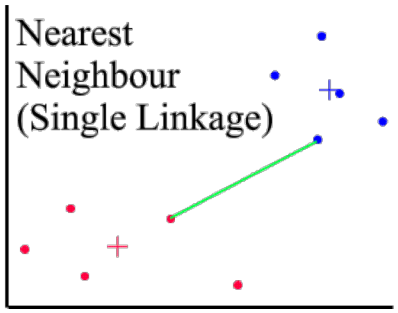
Given the sequence, can select a number of clusters or a dissimilarity threshold:



Algorithmic Complexity: $O(m^2 \log m) + (m-1)*O(m \log m) = O(m^2 \log m)$

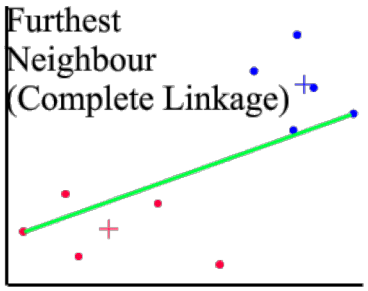
Cluster distances

$$D_{\min}(C_i, C_j) = \min_{x \in C_i,\ y \in C_j} \|x - y\|^2$$



produces minimal spanning tree.

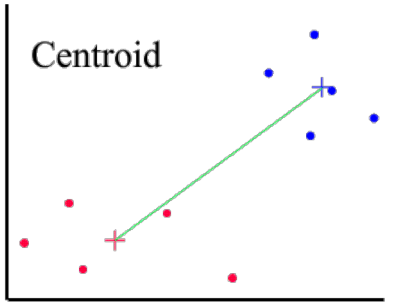
$$D_{\max}(C_i, C_j) = \max_{x \in C_i,\ y \in C_j} \|x - y\|^2$$



avoids elongated clusters.

$$D_{\text{avg}}(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{x \in C_i,\ y \in C_j} \|x - y\|^2$$

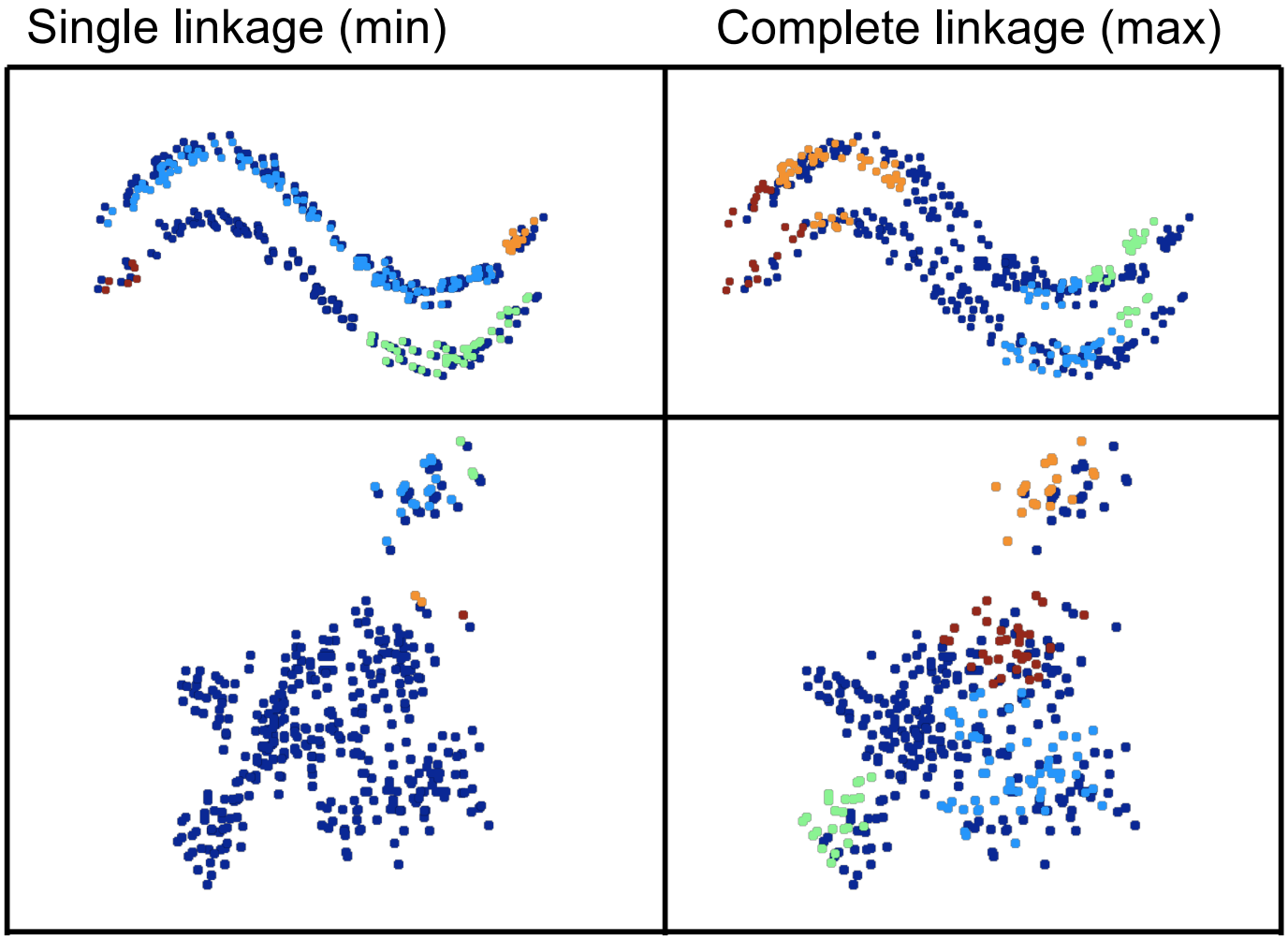
$$D_{\text{means}}(C_i, C_j) = \|\mu_i - \mu_j\|^2$$



Constant time: D(A,C) → D(A+B,C)
D(B,C) →

Cluster distances

- Dissimilarity choice will affect clusters created



Mixtures of Gaussians

- Start with parameters describing each cluster
- Mean μ_c, variance Σ_c, "size" π_c
- Probability distribution:

$$p(x) = \sum_c \pi_c \, \mathcal{N}(x \; ; \; \mu_c, \sigma_c)$$

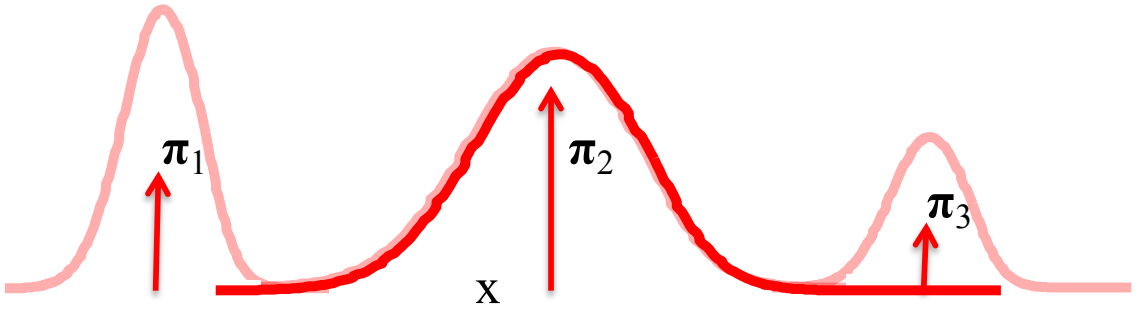
- Equivalent "latent variable" form:

Select a mixture component with probability π $\quad p(z = c) = \pi_c$

Sample from that component's Gaussian $\quad p(x|z = c) = \mathcal{N}(x \; ; \; \mu_c, \sigma_c)$

"Latent assignment" z:
we observe x, but z is *hidden*

p(x) = marginal over x



Mixtures of Gaussians

- Start with parameters describing each cluster
- Mean μ_c, variance Σ_c, "size" π_c
- Probability distribution:

$$p(x) = \sum_c \pi_c \, \mathcal{N}(x \; ; \; \mu_c, \sigma_c)$$

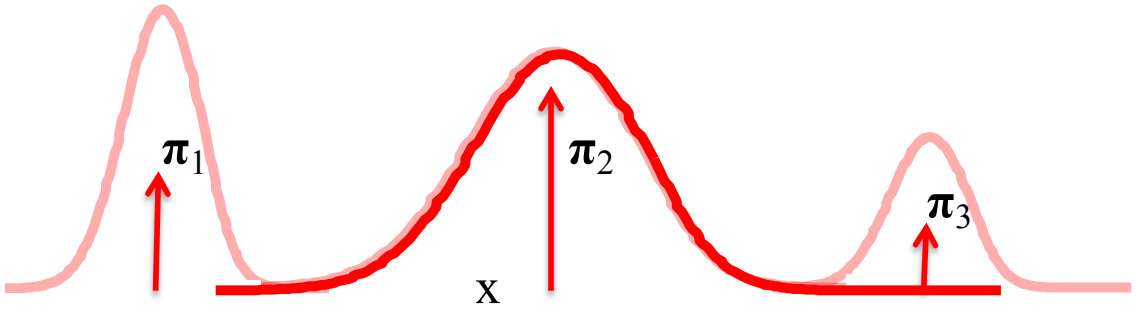
- Equivalent "latent variable" form:

Select a mixture component with probability π $\quad p(z = c) = \pi_c$

Sample from that component's Gaussian $\quad p(x|z = c) = \mathcal{N}(x \; ; \; \mu_c, \sigma_c)$

"Latent assignment" z:
we observe x, but z is *hidden*

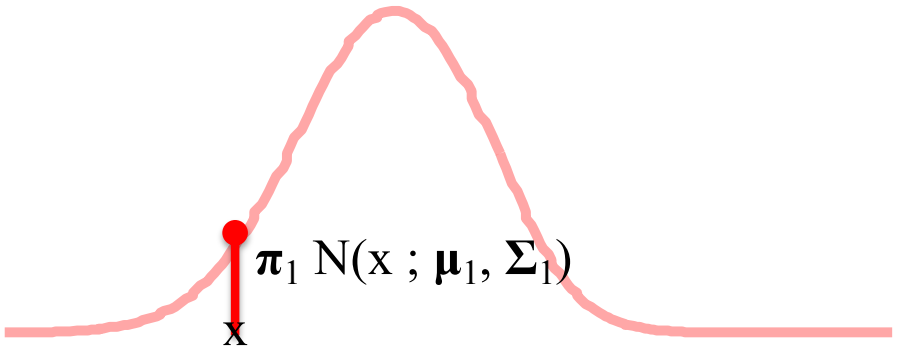
p(x) = marginal over x



EM Algorithm: E-step

- Start with clusters: Mean μ_c, Covariance Σ_c, "size" π_c
- E-step ("Expectation")
  - For each datum (example) x_i,
  - Compute "r_{ic}", the probability that it belongs to cluster c
    - Compute its probability under model c
    - Normalize to sum to one (over clusters c)

$$r_{ic} = \frac{\pi_c \mathcal{N}(x_i \; ; \; \mu_c, \Sigma_c)}{\sum_{c'} \pi_{c'} \mathcal{N}(x_i \; ; \; \mu_{c'}, \Sigma_{c'})}$$



  - If xi is very likely under the c^{th} Gaussian, it gets high weight
  - Denominator just makes r's sum to one

EM Algorithm: M-step

- Start with assignment probabilities r_{ic}
- Update parameters: mean μ_c, Covariance Σ_c, "size" π_c
- M-step ("Maximization")
  - For each cluster (Gaussian) z = c,
  - Update its parameters using the (weighted) data points

$$m_c = \sum_i r_{ic}$$

Total responsibility allocated to cluster c

$$\pi_c = \frac{m_c}{m}$$

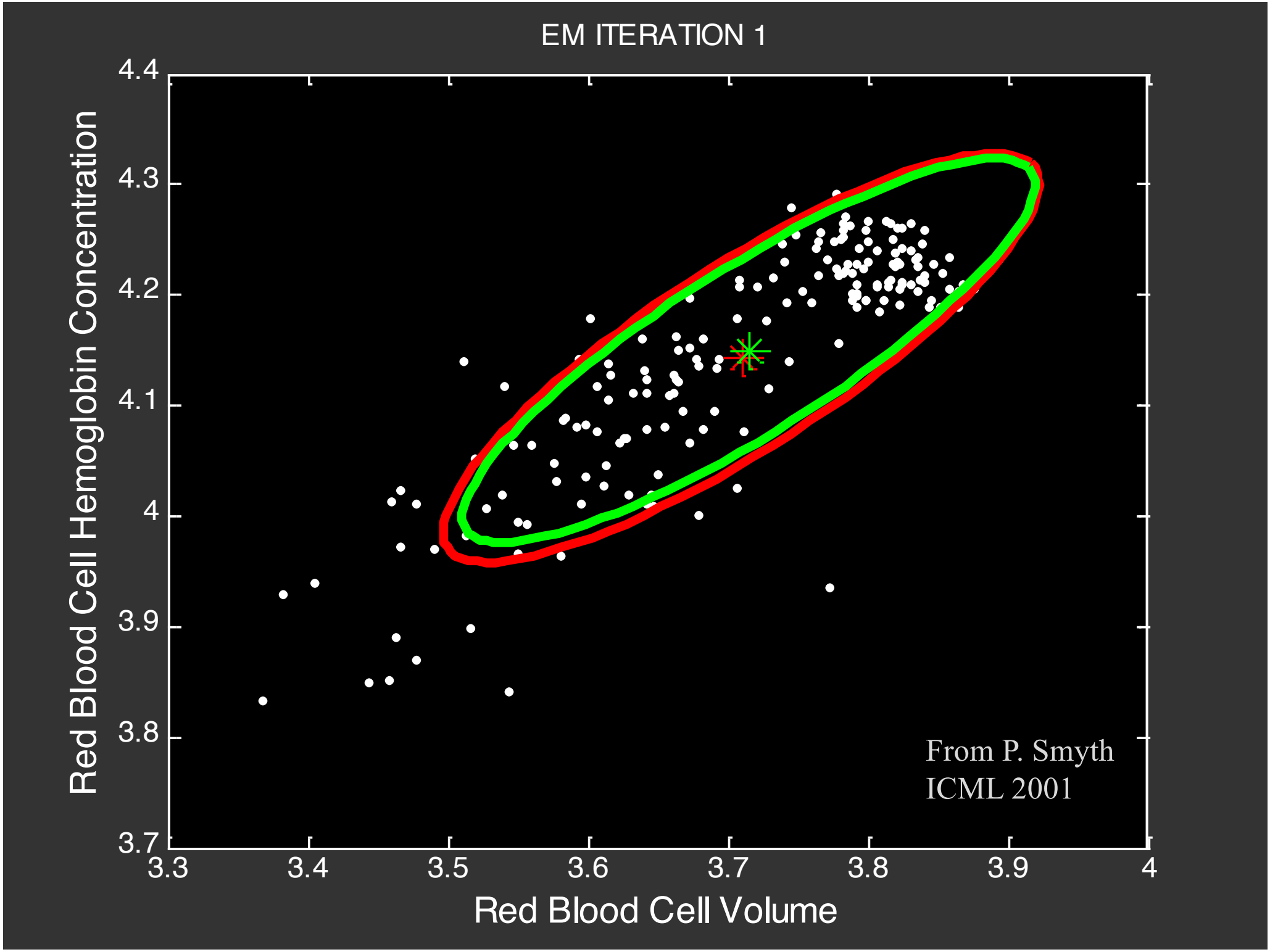
Fraction of total assigned to cluster c

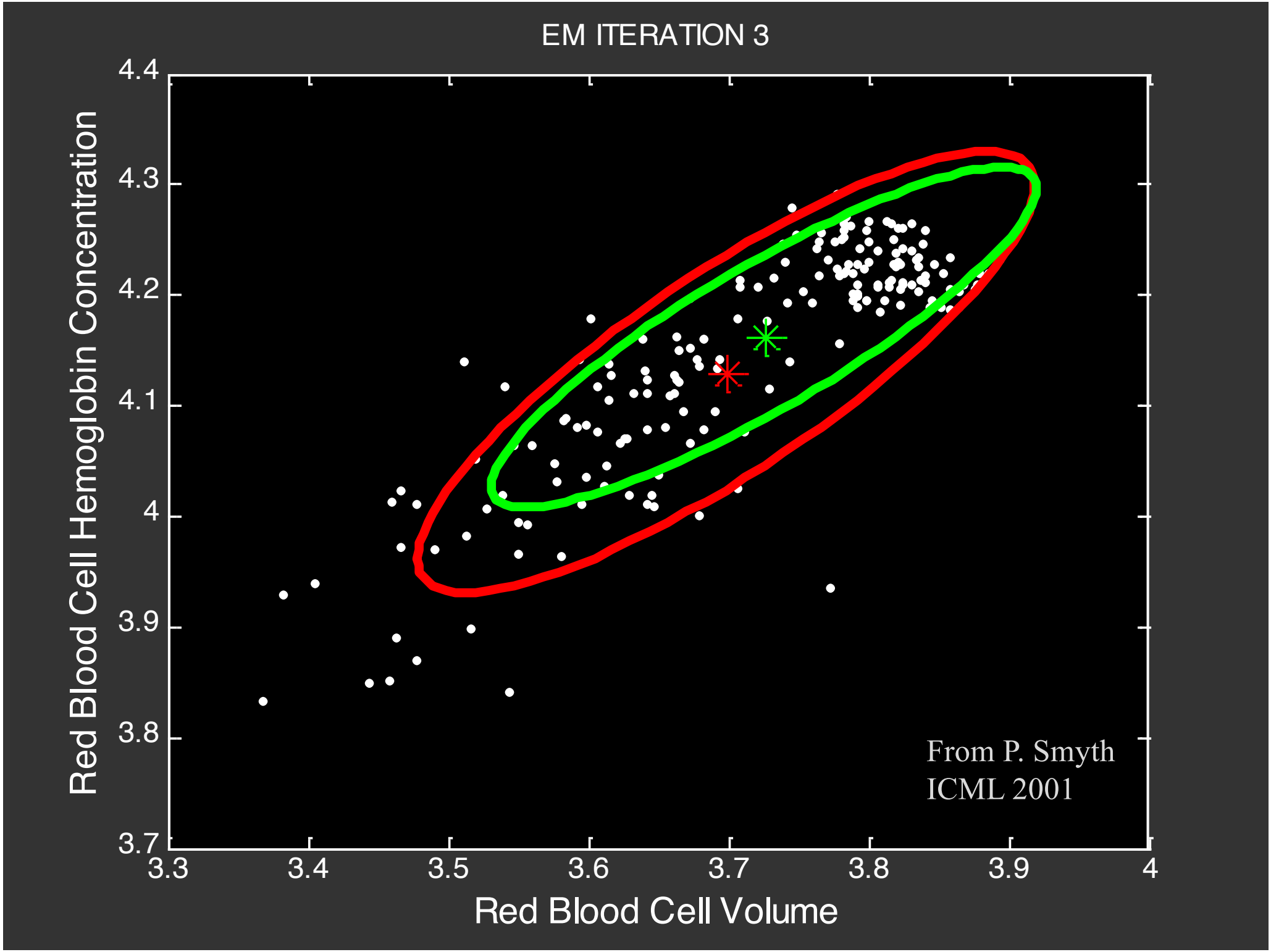
$$\mu_c = \frac{1}{m_c} \sum_i r_{ic} x^{(i)}$$

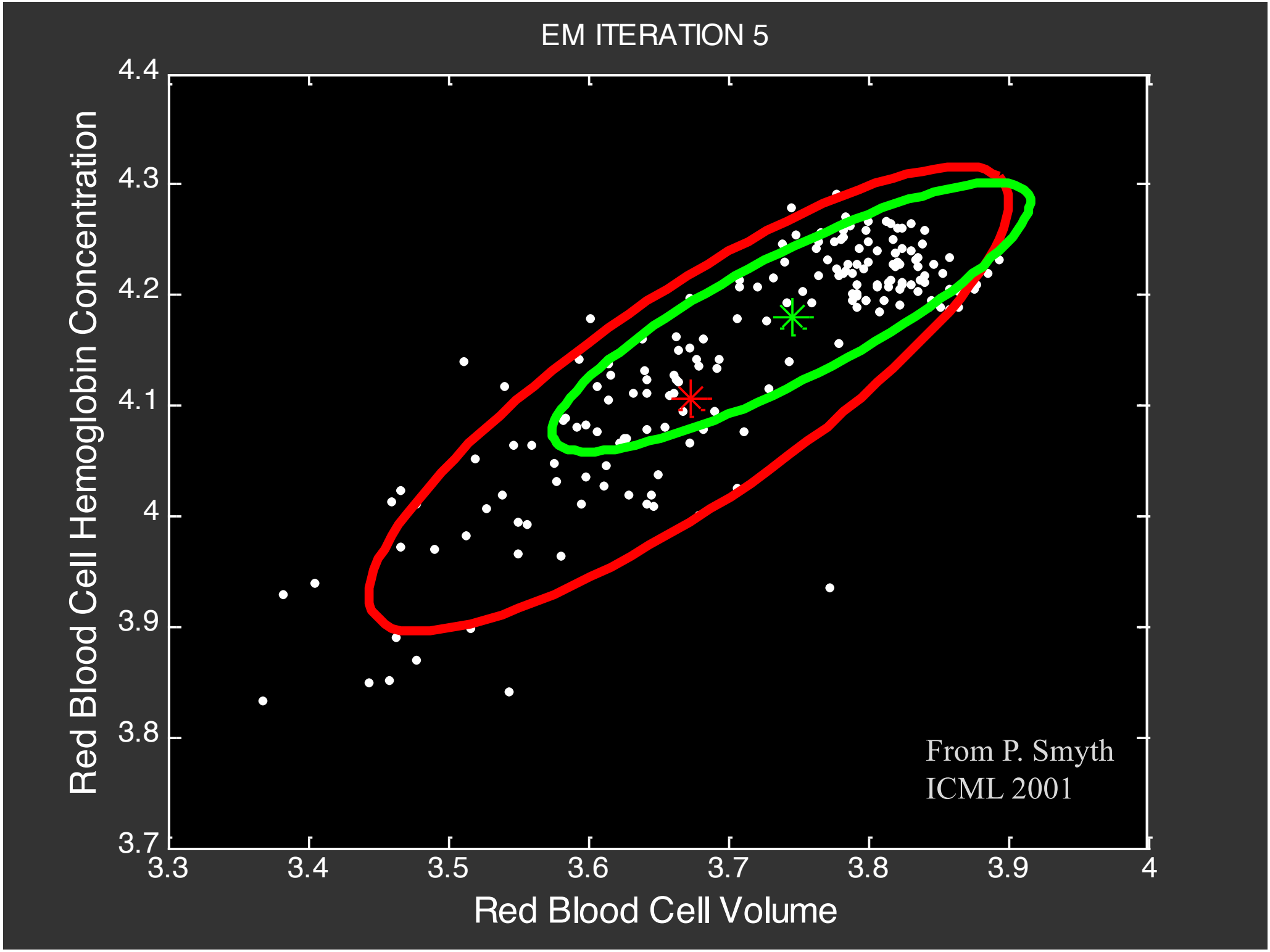
Weighted mean of assigned data

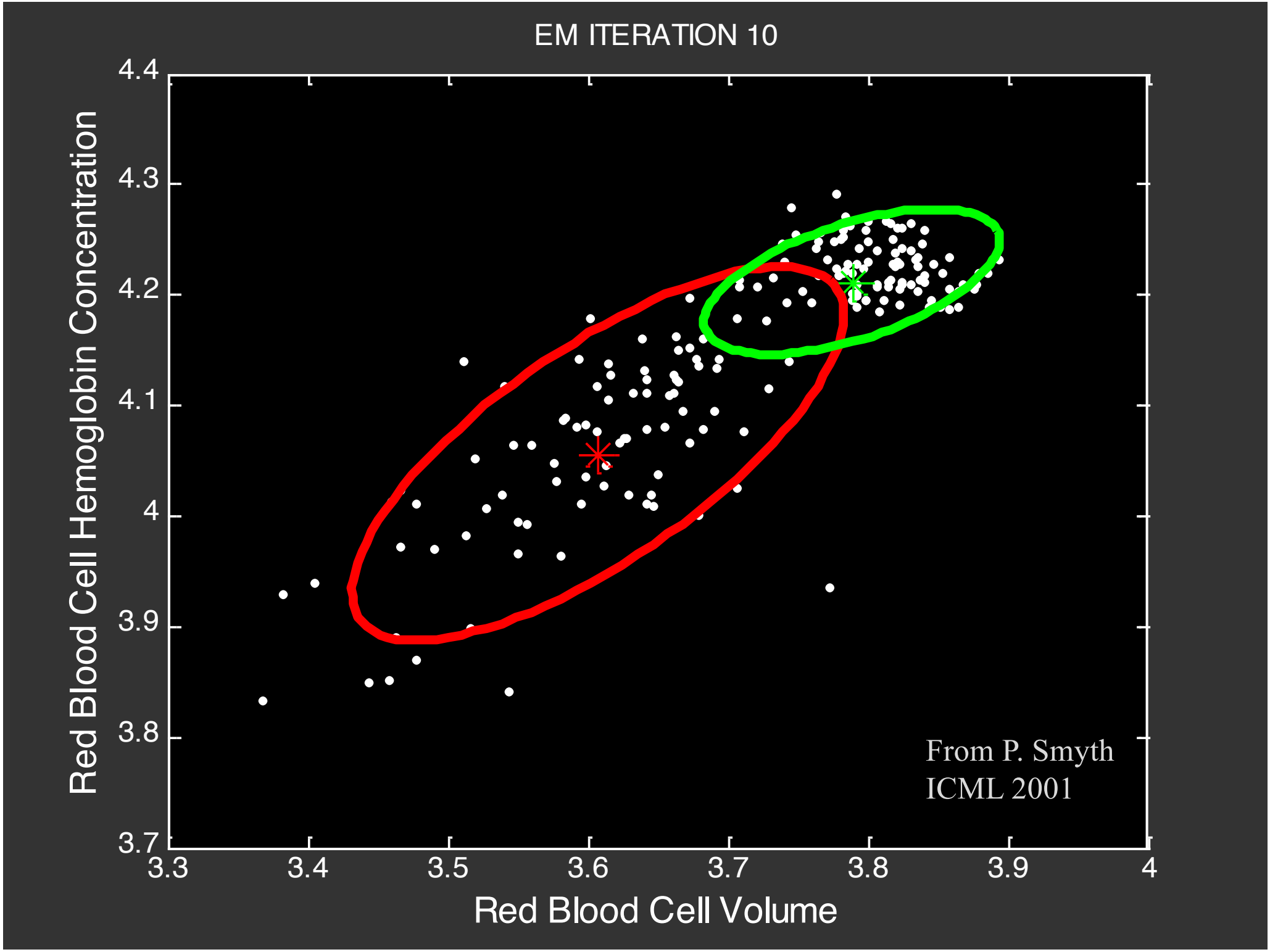
$$\Sigma_c = \frac{1}{m_c} \sum_i r_{ic} (x^{(i)} - \mu_c)^T (x^{(i)} - \mu_c)$$

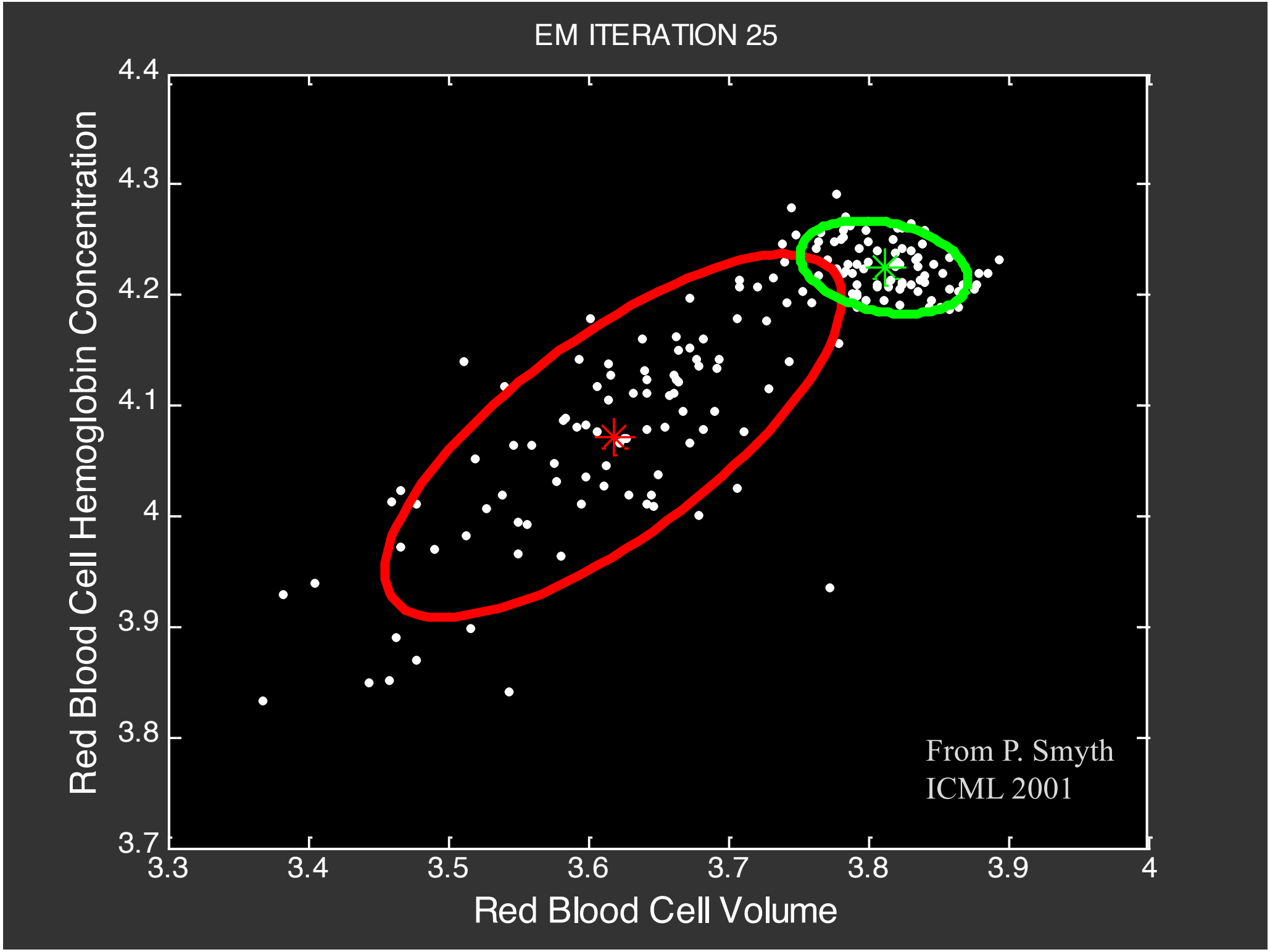
Weighted covariance of assigned data (use new weighted means here)

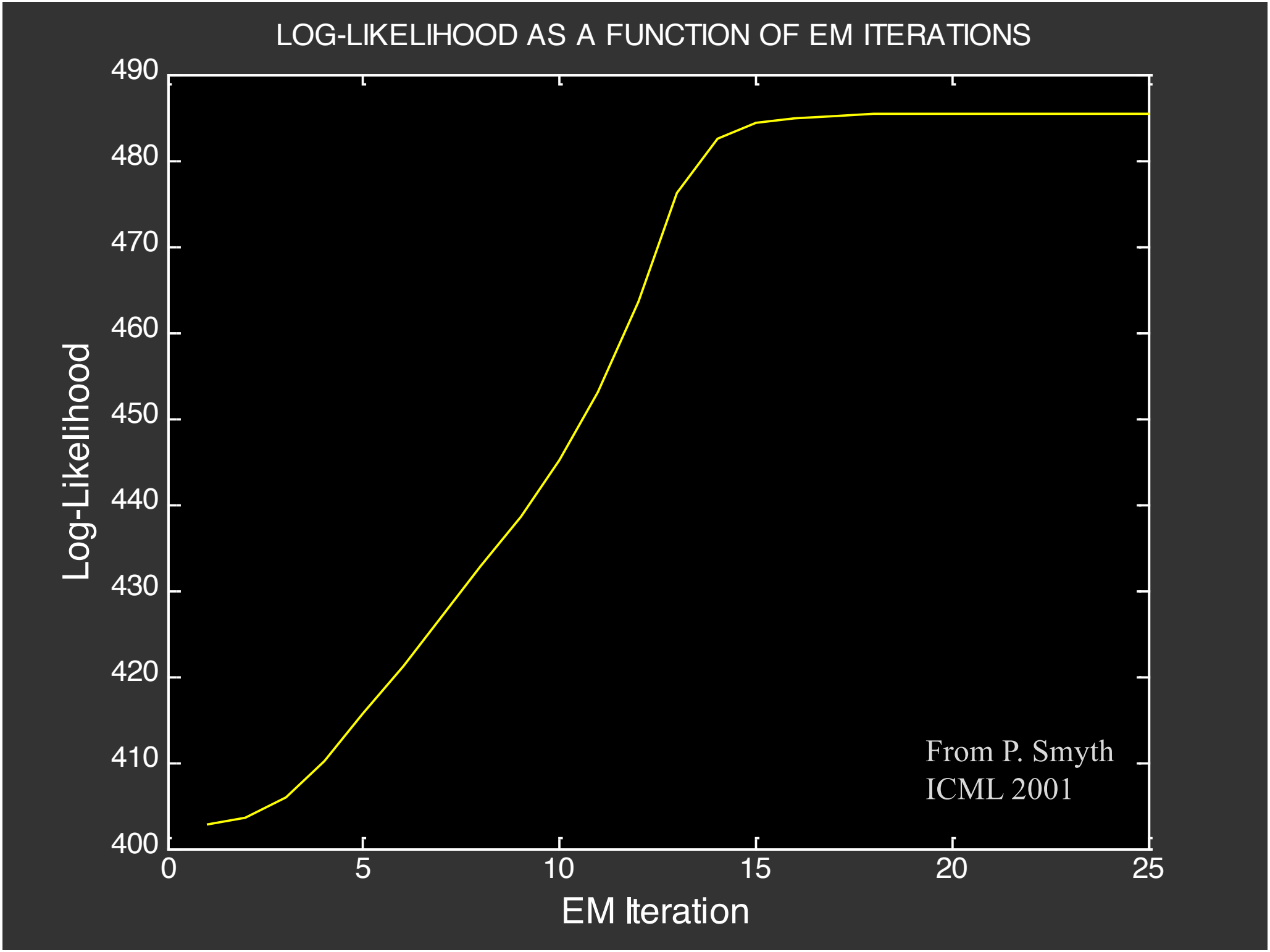

EM ITERATION 1
Red Blood Cell Hemoglobin Concentration
Red Blood Cell Volume
4.4
4.3
4.2
4.1
4
3.9
3.8
3.7
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4
From P. Smyth
ICML 2001


EM ITERATION 3
Red Blood Cell Hemoglobin Concentration
Red Blood Cell Volume
4.4
4.3
4.2
4.1
4
3.9
3.8
3.7
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4
From P. Smyth
ICML 2001


EM ITERATION 5
Red Blood Cell Hemoglobin Concentration
Red Blood Cell Volume
4.4
4.3
4.2
4.1
4
3.9
3.8
3.7
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4
From P. Smyth
ICML 2001


EM ITERATION 10
Red Blood Cell Hemoglobin Concentration
Red Blood Cell Volume
4.4
4.3
4.2
4.1
4
3.9
3.8
3.7
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4
From P. Smyth
ICML 2001


EM ITERATION 25
Red Blood Cell Hemoglobin Concentration
Red Blood Cell Volume
4.4
4.3
4.2
4.1
4
3.9
3.8
3.7
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4
From P. Smyth
ICML 2001


LOG-LIKELIHOOD AS A FUNCTION OF EM ITERATIONS
Log-Likelihood
EM Iteration
490
480
470
460
450
440
430
420
410
400
0
5
10
15
20
25
From P. Smyth
ICML 2001

Machine Learning

Pre-Midterm Topics (supervised learning, see previous review)

VC Dimension

Clustering Methods

Decision Trees

Dimensionality Reduction

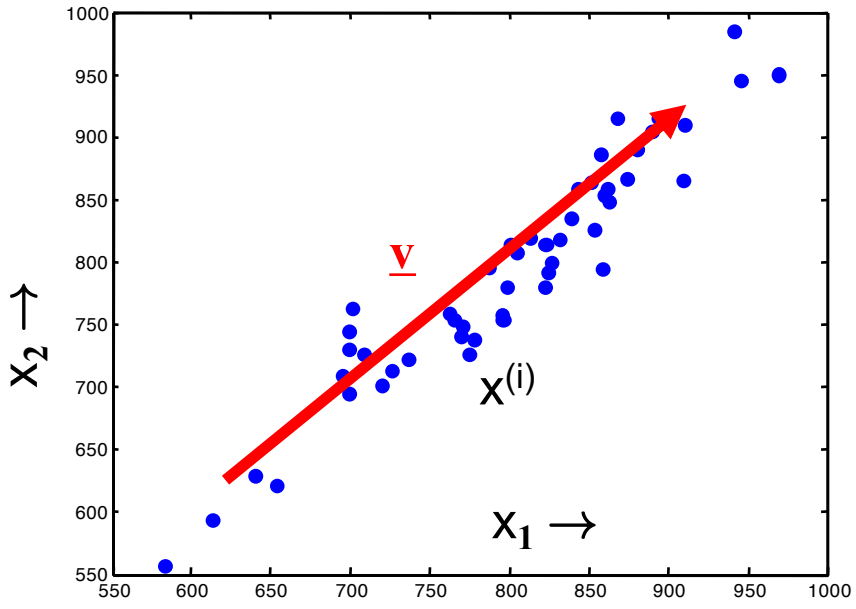
Ensemble Methods

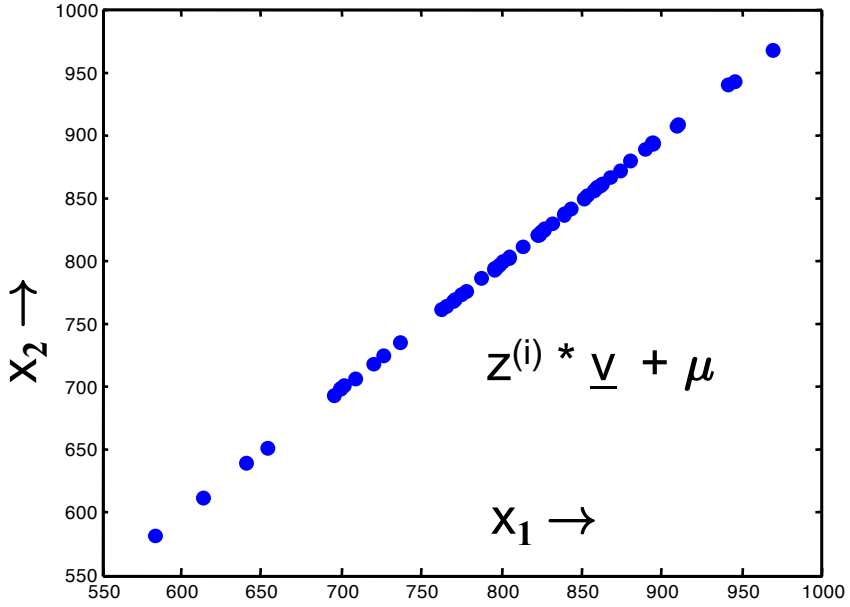
Reinforcement Learning

Dimensionality reduction

- Ex: data with two real values $[x_1,x_2]$
- We'd like to describe each point using only one value $[z_1]$
- We'll communicate a "model" to convert: $[x_1,x_2] \sim f(z_1)$

- Ex: linear function f(z): $[x_1,x_2] = \mu + z * \underline{v} = \mu + z * [v_1,v_2]$
- μ, $\underline{v}$ are the same for all data points (communicate once)
- z tells us the closest point on v to the original point $[x_1,x_2]$

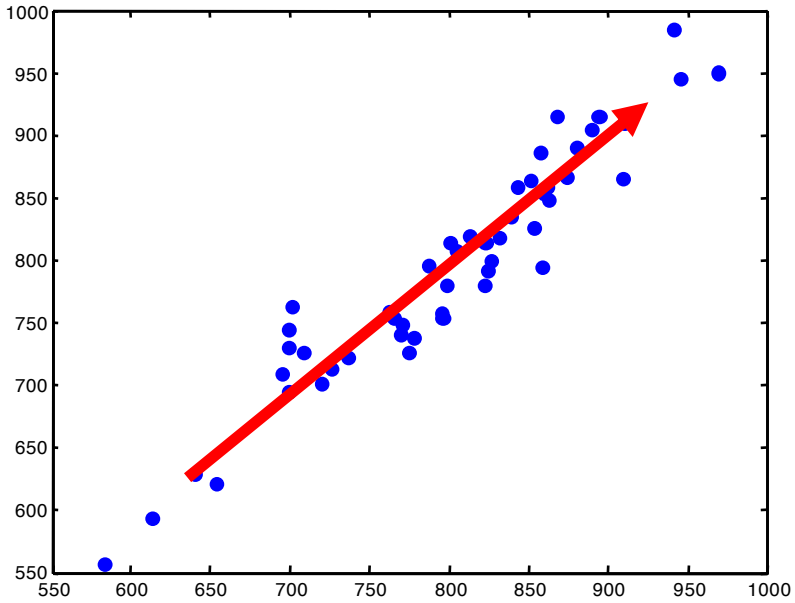




Principal Components Analysis

- How should we find v?
  - Assume X is zero mean, or $\tilde{X} = X - \mu$
  - Find "v" as the direction of maximum "spread" (variance)
  - Solution is the eigenvector with largest eigenvalue

  - Equivalent: v also leaves the smallest residual variance! ("error")



Project X to v: $z = \tilde{X} \cdot v$

Variance of projected points:

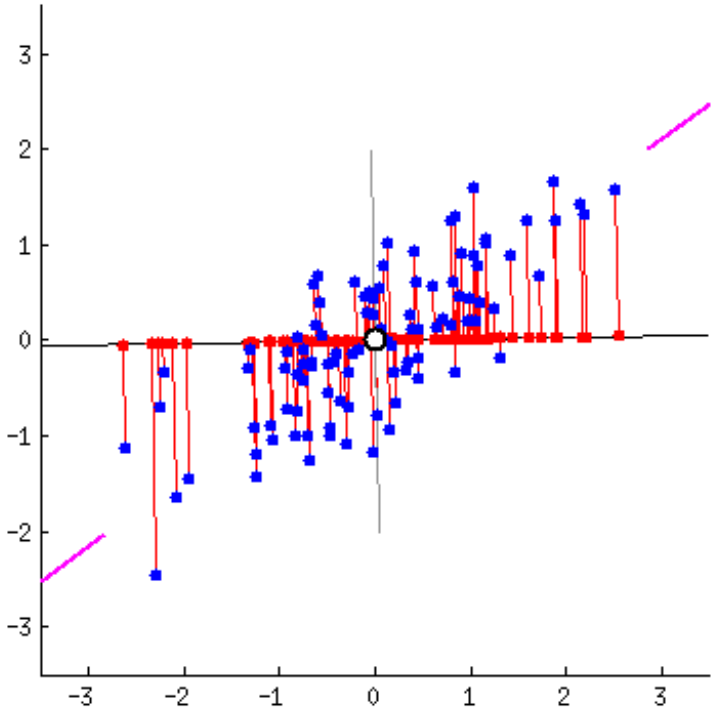
$$\sum_i (z^{(i)})^2 = z^T z = v^T \tilde{X}^T \tilde{X} v$$

Best "direction" v:

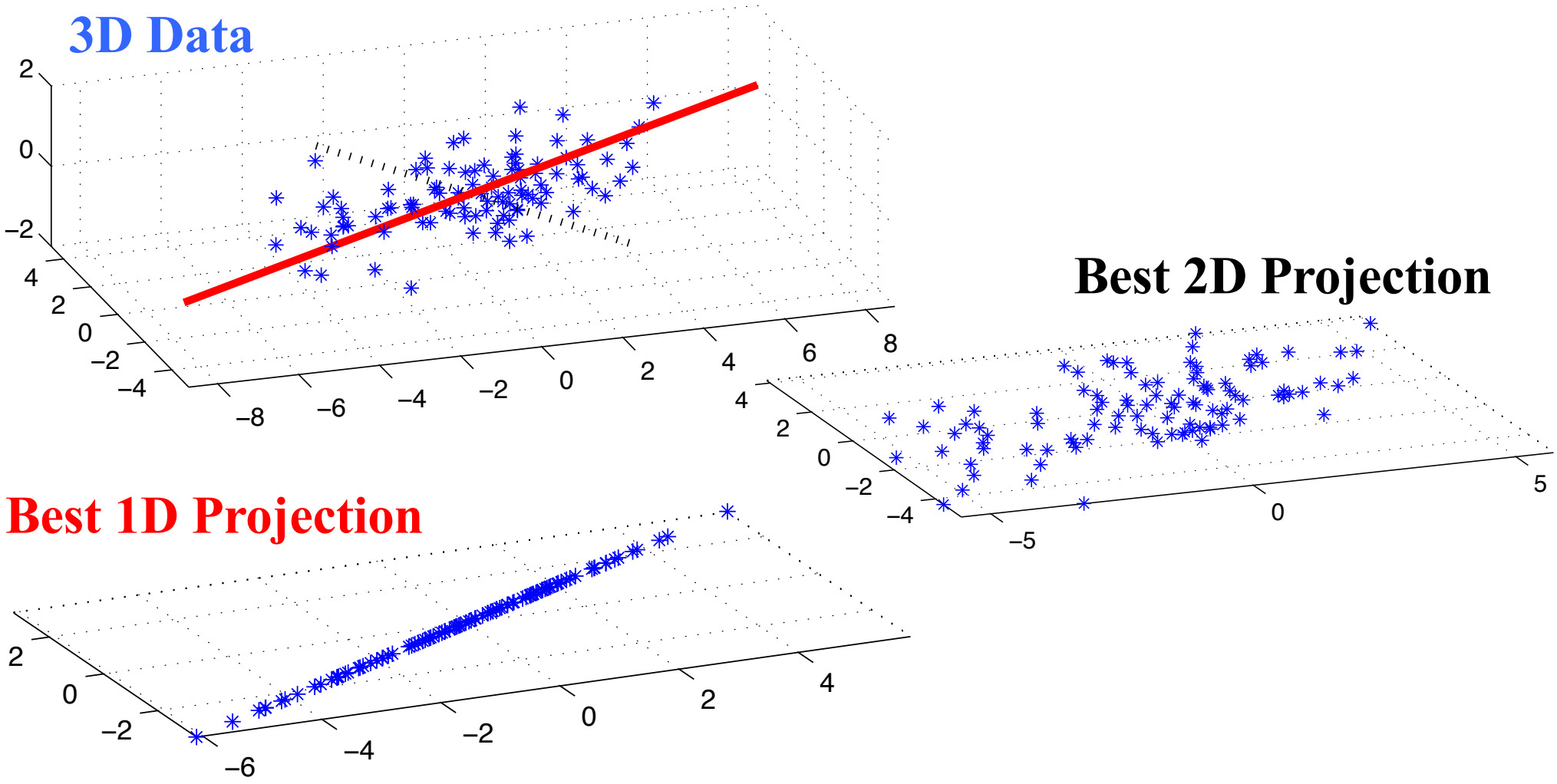
$$\max_v \; v^T \tilde{X}^T \tilde{X} v \quad s.t. \; ||v|| = 1$$

$\Rightarrow$ largest eigenvector of $X^T X$

Principal Components Analysis



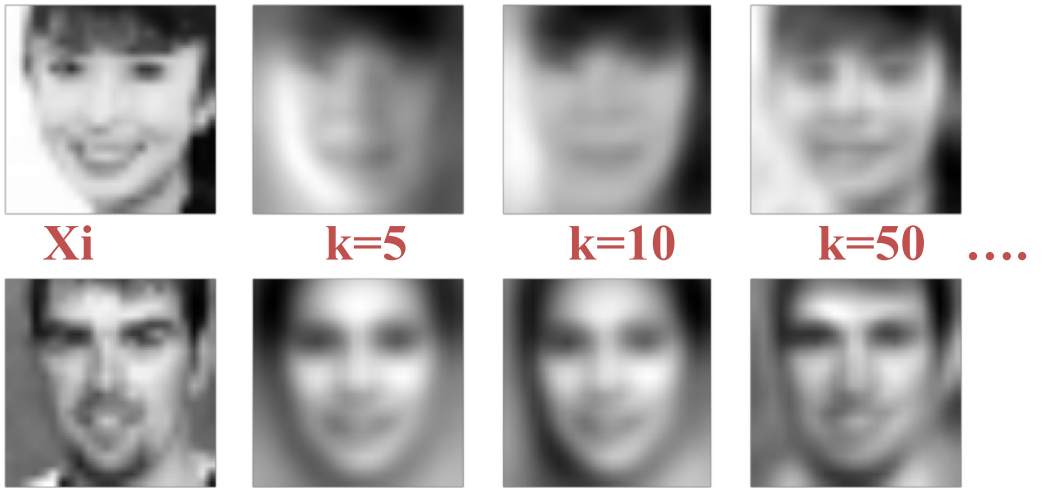
Principal Components Analysis (PCA)



“Eigen-faces”

- “Eigen-X” = represent X using PCA
- Ex: Viola Jones data set
  - 24x24 images of faces = 576 dimensional measurements
  - Take first K PCA components

Projecting data onto first k dimensions



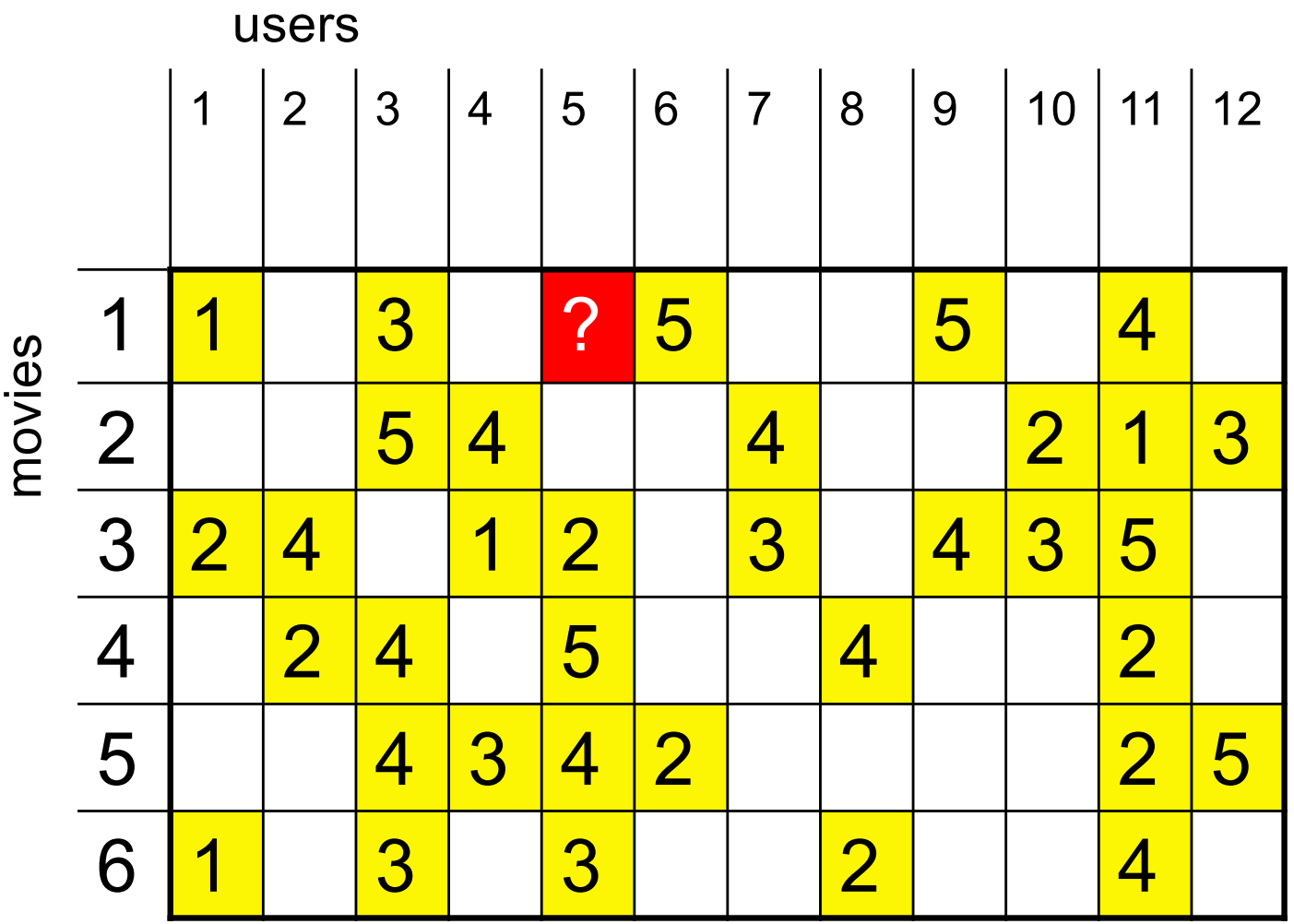
“Eigen-faces”

- “Eigen-X” = represent X using PCA
- Ex: Viola Jones data set
  - 24x24 images of faces = 576 dimensional measurements
  - Take first K PCA components

Embedding data using first K=2 PCA components

Collaborative filtering (Netflix)

From Y. Koren of BellKor team



Latent space models

From Y. Koren of BellKor team

- Model ratings matrix as combination of user and movie factors
- Infer values from known ratings
- Extrapolate to unranked

users

items

1		3			5			5		4	
		5	4			4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

~

items

.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

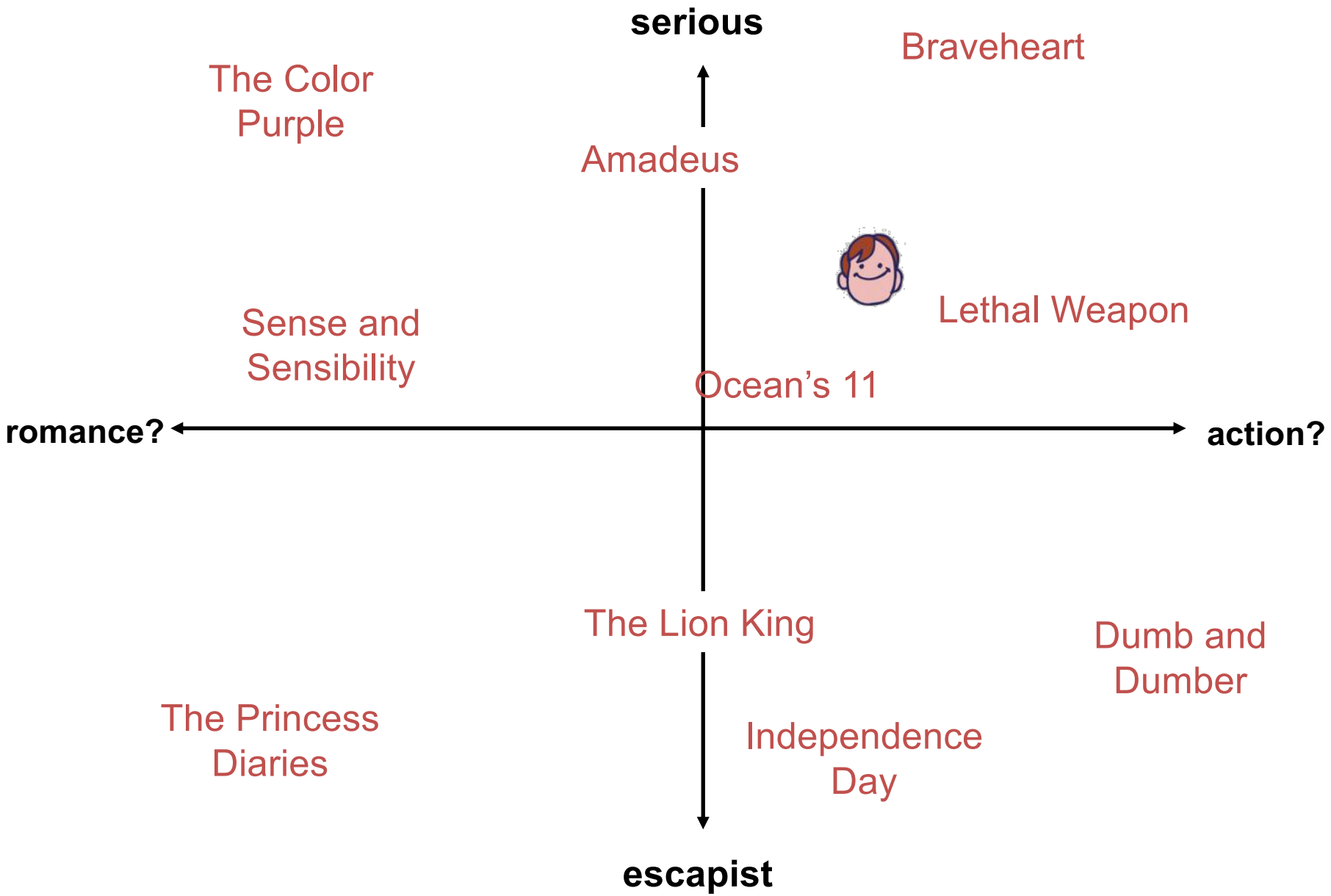
•

users

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1

Latent space models

From Y. Koren of BellKor team



Machine Learning

Pre-Midterm Topics (supervised learning, see previous review)

VC Dimension

Clustering Methods

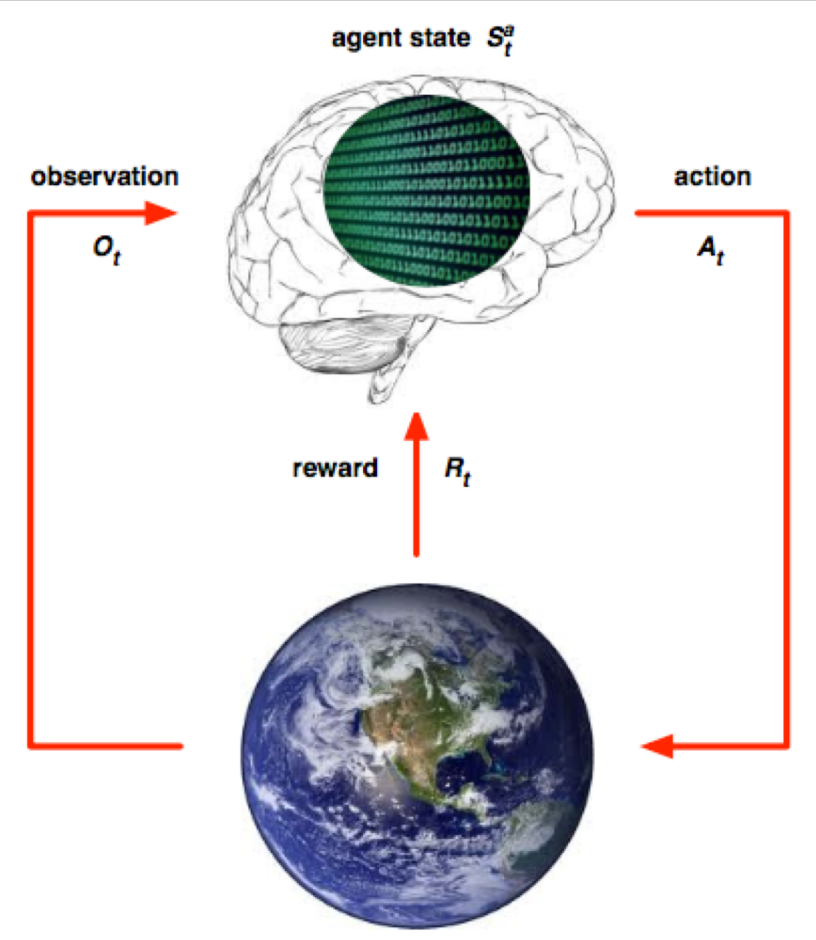
Decision Trees

Dimensionality Reduction

Ensemble Methods

Reinforcement Learning

Agent State, S_t



History: everything that happened so far

$$H_t = O_1R_1A_1O_2R_2A_2O_3R_3,\ldots,A_{t-1}O_tR_t$$

State, S_t can be

- O_t
- O_tR_t
- $A_{t-1}O_tR_t$
- $O_{t-3}O_{t-2}O_{t-1}O_t$

In general, $S_t = f(H_t)$

You, as AI designer, specify this function

State Transition Matrix

For a Markov state s and successor state s', the *state transition probability* is defined by

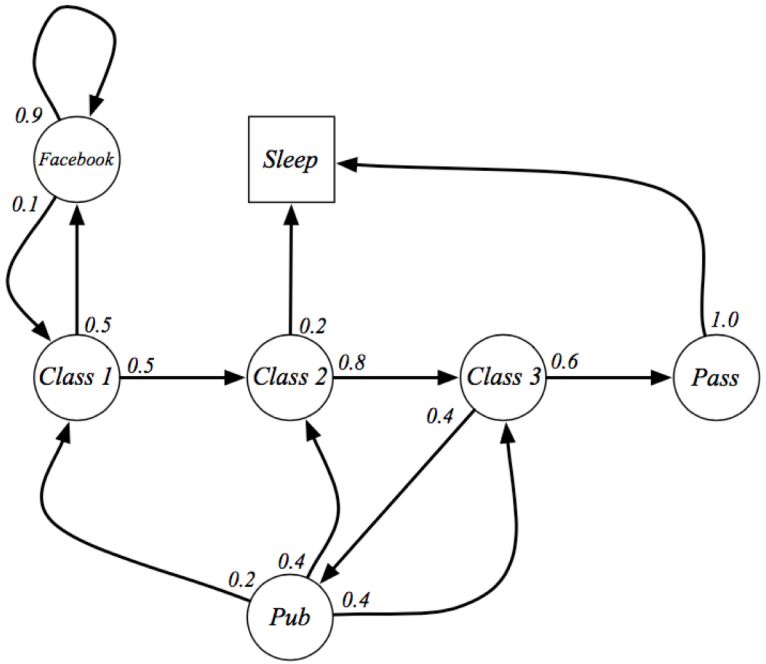
$$\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$$

State transition matrix $\mathcal{P}$ defines transition probabilities from all states s to all successor states s',

$$\mathcal{P} \quad = \textit{from} \begin{array}{c} \textit{to} \\ \begin{bmatrix} \mathcal{P}_{11} & \cdots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \cdots & \mathcal{P}_{nn} \end{bmatrix} \end{array}$$

where each row of the matrix sums to 1.

Student MC: Episodes

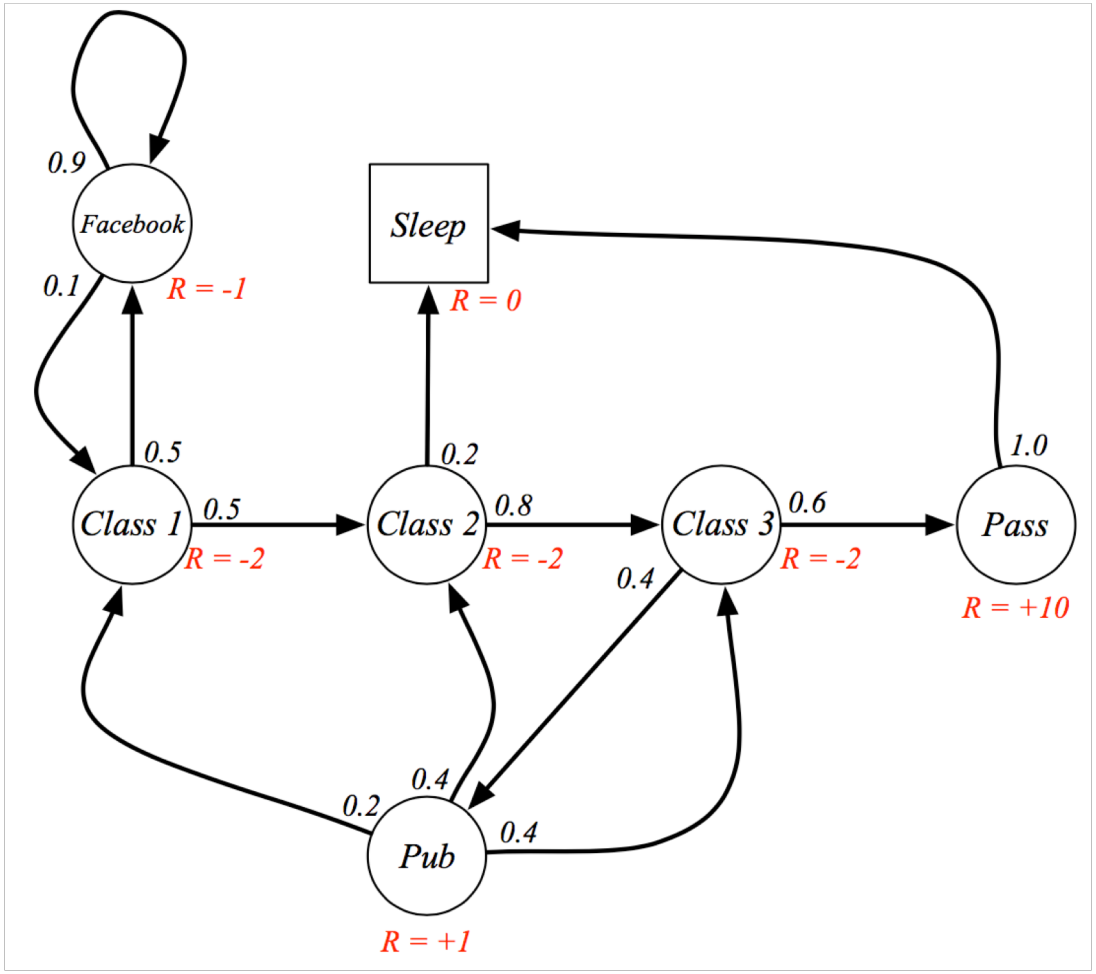


Sample episodes for Student Markov Chain starting from $S_1 = \text{C1}$

$$S_1, S_2, ..., S_T$$

- C1 C2 C3 Pass Sleep
- C1 FB FB C1 C2 Sleep
- C1 C2 C3 Pub C2 C3 Pass Sleep
- C1 FB FB C1 C2 C3 Pub C1 FB FB FB C1 C2 C3 Pub C2 Sleep

The Student MRP



Return

Definition

The *return* G_t is the total discounted reward from time-step t.

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- The *discount* $\gamma \in [0, 1]$ is the present value of future rewards
- The value of receiving reward R after $k + 1$ time-steps is $\gamma^k R$.
- This values immediate reward above delayed reward.
  - γ close to 0 leads to "myopic" evaluation
  - γ close to 1 leads to "far-sighted" evaluation

Value Function

The value function $v(s)$ gives the long-term value of state s

Definition

The *state value function* $v(s)$ of an MRP is the expected return starting from state s

$$v(s) = \mathbb{E}[G_t \mid S_t = s]$$

Matrix Form of Bellman Eq

The Bellman equation can be expressed concisely using matrices,

$$v = \mathcal{R} + \gamma \mathcal{P} v$$

where v is a column vector with one entry per state

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{11} & \dots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

Solving the Bellman Equation

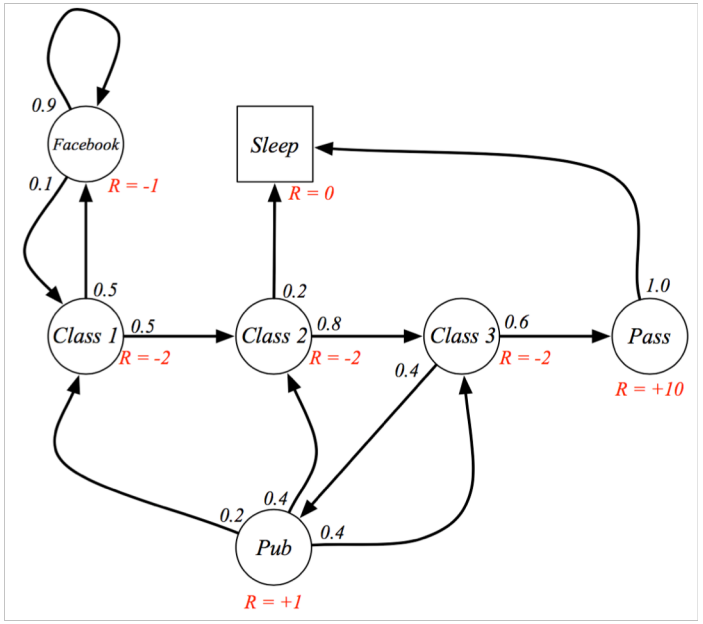
- The Bellman equation is a linear equation
- It can be solved directly:

$$
\begin{aligned}
v &= \mathcal{R} + \gamma \mathcal{P} v \\
(I - \gamma \mathcal{P})\, v &= \mathcal{R} \\
v &= (I - \gamma \mathcal{P})^{-1} \mathcal{R}
\end{aligned}
$$

- Computational complexity is $O(n^3)$ for n states
- Direct solution only possible for small MRPs
- There are many iterative methods for large MRPs, e.g.
  - Dynamic programming
  - Monte-Carlo evaluation
  - Temporal-Difference learning

Dynamic Programming

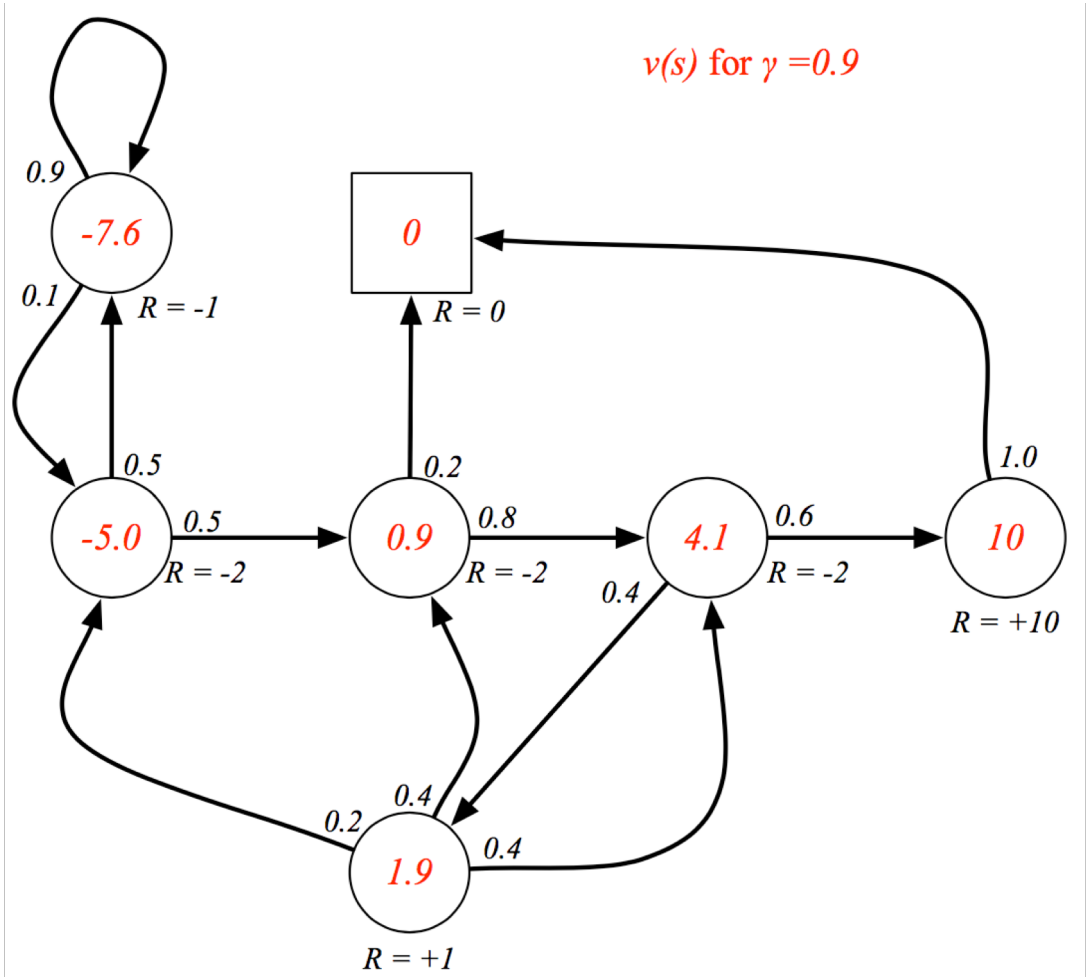
- $v^2(C1) = -2 + \gamma (.5\, v^1(FB) + .5\, v^1(C2))$
- $v^2(FB) = -1 + \gamma (.9\, v^1(FB) + .1\, v^1(C1))$
- …
- $v^3(FB) = -1 + \gamma (.9\, v^2(FB) + .1\, v^2(C1))$
- …



γ=0.5

	C1	C2	C3	Pa	Pub	FB	Slp
t=1	-2	-2	-2	10	1	-1	0
t=2	-2.75	-2.8	1.2	10	0	-1.55	0
t=3	-3.09	-1.52	1	10	0.41	-1.83	0
t=4	-2.84	-1.6	1.08	10	0.59	-1.98	0

Student MRP: Value Function



Markov Decision Process

A Markov decision process (MDP) is a Markov reward process with decisions. It is an *environment* in which all states are Markov.

Definition

A *Markov Decision Process* is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}^{a}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}^{a}_{s} = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$
- γ is a discount factor $\gamma \in [0, 1]$.

Policies

Definition

A *policy* π is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P}\left[A_t = a \mid S_t = s\right]$$

- A policy fully defines the behaviour of an agent
- MDP policies depend on the current state (not the history)

Value Function

Definition

The *state-value function* $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π

$$v_\pi(s) = \mathbb{E}_\pi [G_t \mid S_t = s]$$

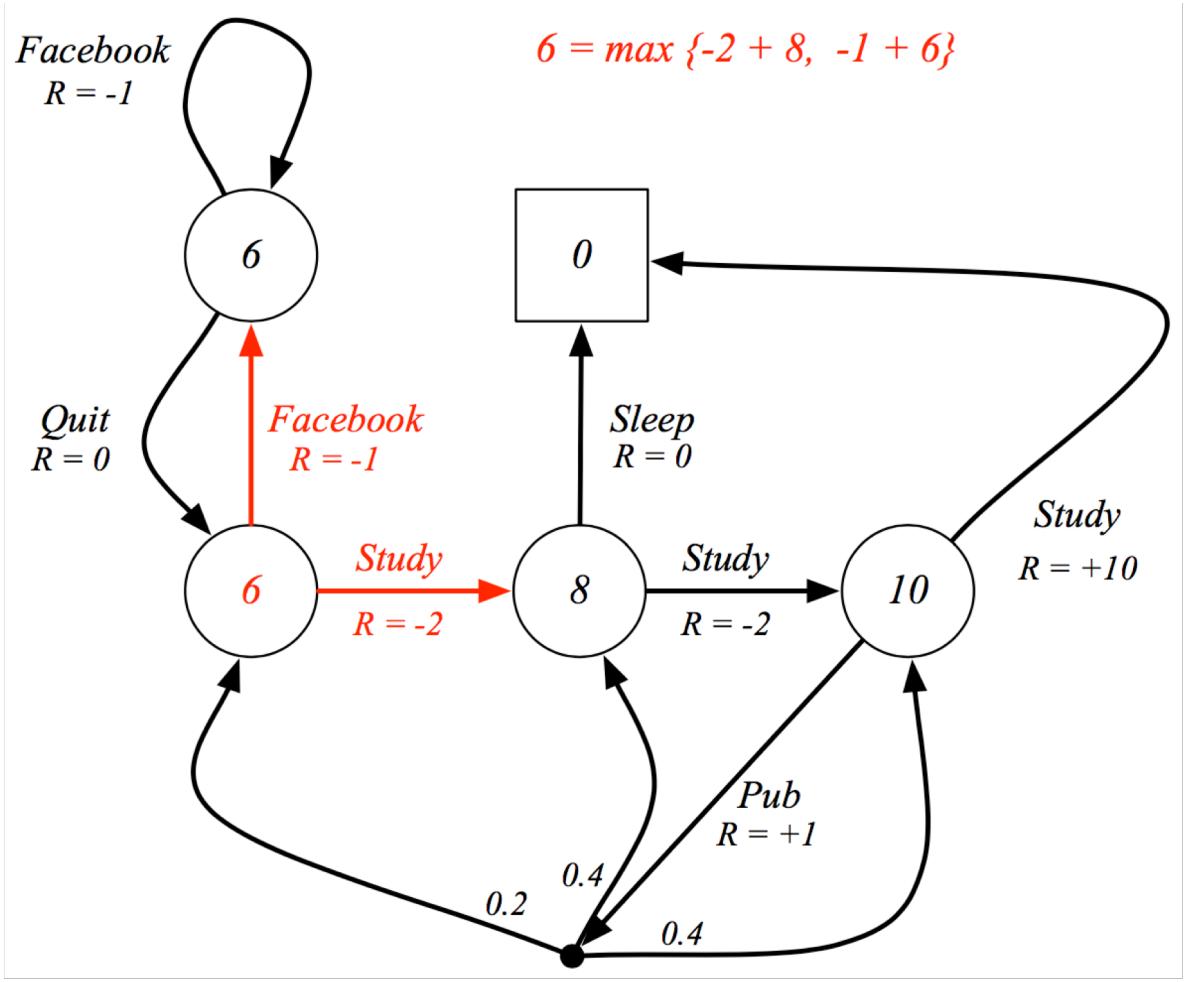
Optimal Value Function

Definition

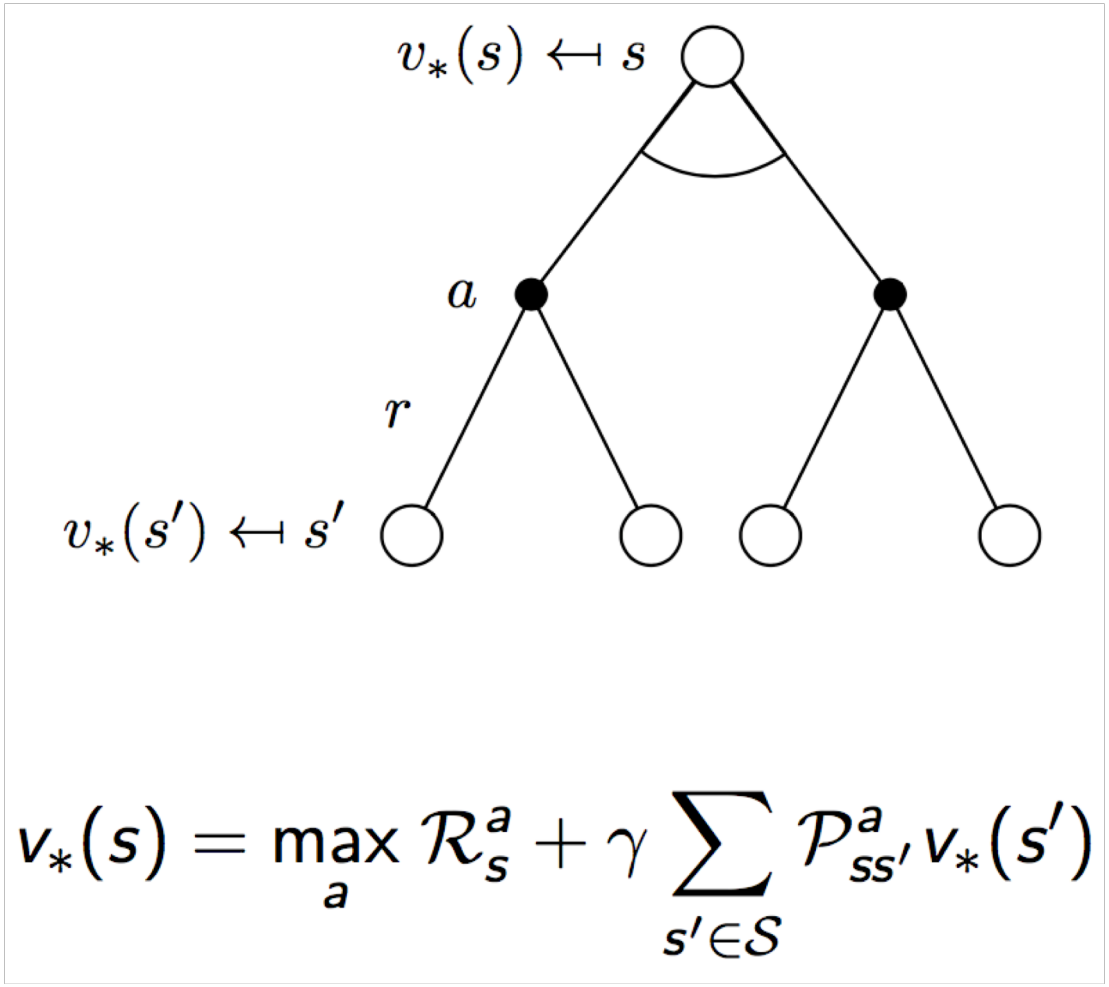
The *optimal state-value function* $v_*(s)$ is the maximum value function over all policies

$$v_*(s) = \max_{\pi} v_\pi(s)$$

Student MDP: Bellman Optimality



Bellman Optimality Eq, V



$$v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

Improving a Policy!

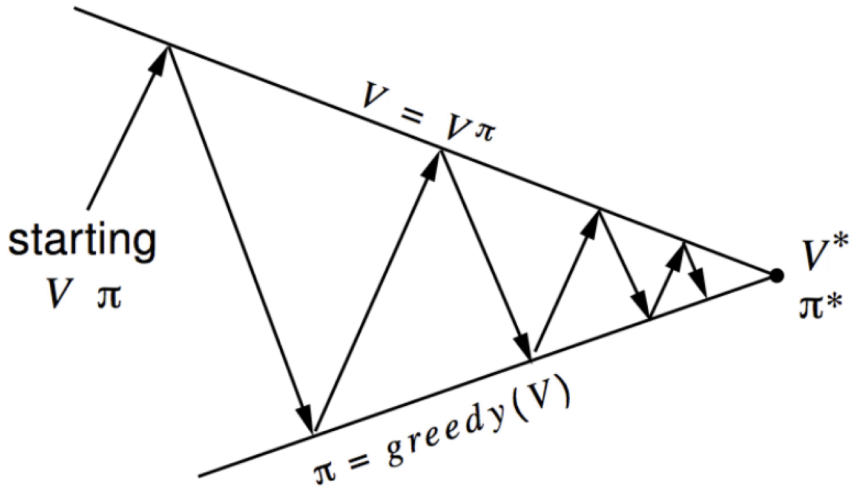
- Given a policy π
  - Evaluate the policy π

$$v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + ... | S_t = s\right]$$

  - Improve the policy by acting greedily with respect to v_π

$$\pi' = \text{greedy}(v_\pi)$$

Policy Iteration



Policy evaluation Estimate v_π

Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$

Greedy policy improvement