Algorithmic Human-Robot Interaction

Modeling with GMMs and HMMs for Activity Recognition

CSCI 7000

Prof. Brad Hayes

University of Colorado Boulder

Last Time…

Papers for Thursday 2/28:
Natural Language Understanding

Robust Robot Learning from Demonstration and Skill Repair Using Conceptual Constraints – Mueller et al.
Pro: Jack Kawell
Con: Matthew Luebbers

Accurately and Efficiently Interpreting Human-Robot Instructions of Varying Granularities – Arumugam et al.
Pro: Karthik Palavalli
Con: Ian Loefgren

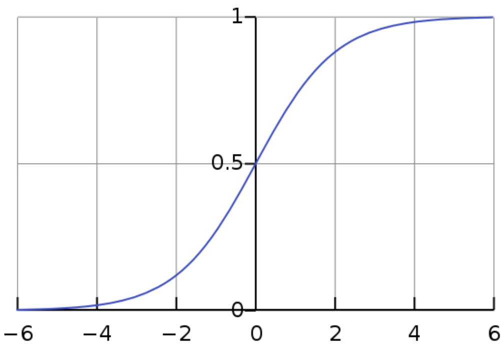
Introduction to Machine Learning

- Regression: How much is this house worth?
- Classification: Is this a photo of a dog or ice cream?

Linear Regression to Logistic Regression

- Linear Regression gives us a continuous-valued function approximation
  - Models relationship between scalar dependent variable y and one or more variables X
- Logistic regression allows us to approximate **categorical** data
  - Pick a model function that squashes values between 0 and 1

$$F(x) = \frac{1}{1 + e^{-x}}$$



  - Apply it to a familiar function: $g(X) = \beta_0 + \beta_1 x + \epsilon$

$$P(Y = 1) = F(g(x)) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 * x)}}$$

Training and Testing Your Algorithms

- Partition your data into TRAIN, VALIDATE, and TEST
- Train your model on TRAIN
- Evaluate your model on VALIDATE
- TRAIN and VALIDATE can be swapped around
- You only get to run on TEST once, ever!

Types of Learning

Supervised Learning

- Given a dataset of (X, y) pairs
- X: Vector representing a data point
- y: Label or value that is the correct answer for $f(X) = y$

"You guess, I tell you the correct answer, you update, repeat"

Reinforcement Learning

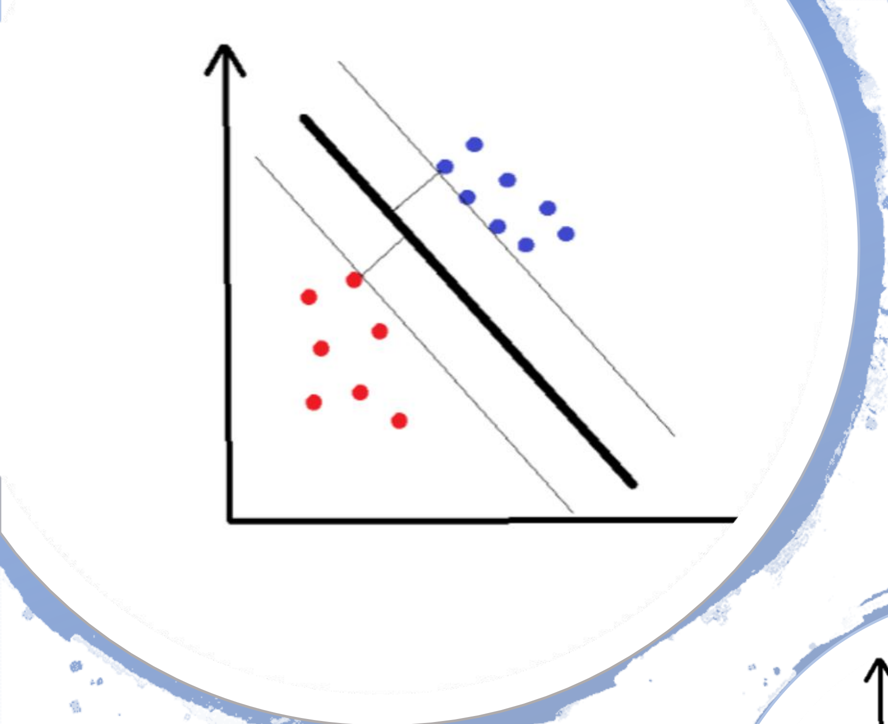
- Given some reward function R
- $R(s, a, s')$ gives the value of moving from state s to s' via action a

"You guess, I tell you if you're on the right track (sometimes)"

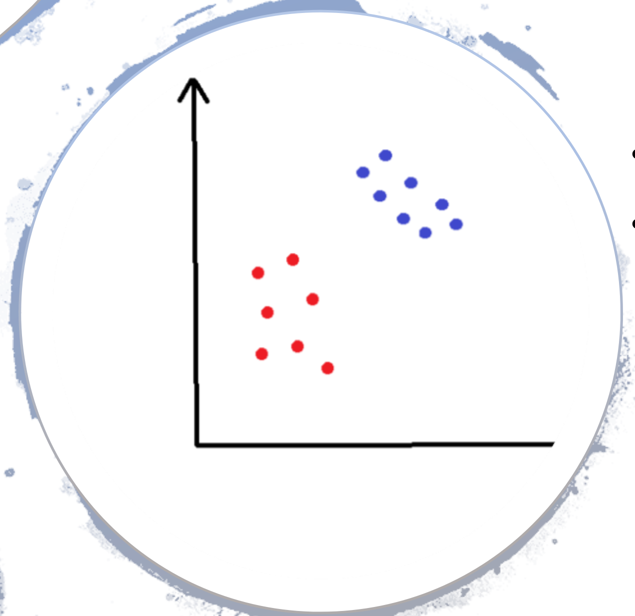
Unsupervised Learning

- Given a dataset of X's
- X: Vector representing a data point
- No labels (answers) given!

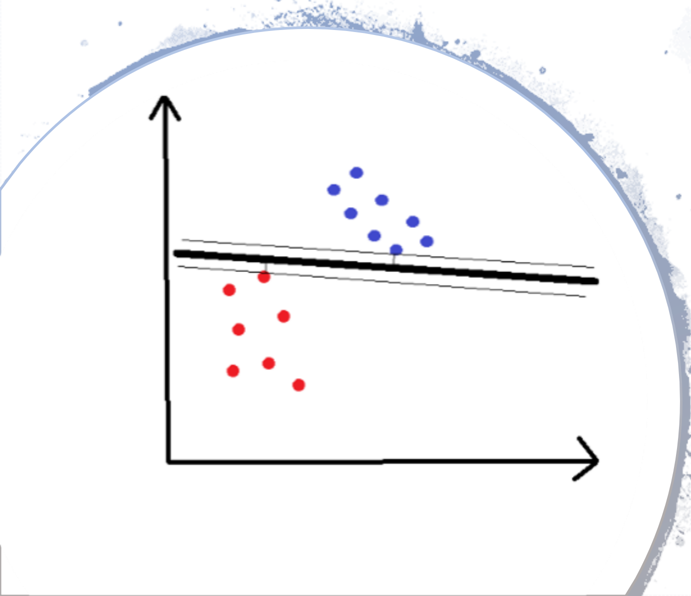
"You guess, I have no feedback to give you. Good luck!"



Supervised Learning: Support Vector Machines



- Is this dot red or blue?
- Classifier will draw a separating line (or hyperplane)

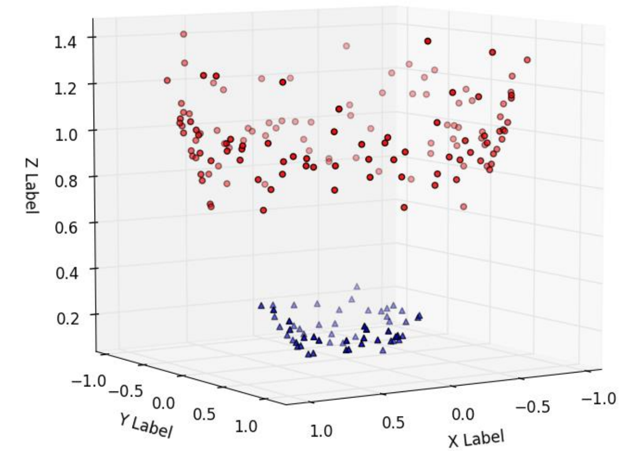


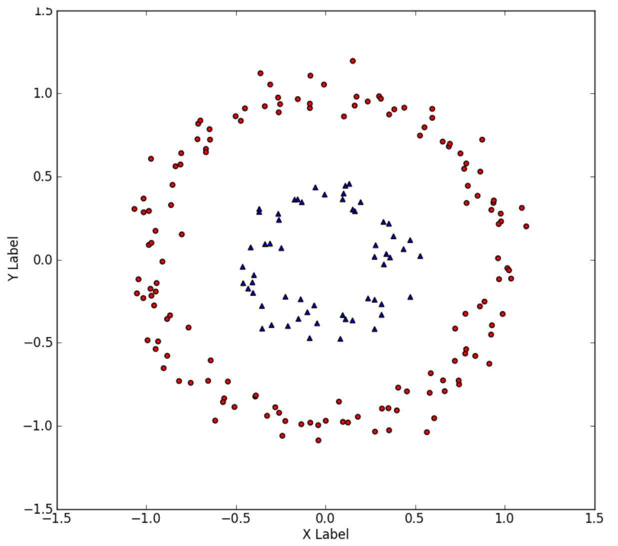
Supervised Learning: Support Vector Machines

What happens if the data doesn't separate cleanly?

- Add a cost for misclassified examples and let the optimizer take care of it!

- Add more dimensions to the data!
  - $x^2, y^2, x^2 + y^2, \cos(x), xy$, etc.





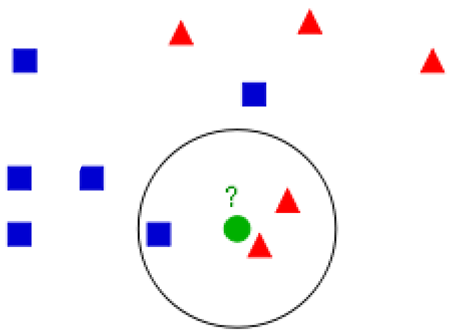
Supervised Learning: Support Vector Machines

Practical details:

- Scikit-Learn (sklearn) Python Package has excellent documentation
    - http://scikit-learn.org/stable/modules/svm.html
- For easier problems, can pretty much use it out of the box to get decent results

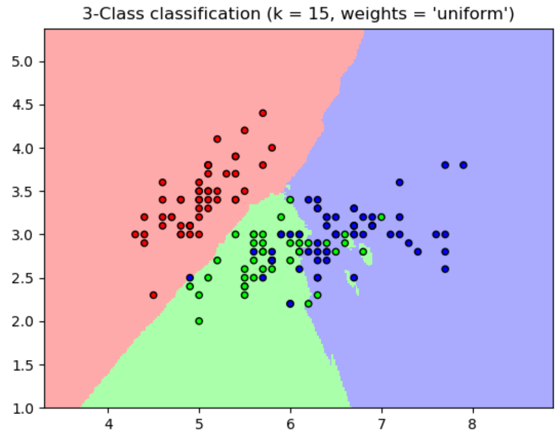
K-Nearest Neighbor

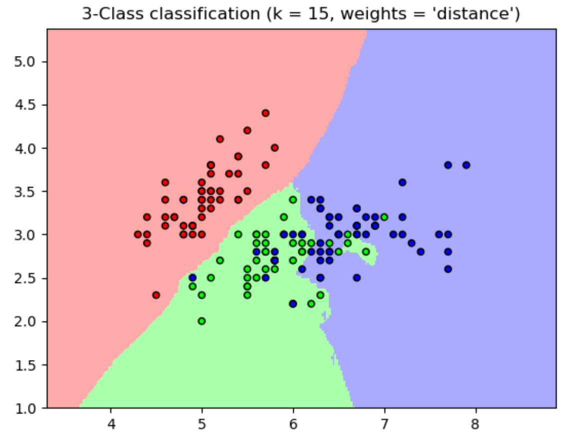
- Can be used for classification or regression
- Simple idea:
  - For a given data point p, find the K nearest labeled points
  - Assign the majority label to p
- Caveats:
  - The order that you label points can matter!
  - Slow! Lots of comparisons required.
  - Choosing ‘K’ has a big impact



Weighted K-Nearest Neighbor

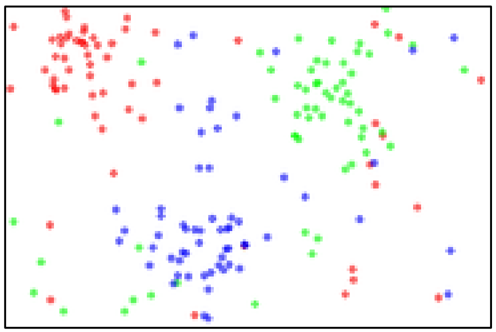
Weight each sample's influence by how far away it is from p



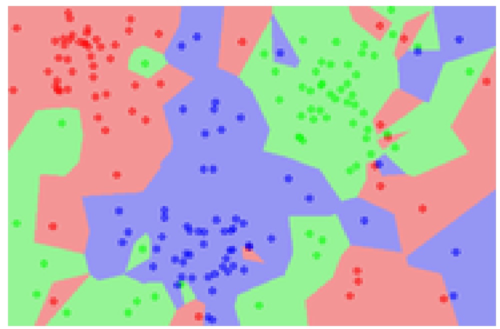


Reduced K-Nearest Neighbor

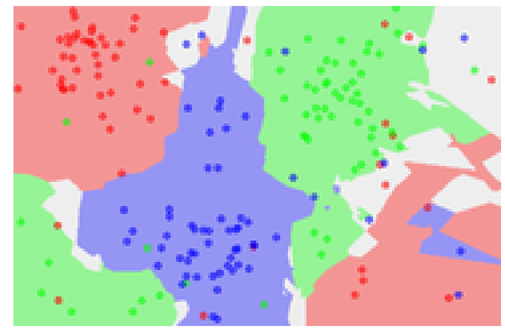
Dataset



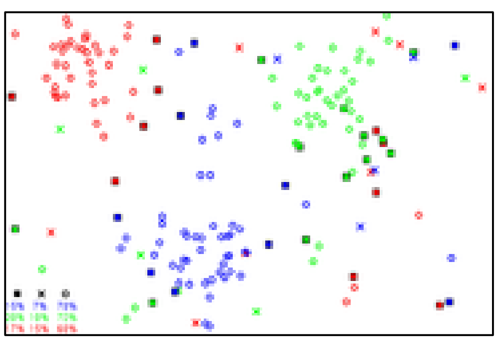
1-NN Classification



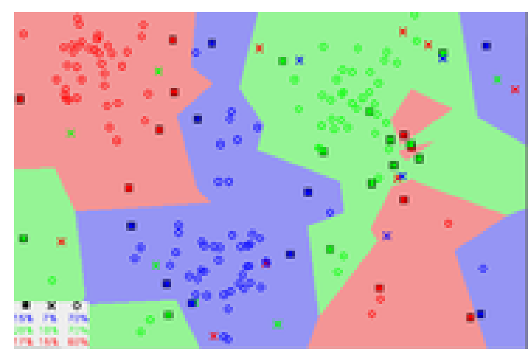
5-NN Classification



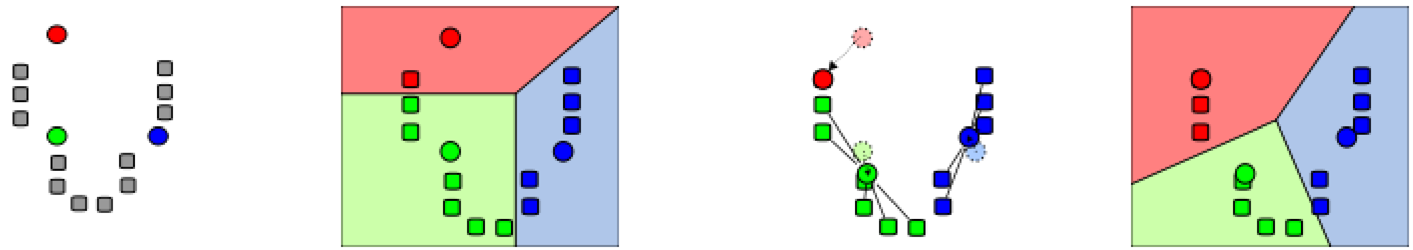
Reduced Dataset



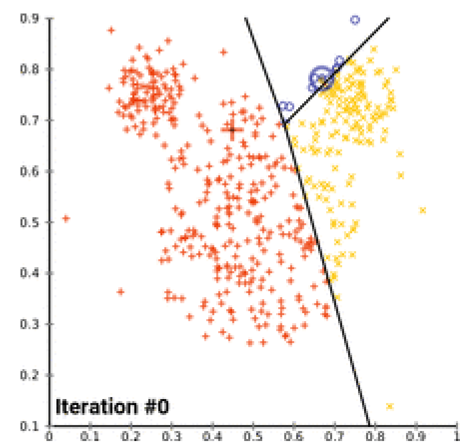
1-NN Classification



Unsupervised Learning: K-Means Clustering



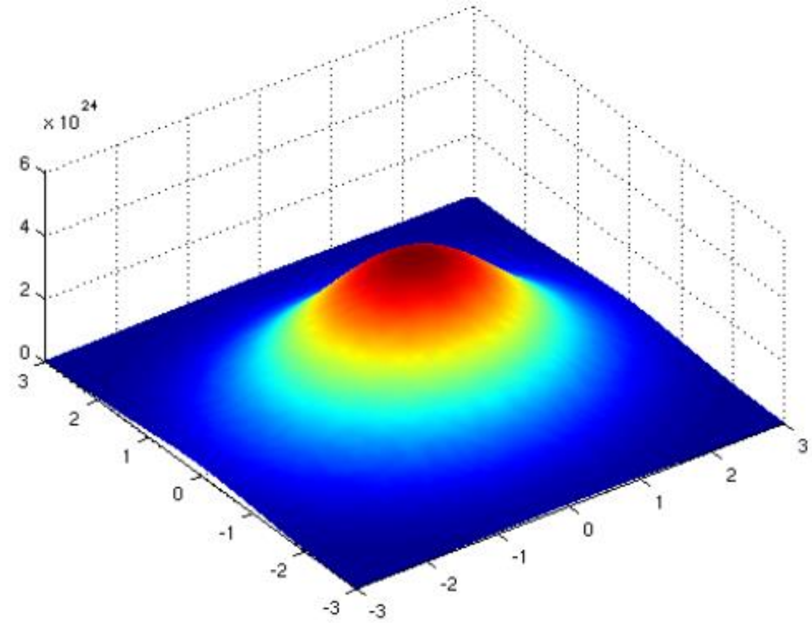
1. Randomly initialize k clusters at random positions.
2. Classify every data point as belonging to a cluster by Euclidean distance measurement (closest cluster wins)
3. Calculate centroid of each cluster. Relocate cluster to centroid position.
4. Repeat 2-3 until centroids converge.



Gaussian Distribution

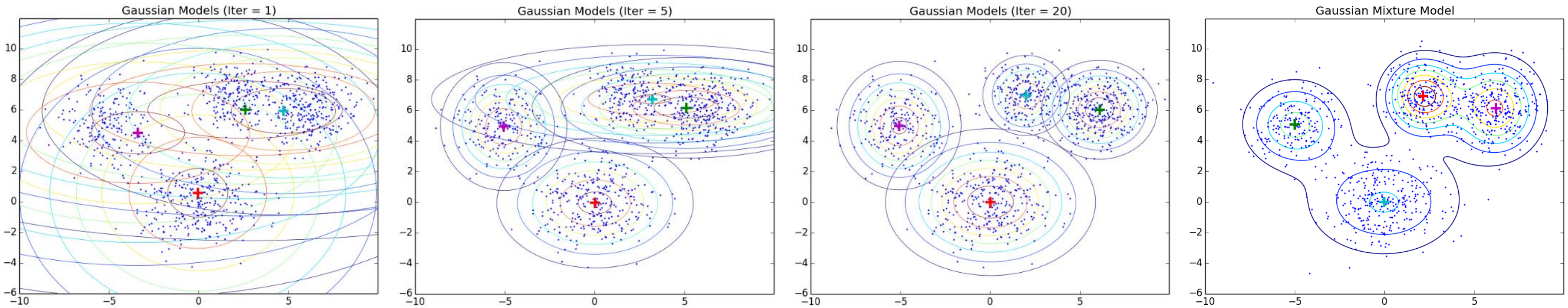
$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$\mathcal{N}(\mathbf{x}|\mu,\mathbf{\Sigma}) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\mathbf{\Sigma}|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\mu)^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)}$$

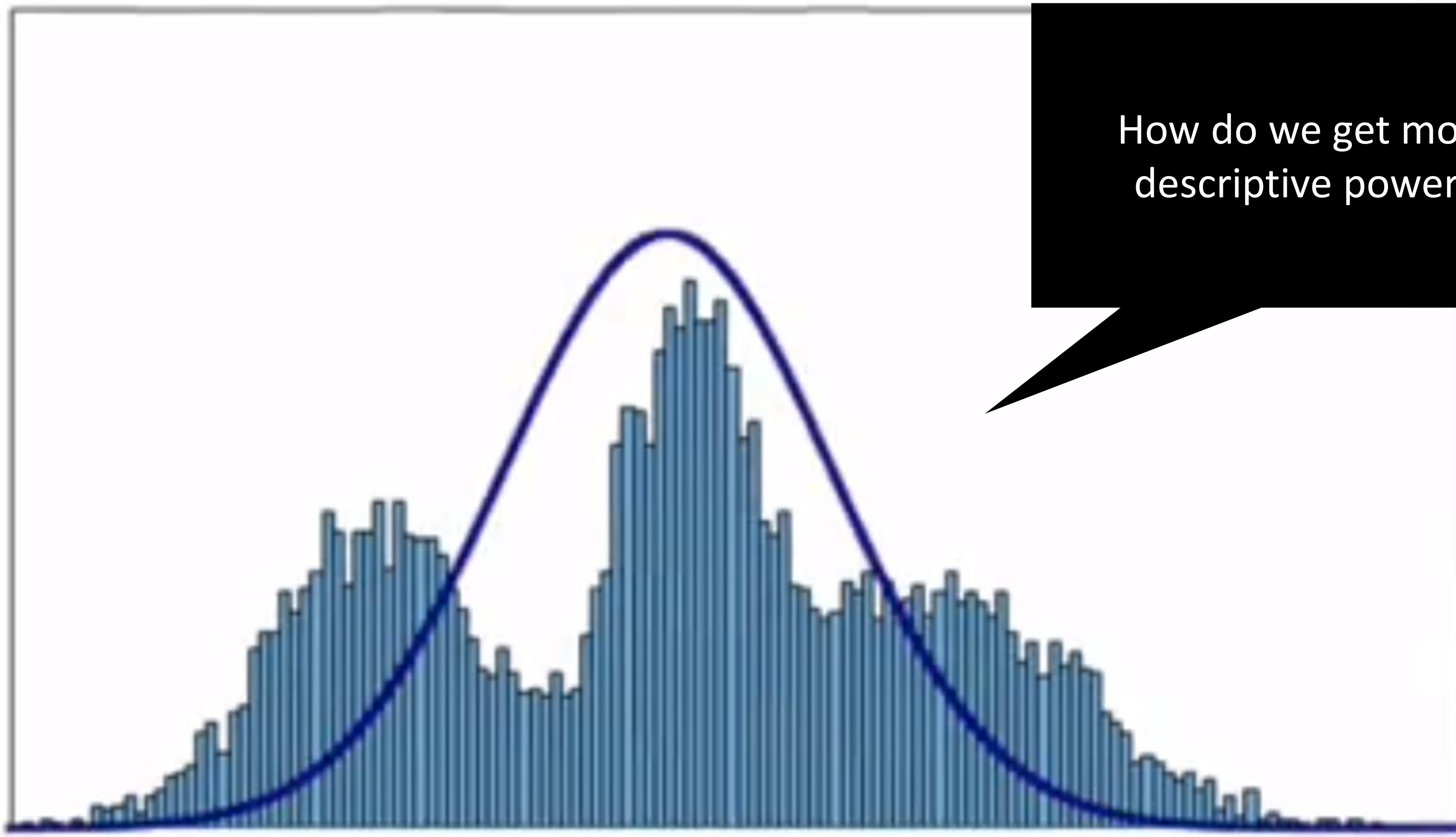


Gaussian Mixture Model (GMM)

- Model that represents a distribution that data are drawn from
  - Lets us classify existing data, and generate "plausible" new data!
- Generalization of k-means clustering to incorporate information about the covariance of the data



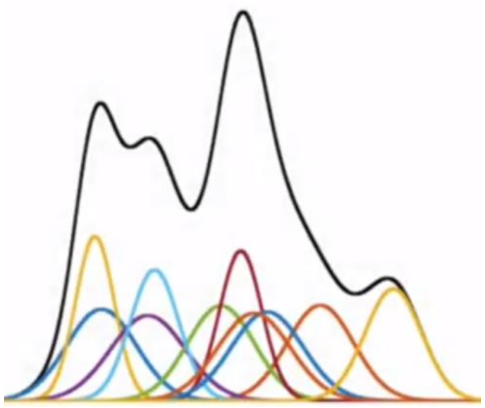
Modeling with one Gaussian



How do we get more descriptive power?

Mixture: Sum of Gaussians

$$p(x) = \sum_{k=1}^{K} w_k * g_k\,(x \mid \mu_k, \Sigma_k)$$

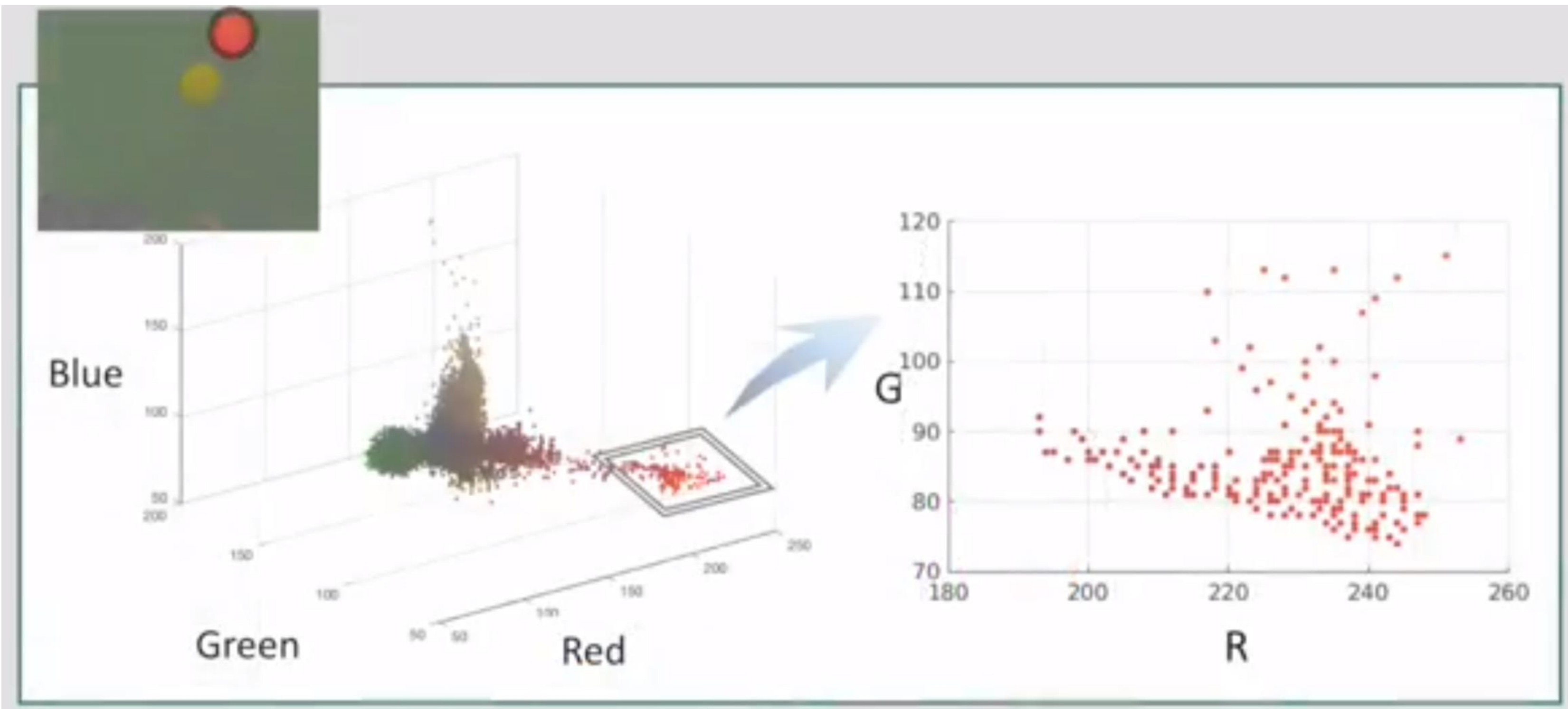


g_k is a Gaussian distribution with mean μ_k and covariance matrix Σ_k

w_k is the mixing coefficient of a particular Gaussian (its weight)

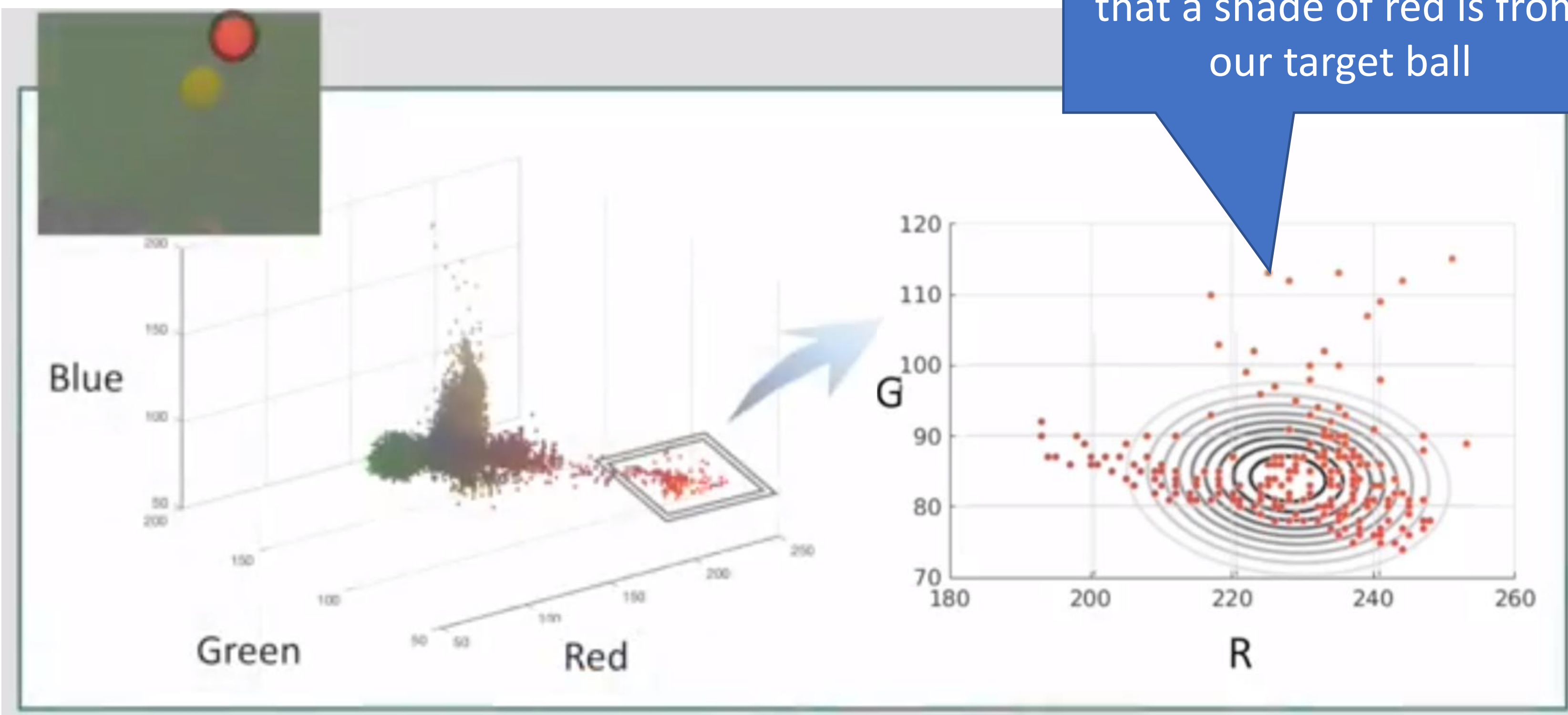
Important: $\mathrm{w_k} > 0$, $\sum_{k=1}^{K} w_k = 1$ must hold

Example: Color Filtering

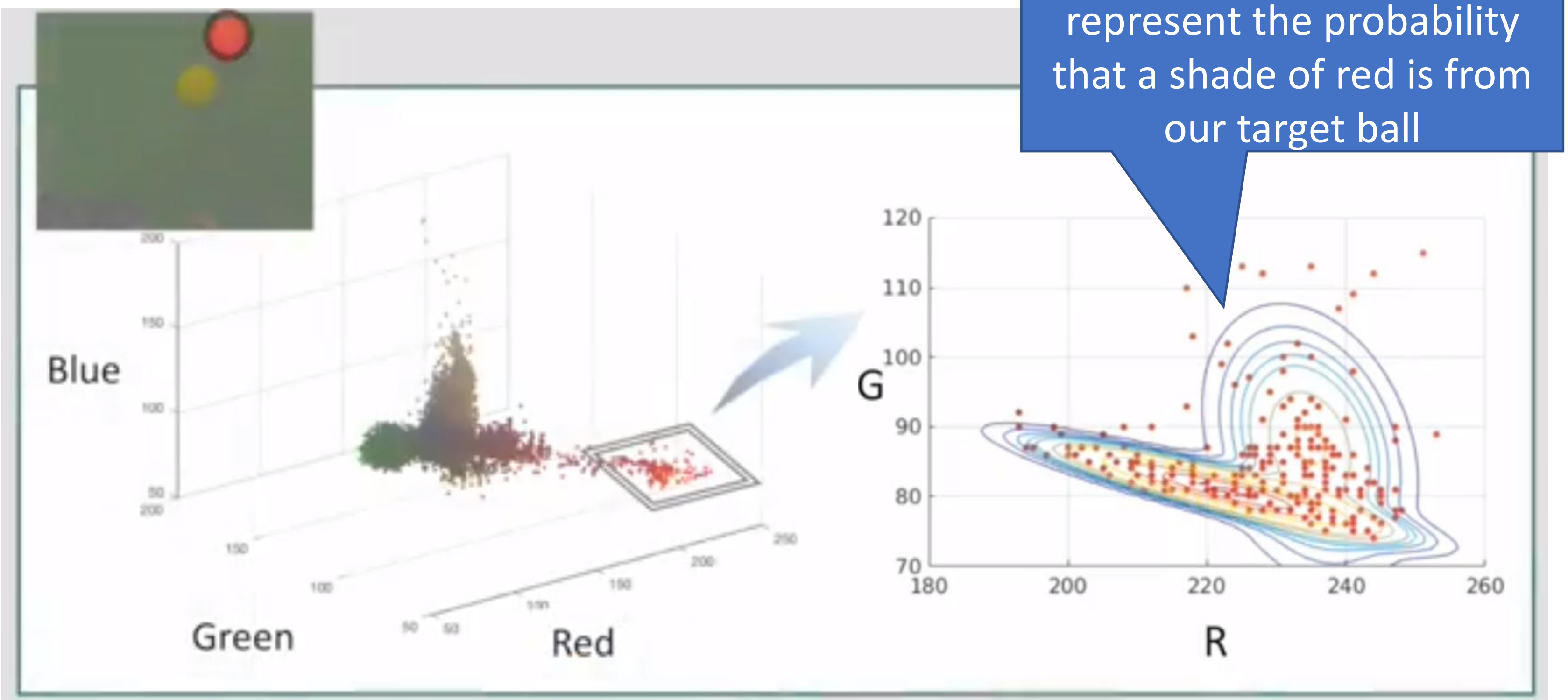


Example: Color Filtering

A single 2D Gaussian can represent the probability that a shade of red is from our target ball



Example: Color Filtering



A mixture of two 2D Gaussians can **better** represent the probability that a shade of red is from our target ball

Exercise:
Improving the original Roomba vacuum cleaner

- Current Operation: Random walk with Vacuum on
  - Limitations:
    - Vacuum operation consumes battery
    - Doesn't return home to charge
    - Has IR sensors to detect ledges so it doesn't fall down stairs
    - Has bump sensors to detect collisions with stationary objects
- If the Roomba had a sensor that could detect dirt underneath it…

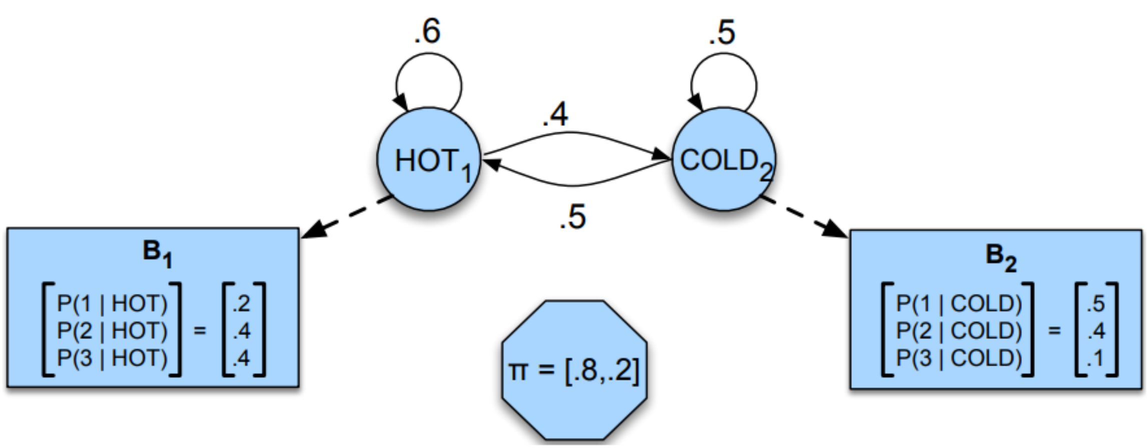
How could we design a smarter Roomba?

Objective: Minimize the amount of dirt on the floor at any given moment
Ideas:
- Odometry
- Mapping
- Machine Learning

Hidden Markov Models

Variables:

$S = s_1, s_2, \ldots, s_N$	(States)
$V = v_1, v_2, \ldots, v_k$	(Observation Vocab.)
$A = a_{11}, \ldots a_{ij}, \ldots a_{NN}$	(Transition prob. Matrix)
$B = P(o_t \vert s_i)\ \forall\ i \in [1, N], t \in [1, T]$	(Obs. Emission Probs)
$\pi = \pi_1, \pi_2, \ldots, \pi_N$	(Initial prob. distribution)
$O = o_1, o_2, \ldots, o_T$	(Observation Sequence)
$Q = s_1, s_2, \ldots, s_T$	(State Sequence)



Three Types of Problems

- **Likelihood:**

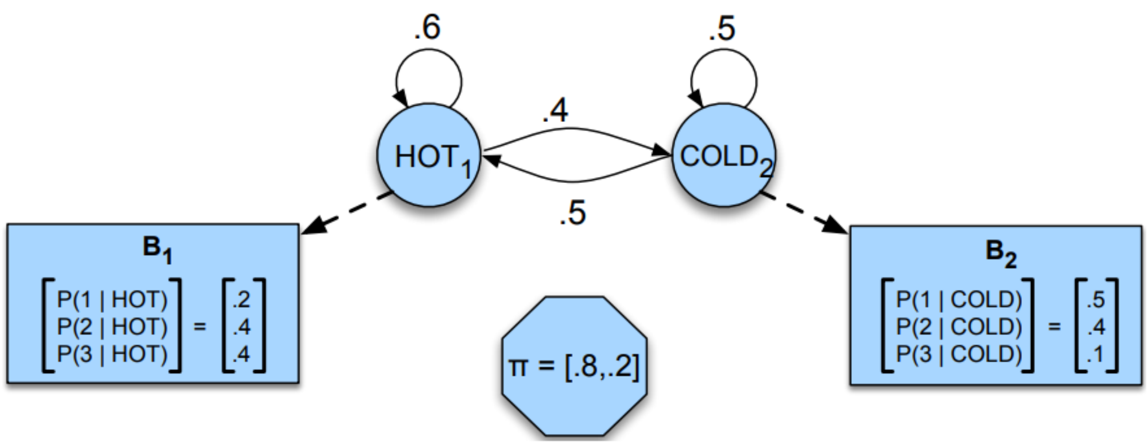
  Given A, B, O … Determine $P(O|A, B)$

- **Decoding:**

  Given $\mathrm{A}, \mathrm{B}, O$ … Determine the 'best' hidden state sequence

- **Learning:**

  Given O and S … Determine A, B



Likelihood Computation

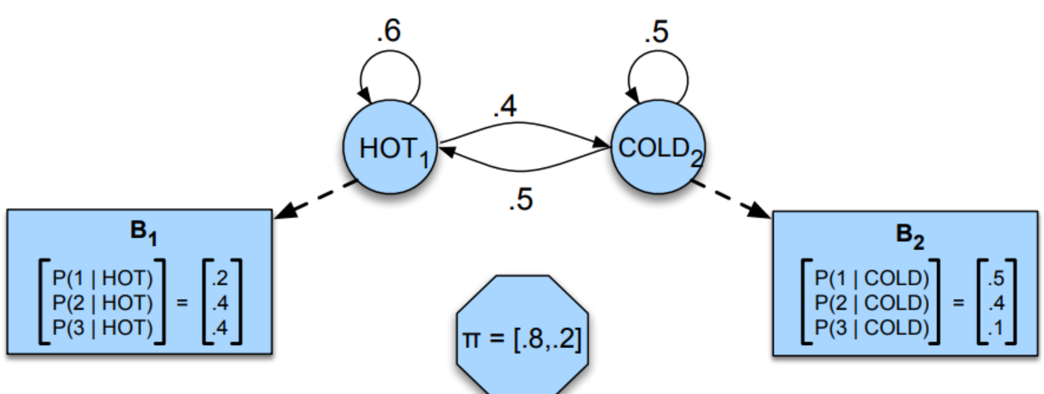
Likelihood: Given $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{O}$ … Determine $\boldsymbol{P(O|A,B)}$

Example: $O = \{3, 1, 3\}$

$$P(O|Q) = \prod_{i=1}^{T} P(o_i|q_i)$$ – Prob. of O given State Seq. Q

For one possible state sequence (*hot, hot, cold*):

$$P(3\ 1\ 3\ |hot\ hot\ cold) = P(3|hot) \times P(1|hot) \times P(3|cold)$$



Likelihood Computation

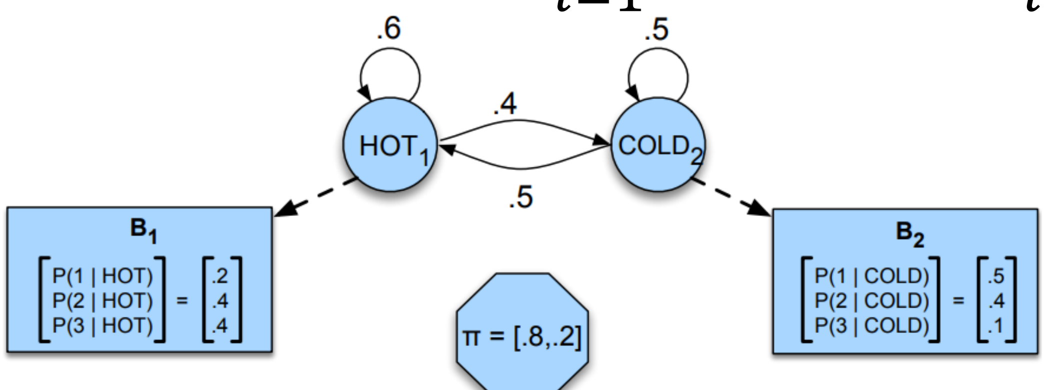
Example: Given A,B and $\boldsymbol{O} = \{3, 1, 3\}$ -- Determine $\boldsymbol{P(O|A, B)}$

But we don't know the state sequence!

Instead, we must weight each sequence by its probability.

$$P(O, Q) = P(O|Q) \times P(Q) = \prod_{i=1}^{T} P(o_i|q_i) \times \prod_{i=1}^{T} P(q_i|q_{i-1})$$

$$\boxed{P(q_0) = \pi(q_0)}$$



Likelihood Computation

$$P(O|Q) = \prod_{i=1}^{T} P(o_i|q_i)$$

$$P(O,Q) = P(O|Q) \times P(Q) = \prod_{i=1}^{T} P(o_i|q_i) \times \prod_{i=1}^{T} P(q_i|q_{i-1})$$

$$P(O) = \sum_{Q} P(O,Q) = \sum_{Q} P(O|Q)P(Q)$$

N^T Sequences!

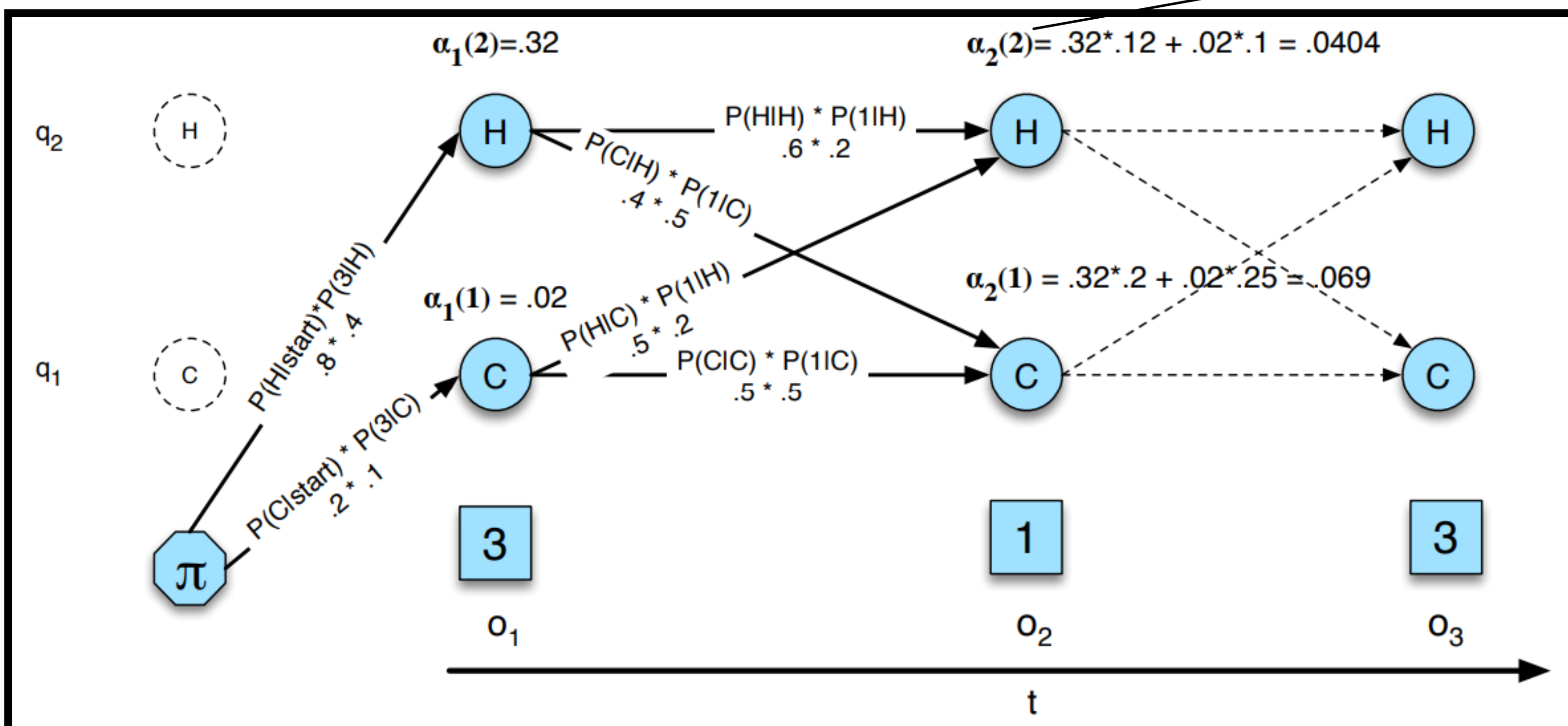
Likelihood Computation: Forward Algorithm

Example: Given A,B and $\boldsymbol{O} = \{3, 1, 3\}$ -- Determine $\boldsymbol{P(O|A,B)}$

Infeasible to solve with $O(N^T)$ algorithm.

Can do it in $O(N^2T)$ with Dynamic Programming!

$\alpha_t(j)$:
P(s=j|T=t)



Likelihood Computation: Forward Algorithm

Example: Given A,B and $\boldsymbol{O} = \{3, 1, 3\}$ -- Determine $\boldsymbol{P(O|A, B)}$

$$\alpha_t(j) = P(o_1, o_2, \ldots, o_t, q_t = j \mid A, B)$$

$$\alpha_t(j) = \sum_{i=1}^{N} \underbrace{\alpha_{t-1}(i)}_{\text{Prev}} * \underbrace{a_{ij}}_{\text{P(i -> j)}} * \underbrace{b_j(o_t)}_{\text{P(o|s)}}$$

function FORWARD(*observations* of len T, *state-graph* of len N) **returns** *forward-prob*

create a probability matrix *forward[N,T]*
for each state s **from** 1 **to** N **do** ; initialization step
$\quad forward[s,1] \leftarrow \pi_s * b_s(o_1)$
for each time step t **from** 2 **to** T **do** ; recursion step
$\quad$ **for** each state s **from** 1 **to** N **do**

$$forward[s,t] \leftarrow \sum_{s'=1}^{N} forward[s',t-1] * a_{s',s} * b_s(o_t)$$

$$forwardprob \leftarrow \sum_{s=1}^{N} forward[s,T] \qquad \text{; termination step}$$

return *forwardprob*

State Sequence Computation: Viterbi Algorithm

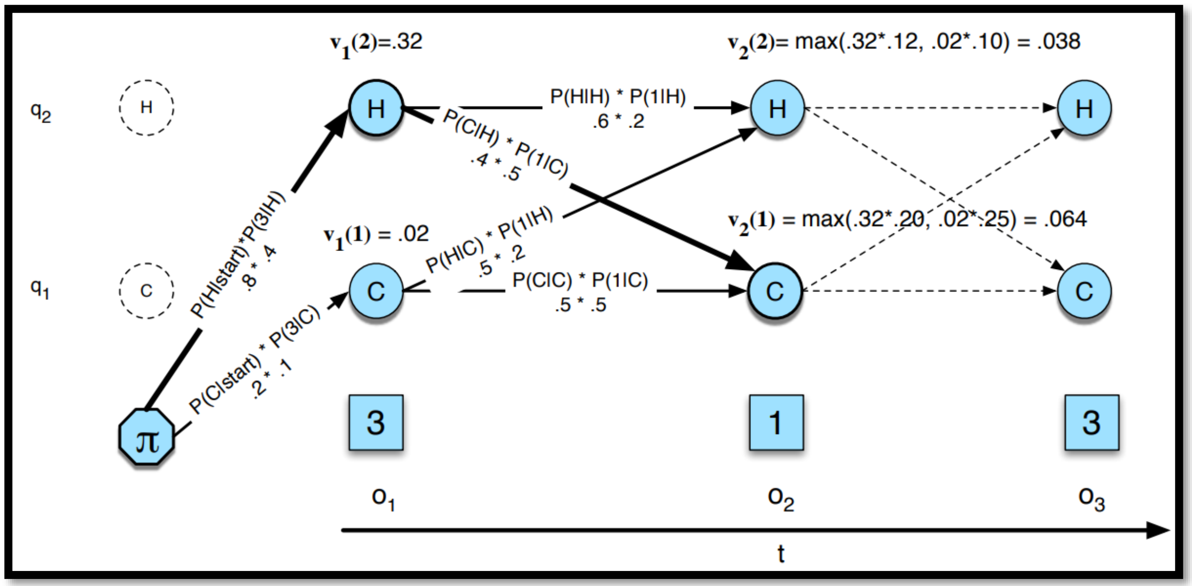
Decoding: Given $\mathbf{A}, \mathbf{B}, \boldsymbol{O}$ … determine $\boldsymbol{Q}$

Option 1: Run Forward Algorithm on all state sequences

Option 2: Use Dynamic Programming

$$v_t(j) = \max_{q_1,\ldots,q_{t-1}} P(q_1 \ldots q_{t-1}, o_1 \ldots o_t, q_t = j \mid A, B)$$

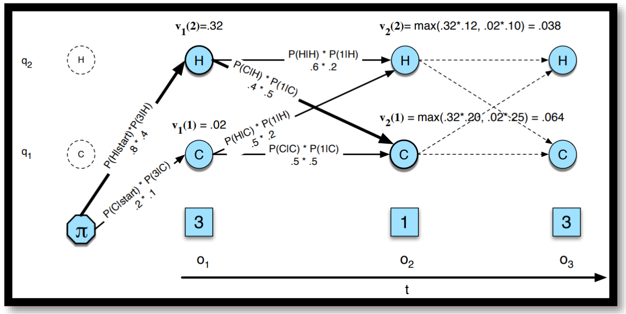
$$v_t(j) = \max_{i \in [1,N]} v_{t-1}(i) * a_{ij} * b_j(o_t)$$



State Sequence Computation: Viterbi Algorithm

function VITERBI(*observations* of len T,*state-graph* of len N) **returns** *best-path*, *path-prob*

create a path probability matrix *viterbi[N,T]*
for each state s **from** 1 **to** N **do** ; initialization step
$$viterbi[s,1] \leftarrow \pi_s * b_s(o_1)$$
$$backpointer[s,1] \leftarrow 0$$
for each time step t **from** 2 **to** T **do** ; recursion step
for each state s **from** 1 **to** N **do**
$$viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$
$$backpointer[s,t] \leftarrow \operatorname*{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$
$$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T]$$; termination step
$$bestpathpointer \leftarrow \operatorname*{argmax}_{s=1}^{N} viterbi[s,T]$$; termination step
bestpath $\leftarrow$ the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time
return *bestpath*, *bestpathprob*



Learning an HMM's Parameters

Learning: Given $\boldsymbol{O}$ and $\boldsymbol{S}$ … Determine $\boldsymbol{A}, \boldsymbol{B}$

Challenge: Must simultaneously determine **transition probabilities** AND **emission probabilities**!

Special case of Expectation-Maximization, iteratively improving an initial estimate.

But first, let's solve for a Markov Chain (*fully observable*) given $\boldsymbol{O}, \boldsymbol{S}, \boldsymbol{Q}$

Learning Not-so-HMM Parameters A, B

Given Sequences: { 3H, 3H, 2C } { 1C, 1C, 2C } {1C, 2H, 3H}

Compute Initial Probabilities:

$$\pi = \begin{cases} H: 1/3 \\ C: 2/3 \end{cases}$$

Compute Transition Probabilities:

P(H|H) = 2/3	P(H|C) = 1/2
P(C|H) = 1/3	P(C|C) = 1/2

$$A = \begin{cases} HH: \frac{2}{3} \quad CH: \frac{1}{3} \\ HC: \frac{1}{2} \quad CC: \frac{1}{2} \end{cases}$$

Compute Emission Probabilities:

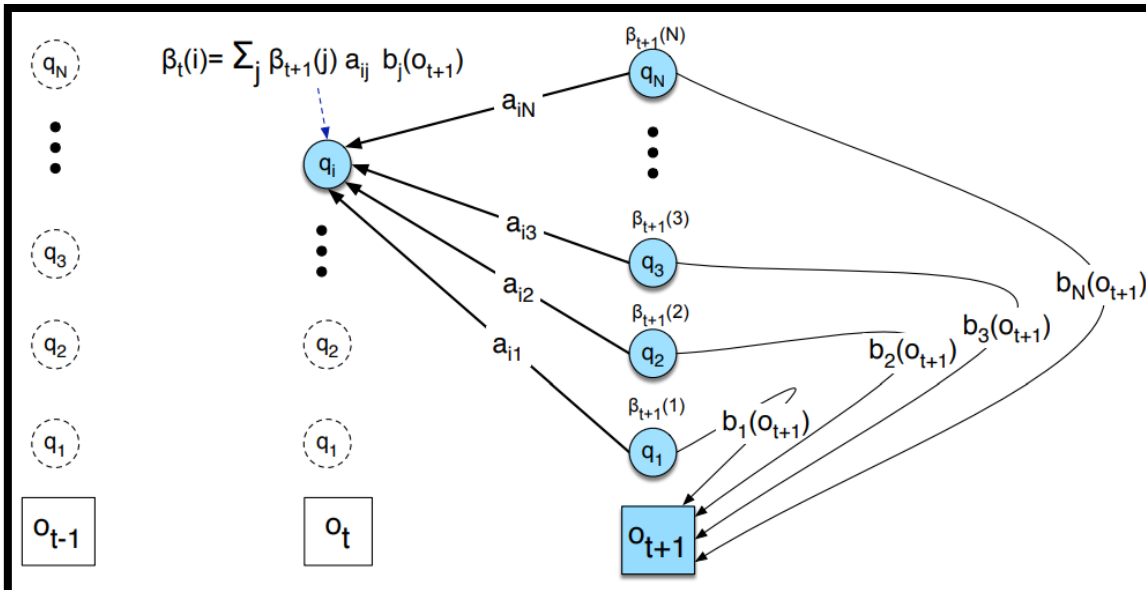
P(1|hot) = 0	P(1|cold) = 3/5
P(2|hot) = 1/4	P(2|cold) = 2/5
P(3|hot) = 3/4	P(3|cold) = 0

$$B = \begin{cases} Hot = \{ 0, \frac{1}{4}, \frac{3}{4} \} \\ Cold = \{ \frac{3}{5}, \frac{2}{5}, 0 \} \end{cases}$$

Backward Algorithm

Backward Probability: $\beta_t(i) = P(o_{t+1}, o_{t+2}, \ldots o_t | q_t = i, A, B)$

If we're in state i at time t, what's P(Obs) from then to end?



Initialization: $\beta_T(i) = 1, \;\; i \in [1, N]$

Recursion: $\beta_t(i) = \sum_{j=1}^{N} a_{ij} * b_j(o_{t+1}) * \beta_{t+1}(j)$, $i \in [1, N], t \in [1, T)$

Termination: $P(O|A, B) = \sum_{j=1}^{N} \pi_j * b_j(o_1) * \beta_1(j)$

Forward-Backward: Learning A

$$\hat{a}_{ij} = \frac{Expected\ \#transitions\ from\ i\ to\ j}{Expected\ \#transitions\ from\ i}$$

To compute numerator:

1. Assume we have probability estimate for $i \rightarrow j$ at time t
2. Now assume we had that for all t: sum over all $t \in [0, T)$ to get the total count for $i \rightarrow j$

Define ξ_t as probability of transition from i to j at time t:

$$\xi_t(i, j) = P(q_t = i, q_{t+1} = j \mid O, A, B)$$

…But we don't know the relation between O and Q!

Forward-Backward: Learning A

Define ξ_t as probability of transition from i to j at time t:

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j \mid O, A, B)$$

...But we don't know the relation between O and Q!

So we define sort-of-$\xi_t(i,j) = P(q_t = i, q_{t+1} = j, O \mid A, B)$

$$\text{sort-of-}\xi_t(i,j) = \alpha_t(i) * a_{ij} * b_j(o_{t+1}) * \beta_{t+1}(j)$$

Forward

$$\boxed{\alpha_t(j) = P(o_1, o_2, \dots, o_t, q_t = j \mid A, B)}$$

Backward

$$\boxed{\beta_t(i) = P(o_{t+1}, o_{t+2}, \dots o_t | q_t = i, A, B)}$$

Forward-Backward: Learning A

$$\text{sort-of-}\xi_t(i,j) = \alpha_t(i) * a_{ij} * b_j(o_{t+1}) * \beta_{t+1}(j)$$

How do we go from $\boxed{P(q_t = i, q_{t+1} = j, O \mid A, B)}$ to $\boxed{P(q_t = i, q_{t+1} = j \mid O, A, B)}$

Recall: $P(X|Y,Z) = \dfrac{P(X,Y|Z)}{P(Y|Z)}$

Thus, because $P(O|A,B) = \sum_{j=1}^{N} \alpha_t(j) * \beta_t(j)$

$$\xi_t(i,j) = \frac{\alpha_t(i) * a_{ij} * b_j(o_{t+1}) * \beta_{t+1}(j)}{\sum_{j=1}^{N} \alpha_t(j) * \beta_t(j)}$$

Forward

$\boxed{\alpha_t(j) = P(o_1, o_2, \ldots, o_t, q_t = j \mid A, B)}$

Backward

$\boxed{\beta_t(i) = P(o_{t+1}, o_{t+2}, \ldots o_t | q_t = i, A, B)}$

Forward-Backward: Learning A

To compute numerator:

1. Assume we have probability estimate for $i \to j$ at time t
2. Now assume we had that for all t: sum over all $t \in [0, T)$ to get the total count for $i \to j$

$$\hat{a}_{ij} = \frac{Expected\ \#transitions\ from\ i\ to\ j}{Expected\ \#transitions\ from\ i} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{k=1}^{N} \xi_t(i,k)}$$

$$\xi_t(i,j) = \frac{\alpha_t(i) * a_{ij} * b_j(o_{t+1}) * \beta_{t+1}(j)}{\sum_{j=1}^{N} \alpha_t(j) * \beta_t(j)}$$

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j \mid O, A, B)$$

Forward

$$\boxed{\alpha_t(j) = P(o_1, o_2, \ldots, o_t, q_t = j \mid A, B)}$$

Backward

$$\boxed{\beta_t(i) = P(o_{t+1}, o_{t+2}, \ldots o_t | q_t = i, A, B)}$$

Forward-Backward: Learning B

$$\hat{a}_{ij} = \frac{Expected\ \#transitions\ from\ i\ to\ j}{Expected\ \#transitions\ from\ i} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{k=1}^{N} \xi_t(i,k)}$$

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j \mid O, A, B)$$

Now we need to compute observation emission probability:

$$\widehat{b}_j(v_k) = \frac{Expected\ \#\ of\ v_k\ seen\ in\ state\ j}{Expected\ \#times\ in\ state\ j}$$

Forward

$$\boxed{\alpha_t(j) = P(o_1, o_2, \dots, o_t, q_t = j \mid A, B)}$$

Backward

$$\boxed{\beta_t(i) = P(o_{t+1}, o_{t+2}, \dots o_t | q_t = i, A, B)}$$

Forward-Backward: Learning B

Now we need to compute observation emission probability:

$$\hat{b}_j(v_k) = \frac{Expected\ \#\ of\ v_k\ seen\ in\ state\ j}{Expected\ \#times\ in\ state\ j}$$

But first, we need to know prob. of being in state j at time t

$$\gamma_t(j) = \mathrm{P}(\mathrm{q_t} = j \mid O, A, B) = \frac{P(q_t = j, O|A, B)}{P(O|A, B)}$$

$$\gamma_t(j) = \frac{\alpha_t(j) * \beta_t(j)}{P(O|A, B)}$$

Forward

$$\boxed{\alpha_t(j) = P(o_1, o_2, \ldots, o_t, q_t = j \mid A, B)}$$

Backward

$$\boxed{\beta_t(i) = P(o_{t+1}, o_{t+2}, \ldots o_t | q_t = i, A, B)}$$

Forward-Backward: Learning B

Now we need to compute observation emission probability:

$$\hat{b}_j(v_k) = \frac{Expected\ \#\ of\ v_k\ seen\ in\ state\ j}{Expected\ \#times\ in\ state\ j} = \frac{\sum_{t=1}^{T} \gamma_t(j) * I(o_t = v_k)}{\sum_{t=1}^{T} \gamma_t(j)}$$

$\gamma_t(j) =$ prob. of being in state j at time t

$$\gamma_t(j) = \frac{\alpha_t(j) * \beta_t(j)}{P(O|A,B)}$$

Forward

$$\boxed{\alpha_t(j) = P(o_1, o_2, \ldots, o_t, q_t = j \mid A, B)}$$

Backward

$$\boxed{\beta_t(i) = P(o_{t+1}, o_{t+2}, \ldots o_t | q_t = i, A, B)}$$

Expectation-Maximization on A, B

E-Step: Compute state occupancy count γ, expected state transition count ξ using existing A, B probabilities

M-Step: Compute A, B using existing γ and ξ probabilities

$\alpha_t(j)$ = prob. to be in state j at t

$\beta_t(j)$ = prob. of O from state j at t

$\xi_t(i,j)$ = prob. of transition from i to j at time t

$\gamma_t(j)$ = prob. of being in state j at time t

function FORWARD-BACKWARD(*observations* of len *T*, *output vocabulary V*, *hidden state set Q*) **returns** *HMM=(A,B)*

initialize *A* and *B*
iterate until convergence
E-step

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\alpha_T(q_F)} \quad \forall\, t \text{ and } j$$

$$\xi_t(i,j) = \frac{\alpha_t(i)\,a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(q_F)} \quad \forall\, t,\ i, \text{ and } j$$

M-step

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{k=1}^{N} \xi_t(i,k)}$$

$$\hat{b}_j(v_k) = \frac{\sum_{t=1 \text{ s.t. } O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

return *A*, *B*