CSCI 181 / E-181 Spring 2014

Midterm 2 Review Notes

David Wihl

April 29, 2014

Contents

1 Support Vector Machines **2**
- 1.1 Background . . . 2
- 1.2 Definitions . . . 2
- 1.3 Max Marginalization . . . 3
- 1.4 Slack Variables . . . 5
- 1.5 Duality . . . 6
- 1.6 Kernel Tricks . . . 7
- 1.7 Extensions to SVM . . . 8
- 1.8 Sources . . . 8

2 Reinforcement Learning **9**
- 2.1 Markov Decision Processes . . . 9
- 2.2 Expectimax . . . 10
- 2.3 Value Iteration . . . 11
  - 2.3.1 Finite Horizon . . . 11
  - 2.3.2 Infinite Horizon . . . 12
- 2.4 Policy Iteration . . . 12
- 2.5 Reinforcement Learning . . . 13
  - 2.5.1 Model Based RL . . . 14
  - 2.5.2 Model Free RL . . . 15
- 2.6 Partially Observable MDP . . . 15
  - 2.6.1 Belief State . . . 17
  - 2.6.2 Policy Iteration Methods . . . 17
- 2.7 Sources . . . 17

3 Expectation Maximization **18**

3.1 Mixture Models . . . 18
3.1.1 Soft K-means . . . 19
3.1.2 Mixture of Gaussians . . . 19
3.1.3 Generative Latent Variable Models . . . 20
3.1.4 Expectation Maximization Intro . . . 20
3.1.5 Simplest Possible Gaussian Mixture Model . . . 21
3.2 Expectation Maximization in Depth . . . 21
3.3 Hidden Markov Models . . . 22
3.4 Sources . . . 22

1 Support Vector Machines

1.1 Background

Characteristics of SVMs:

- *stock* – SVMs are "off the shelf" and ready to use. No special modification is necessary.
- *linearly separable* – assumes that linear separation is possible. Used natively as a binary classifier.
- *convex optimization*. SVM originated as a backlash against neural nets due to nets' non-convexity. In Neural Nets, results were often non-reproducible as different researchers found different results due to different initializations.
- *global optimum* – SVMs will find the global optimum.

SVMs are based on three "big ideas":

- *margin* Maximizes distance between the closest points
- *duality* Take a hard problem and transform it into an easier problem to solve.
- *kernel functions* map input vectors into higher dimensional, more expressive features to avoid costly computations.

1.2 Definitions

Data: $\{x_n, t_n\}_{n=1}^{N}, t_n \in \{-1, +1\}$. t_n is the target or the expected result of the classification.

J Basis functions: $\phi_j(x) \to \mathbb{R}$, therefore

Vector function: $\Phi X \rightarrow \mathbb{R}^J$ produces a column vector.

Objective function: $f(\mathbf{x}, \mathbf{w}, b) = \mathbf{w}^\intercal \phi(\mathbf{x}) + b$ where b is the bias.

The sign of $f(\cdot)$ will determine classification $(-1, +1)$

So the actual classifier will be:

$$y(\mathbf{x}, \mathbf{w}, b) = \begin{cases} +1, & \text{if } \mathbf{w}^\intercal \phi(\mathbf{x}) + b > 0 \\ -1, & \text{otherwise} \end{cases}$$

Unlike Logistic Regression (which uses $\{0, 1\}$), it is preferable to use $\{-1, +1\}$ as the classification result. If $t_n * y$ is positive, then the produced classification is correct (positive $\times$ positive is positive, negative $\times$ negative is also positive).

Decision Boundary is the hyperplane where $\mathbf{w}^\intercal \phi(\mathbf{x}) + b = 0$. We want to find the Decision Boundary that creates the most separation between the two different classes by maximizing the distance between the two closest points. The distance between the Decision Boundary and the closest point is called the *margin*. The points closest to the Decision Boundary are called the *support vectors*.

1.3 Max Marginalization

The margin is determined by the orthogonal distance from the closest point to the Decision Boundary:

$$\frac{|\mathbf{w}^\intercal \phi(\mathbf{x}) + b|}{||\mathbf{w}||} \tag{1}$$

Maximizing the margin can be written as:

$$\underset{w,b}{\text{argmax}} \left\{ \min_n (t_n \cdot (\mathbf{w}^\intercal \phi(\mathbf{x}) + b)) \cdot \frac{1}{||w||} \right\} \tag{2}$$

Maximizing the margin helps ensure that points which are close to margin will not be pushed over the boundary by noise.

$\mathbf{w}$ is orthogonal to vectors in the Decision Boundary. Here's how: pick two points on the Decision Boundary $\phi(x_1)$ and $\phi(x_2)$. So

$$\begin{aligned} & \mathbf{w}^\intercal (\phi(x_2) - \phi(x_1)) = 0 \text{ for orthogonal dot product} \\ & \mathbf{w}^\intercal \phi(x_2) - \mathbf{w}^\intercal \phi(x_1) = 0 \\ \text{Note: } & \mathbf{w}^\intercal \phi(x_n) = (-b), \text{ so} \\ & = (-b) - (-b) \\ & = 0 \end{aligned}$$

Since $\mathbf{w}$ is orthogonal, we want to maximize it. It is not unit length, but could be, by scaling with a factor of r.

The margin is defined where

$$\mathbf{w}^\intercal \phi(x) + b = \pm 1$$

The origin can be found at $\frac{b}{||w||}$.

See Stanford CS229 SVM Notes re: Functional vs. Geometric Margins

The support vector is defined as $r\frac{\mathbf{w}}{||\mathbf{w}||_2}$, where r is multiplied by the unit vector orthogonal to the Decision Boundary hyperplane.

Define the point where the vector meets the plane as $\phi_\perp(\mathbf{x})$, so

$$\phi(\mathbf{x}) = \phi_\perp(\mathbf{x}) + r\frac{\mathbf{w}}{||\mathbf{w}||_2} \tag{3}$$

Solving for r, multiple both sides by $\mathbf{w}^\intercal$.

(Recall: $\mathbf{w}^\intercal\mathbf{w} = ||\mathbf{w}||^2$)

$$\begin{aligned}\mathbf{w}^\intercal\phi(\mathbf{x}) =& \mathbf{w}^\intercal\phi_\perp(\mathbf{x}) + r\frac{\mathbf{w}^\intercal\mathbf{w}}{||\mathbf{w}||} &(4)\\ =& (-b) + r||\mathbf{w}|| &(5)\end{aligned}$$

Therefore, the margin for a point $\mathbf{x}$.

$$\begin{aligned}r =& \frac{\phi(\mathbf{x})^\intercal\mathbf{w} + b}{||w||} &(6)\\ =& \frac{f(\mathbf{x}, \mathbf{w}, b)}{||w||} &(7)\end{aligned}$$

This makes it easy to calculate how far away a point is from the Decision Boundary. r is strictly not a length because it could be negative. However, we only care about the actual distance to the boundary.

Margin for a datum n:

$$margin = t_n\frac{w^\intercal\boldsymbol{\phi}(x_n) + b}{||w||}$$

This is getting close to a loss function as we can now figure out the worst of these. The Margin for all the training data will be the point closest to the Decision Boundary:

$$min_n\left\{t_n\frac{w^\intercal\boldsymbol{\phi}(x_n) + b}{||w||}\right\}$$

As mentioned at the beginning of this section, the Objective Function is

$$\mathbf{w}^*, b^* = \underset{w,b}{\text{argmax}} \left\{ \frac{1}{||w||} \cdot \min_n [t_n(\mathbf{w}^\intercal \boldsymbol{\phi}(\mathbf{x_n}) + b)] \right\}$$

but now we can simplify some things. $\mathbf{w}$ are b are scale free (if we multiply by some β, the max and min will still be the same.)

Let's define a set of linear constraints such that the margin is always ≥ 1 to make this easier to solve.

$$\mathbf{w}^*, b^* = \underset{\mathbf{w},b}{\text{argmax}} \frac{1}{||w||} \tag{8}$$

subject to

$$t_n \cdot (\phi(x_n)^\intercal \mathbf{w} + b) \geq 1 \, \forall \, n \tag{9}$$

This can be made even easier. Finding the max of $\frac{1}{||w||}$ is like finding the min of $||w||^2$ because $||\mathbf{w}|| \geq 0$, so

$$\mathbf{w}^*, b^* = \underset{\mathbf{w},b}{\text{argmin}} \left\{ \frac{1}{2} ||w||_2^2 \right\} \tag{10}$$

$$\text{s.t. } t_n(\mathbf{w}^\intercal \boldsymbol{\phi}(\mathbf{x}_n) + b) \geq 1 \tag{11}$$

so this reduces to just a quadratic program (QP) with linear constraints that could be solved by any number of commercial packages and produces a global minimum. Note that the closest point to the margin will have a scaled distance $= \pm 1$ ("the tightest").

1.4 Slack Variables

If the data is not strictly linearly separable, it can mitigated by slack variables.

$\xi_n \leftarrow$ one for each datum.

$\xi_n = 0$ then the datum is correctly classified and outside the margin.

$0 < \xi_n \leq 1$ the datum is correctly classified and within the margin

$\xi_n > 1$ the datum is misclassified

We will now add ξ as a constraint to minimize.

$$t_n \cdot (\phi(x_n)^\intercal \mathbf{w} + b) \geq 1 - \xi_n$$

New objective function:

$$\mathbf{w}^*, b^*, \xi^* = \underset{\mathbf{w},b,\xi}{\operatorname{argmin}} \left\{ ||w||^2 + c\sum_{n=1}^{N} \xi_n \right\} \tag{12}$$

$$\text{s.t. } t_n(\phi(x_n)^\intercal \mathbf{w} + b) \geq 1 - \xi_n \tag{13}$$

$$\xi \geq 0 \quad \forall\, n \tag{14}$$

where $c > 0$ is the regularization parameter. A small C means that you don't care much about errors. The sum of ξ is the upper bound on how many can be wrong. If $c = 0$, it becomes the original function. Typically,

$$c = \frac{1}{\nu N}, \text{ where } 0 < \nu \leq 1$$

where ν is the tolerance for percentage willing to get wrong.

1.5 Duality

First a recap of Lagrange Multipliers.

Lagrange multipliers solve for $f(x, y, z)$ subject to $g(x, y, z) = k$ where $g(\cdot)$ is the *constraint*.

Form:

$$F(x, y, z, \lambda) = f(x, y, z) - \lambda(g(x, y, z) - k)$$

then solve for $F_x = 0, F_y = 0, F_z = 0, F_\lambda = 0$ using partial derivatives which will provide a max or min.

Maximizing the margin:

$$\mathbf{w}^*, b^* = \underset{w,b}{\operatorname{argmax}} \left\{ \min_n (t_n \cdot (\mathbf{w}^\intercal \phi(\mathbf{x}) + b)) \cdot \frac{1}{||w||} \right\} \tag{15}$$

$$= \underset{w,b}{\operatorname{argmax}} \frac{1}{||w||} \min_n (t_n \cdot (\mathbf{w}^\intercal \phi(\mathbf{x}) + b)) \tag{16}$$

This is still hard to differentiate. This is scalable by β.

We want to try a lot of basis functions. Duality allows us to try weights on *data*, rather than basis functions. This allows an infinite number of dimensions.

Solution: use duality. Associate a scalar with each constraint $\alpha = [\alpha_1, \alpha_2, ..., \alpha_N]$, each of which must be non-negative. Each constraint has it's own α which serves as the Lagrange

multiplier for each constraint.

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{n=1}^{N} \alpha_n (t_n(\mathbf{w}^\intercal \phi(\mathbf{x}) + b) - 1)$$

The summation term will be negative if the constraint is violated. By throwing a minus sign in front and then maximizing α will make the term really large (maybe ∞). This will make min() unhappy - causing a huge value in the loss function. It is a math trick to show a constraint has been violated.

$$\mathbf{w}^*, b^* = \underset{\mathbf{w},b}{\operatorname{argmin}} \max_{\alpha \geq 0} \mathcal{L}(\mathbf{w}, b, \alpha)$$

The problem is now restated as a function of α, not a $\mathbf{w}, b$ problem. This is duality.

Weak duality: the solution to an α problem makes the lower bounds of the w,b problem. Max α = min w,b. See formula 20 in maxmargin notes.

Strong duality: min and max can be switched without changing the answer. See formula 21 in maxmargin notes.

$$\frac{\partial L}{\partial \mathbf{w}} = 0$$

$$\frac{\partial L}{\partial b} = 0$$

α's are sparse so a lot of these terms will disappear. The remaining terms will provide the values as support vectors. Rewriting the Lagrangian just in terms of alpha can be found in formula 28 in maxmargin notes.

See formulas 28, 29 in maxmargin notes. This is a relatively easier problem to solve as it involves just the alphas.

Either alpha = 0, or $t_n(w^*\phi(x_n) + b^*) = 0$ These latter terms will be closest to the margin (they will be "tightest"). A small subset of the data determine the boundary. Train on all of data and throw away most of the data in order to have the classifier.

The only time we need to compute something J dimensional is on the inner product. See ϕ term.

1.6 Kernel Tricks

"A kernel function is a scalar product on two vectors mapped by basis functions into a feature space. In general, we use kernels to map into higher dimensional feature spaces, using them to circumvent costly computations in high dimension spaces."

$$K(\mathbf{x}, \mathbf{x}') = \phi^{\intercal}(\mathbf{x})\phi(\mathbf{x}')$$

Kernel functions can be generalized to any distance measure. The larger K means the distance is closer. This can be applied to text, proteins, etc. (see Alpaydin example).

Exponentiate the negative squared distance of two dissimilar things, e.g.

$$K(x, z) = \exp\left\{-\frac{1}{2}||x - z||^2\right\}$$

Don't engineer features - engineer distances. Feature space can be infinite (another way of saying this is that distances can be continuous as in a Gaussian distribution). Don't worry about features - worry about distance between things. Then create a kernel and use distance to discriminate.

Bayesian linear regression where all features interact as inner product so it can be turned into a kernel function. This can be a Gaussian kernel. Non-parametric infinite dimensional models using finite computers. Can also be used with PCA.

Mercer function, infinite dimensions (justification for duality). See also QR decomposition for large datasets.

Be cautious of pathological distance functions (as opposed to feature functions).

CS229 goes into a lot more depth about this: Gaussian kernels in depth, plus SMO and Karush-Kuhn-Tucker (KKT) conditions.

1.7 Extensions to SVM

SVM can be used for other purposes as well, such as regression. It can also be used for multi-class classification using *one vs many*.

1.8 Sources

1. Lecture 14, March 24, 2014
2. Lecture 15, March 31, 2014
3. Bishop 6.0-6.2
4. Bishop 7.0-7.1
5. Course notes - maxmargin

6. Section 7 review
7. Section 8 review
8. Stanford CS229 SVM notes
9. Machine Learning in Action, Chapter 6

2 Reinforcement Learning

Online, related to planning, as opposed to the rigid data sets we've seen so far in regression and classification.

2.1 Markov Decision Processes

All of the following states, actions, transitions, rewards and policies look like functions, but in practice they are implemented as matrices in memory, given a finite set of actions and states.

States: $\mathcal{S} = \{1, 2, 3, ...N\}$

Actions: $\mathcal{A} = \{1, 2, 3, ...M\}$ which move us from state to state over a probability distribution. (You don't always land where you expected to go.). This noisy movement is defined by:

Transition Model $p(s' \mid s, a) \leftarrow$ PMF, indexed by state, action. These are a collection of M stochastic matrices. Each matrix is $N \times N$, non-negative and all rows sum to 1. So each matrix $\mathbf{P}^{(a)}_{s,s'}$ where

$$P^{(a)}_{s,s'} \geq 0, \sum_{s'} P^{(a)}_{s,s'} = 1$$

Reward Function: $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ Could be expressed as a $N \times M$ matrix, $\mathcal{R}$

Policy: $\pi_t : \mathcal{S} \to \mathcal{A}$. **Finding the right policy is the objective here.** Array of length of N containing integers between 1 and M. $\pi_s \in \{1, 2, 3, ...M\}$. (Visualize as a deck of cards, where each card is an action containing a matrix. Each row of the matrix is "where you are" and each column is "where you are going", which sends you off to the next action / card).

Similarly, the Value function $V(s)$ (see below) could be expressed as a vector $V_{(s)} \in \mathbb{R}^N$. In big state spaces (like a chess game), the value function has to be approximated. The approximation could be generated by something like a neural net to compress the current

information about the state and next steps. (A chess game is not a fully observable Markov model).

Assumptions (for now – these assumptions will be relaxed later):

1. *fully observable* You know what state you are in.
2. *known model*
3. *Markov Property* meaning the way the world works in the future is completely summarized by the current state. The past is not relevant. "The future is independent of the past given the present."
4. *Finite actions and states*
5. *Bounded Rewards* - there is a maximum reward that dominates all rewards over all states.

Agent could live forever (so it tries to maximize average reward) or lives a finite time (so it tries to maximize short term reward). Trade off of exploration vs. exploitation

Different types of utility (u) function:

Finite Horizon* $u = \sum_{t=0}^{T-1} \mathcal{R}(s_t, a_t)$

Total Reward $u = \sum_{t=0}^{\infty} \mathcal{R}(s_t, a_t)$ (may not converge)

Total Discounted Reward* $u = \sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t)$ where the discount factor $\gamma = [0, 1)$

Long-run Average Reward $u = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathcal{R}(s_t, a_t)$

* Most common and tractable utility functions

Value means "make a good move" in the anticipation of getting a reward.

2.2 Expectimax

Take an action to maximize utility. Nature responds randomly. For now, we know everything about everything except what nature will do.

We build the whole tree, work back the probabilities and then choose the path with the highest reward. The complexity is exponential – such as $O(2^n)$ – as it involves traversing a tree completely, depth-first. See Algorithm 1 in mdp Course Notes.

2.3 Value Iteration

Reward function: *finite* - fixed set of rewards or *infinite* - get as many rewards as possible with future rewards discounted compared to current rewards.

The Objective is to search the space of possible policies that maximizes rewards.

2.3.1 Finite Horizon

In the case of a finite horizon value iteration, start at the final state and work backwards.

Imagine the "last gasp" policy, with one action left to take - pick the one that will maximize the reward.

$$\pi_1(s) = \underset{a}{\operatorname{argmax}}\, R(s,a) \tag{17}$$

$$V_1(s) = \max_a R(s,a) \tag{18}$$

where 1 represents the number of steps left to take before the agent dies.

Working backwards one more step "second to last gasp" - two actions left to take. In this case take into account the current reward and the next reward in the following step.

$$\pi_2(s) = \underset{a}{\operatorname{argmax}} \left\{ R(s,a) + \sum_{s'} P(s' \mid s,a) \max_{a'} R(s',a') \right\} \tag{19}$$

$$V_2(s) = \max_a \left\{ R(s,a) + \sum_{s'} p(s' \mid s,a) V_1(s') \right\} \tag{20}$$

So generalizing with K steps to go:

$$V_k(s) = \max_a \left\{ R(s,a) + \sum_{s'} P(s' \mid s,a) V_{k-1}(s') \right\} \tag{21}$$

Q-function = value of the (state, action) pair. Just like V though it includes the action. If I know Q, then I know what action to take. Fix s, maximize over Q, and then choose the appropriate action to take.

$$Q_k(s,a) = R(s,a) + \sum_{s'} P(s' \mid s,a) V_{k-1}(s'), \text{ where} \tag{22}$$

$$V_k(s) = \max_a Q_k(s,a) = Q_k(s, \pi_k^*(s)) \tag{23}$$

$$\pi_k^*(s) = \underset{a}{\operatorname{argmax}}\, Q_k(s,a) \tag{24}$$

So now we can build a recursive implementation to find the best action for a given state. To make the recursion well defined, make sure that $V_0(s) = 0$. See Algorithm 2 pseudocode in mdp course notes. No need for a huge decision tree like in the Expectimax. Instead this is a simple table that is cheap and quick to populate.

2.3.2 Infinite Horizon

Here, we discount rewards for increasingly distant horizons. γ is given as a parameter.

$$u = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t), \text{ where} \tag{25}$$

$$\gamma \in (0, 1) \tag{26}$$

Unlike finite horizon, in this case we move forward in time. So the immediate reward (not Q_k, just Q and just π):

$$Q(s, a) = R(s, a) + \gamma \sum_{s'} p(s'|s, a) V(s') \tag{27}$$

$$V(s) = \max_a Q(s, a) = Q(s, \pi^*(s)) \tag{28}$$

$$\pi^*(s) = \operatorname*{argmax}_a Q(s, a) \tag{29}$$

See CS229 Notes12 on Reinforcement Learning for a good discussion about this.

See policyiter class notes, algorithm 1.

2.4 Policy Iteration

Policy Iteration takes advantage of Linear Algebra to propagate information. Unlike Value Iteration, it does not need to wait until value convergence or the need to specify an ε. "You don't really care about the value function - you care about what to do next."

See algorithm 2 in policyiter Class Notes.

Use π as an index into the reward matrix to determine the next appropriate action.

$$r_s^\pi = R_{s,\pi_s} \text{or as a vector:} \tag{30}$$

$$\mathbf{r}^\pi \in \mathbb{R}^N \tag{31}$$

Likewise, the policy could applied as an index into the transition matrix:

$$\mathbf{P}^{\pi} = \begin{bmatrix} P_{1,1}^{(\pi_1)} & P_{1,2}^{(\pi_1)} & \dots & P_{1,N}^{(\pi_1)} \\ P_{2,1}^{(\pi_2)} & P_{2,2}^{(\pi_2)} & \dots & P_{2,N}^{(\pi_2)} \\ \vdots & \vdots & \ddots & \vdots \\ P_{N,1}^{(\pi_N)} & P_{N,2}^{(\pi_N)} & \dots & P_{N,N}^{(\pi_N)} \end{bmatrix}$$

To compute a new value iteration:

$$\mathbf{V} \leftarrow \mathbf{r}^{\pi} + \gamma \mathbf{P}^{\pi} \mathbf{V} \tag{32}$$

$$V(s) \leftarrow R(s, \pi(s)) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) \times V(s') \tag{33}$$

This recursion is how the value propagates downs the chain, and allows us how to solve for V in a fast and efficient way. (This corresponds to policyiter class notes, algorithm 2, line 5, "Solve system $V(s)$")

$$\mathbb{I}\mathbf{v} - \gamma \mathbf{P}^{\pi} \mathbf{v} = \mathbf{r}^{\pi} \tag{34}$$

$$(\mathbb{I} - \gamma \mathbf{P}^{\pi}) \mathbf{v} = \mathbf{r}^{\pi} \tag{35}$$

$$\mathbf{v} = (\mathbb{I} - \gamma \mathbf{P}^{\pi})^{-1} \mathbf{r}^{\pi} \tag{36}$$

Bad thing about Policy Iteration: solving the above matrix inverse is an $O(N^3)$ expensive operation. So each Policy Iteration is more expensive than Value Iteration, but in practice it is worth doing since it lets the information flow around the system is more efficiently than Value Iteration and will leverage common matrix libraries. (See discussion in policyiter, section 3 "Comparing VI and PI").

2.5 Reinforcement Learning

(see RL Course Notes)

Relaxing of the previous assumptions, Reinforcement Learning is about discovering the world with an:

- **Unknown Model** $P(s' \mid s, a)$
- **Unknown Reward** $R(s, a)$

and then **plan** as these are discovered.

Trade off of **exploration** vs **exploitation**. How much time should an agent invest in learning the world vs using gained knowledge? Initially, more time would be spent in exploration. If the agent will die soon, it should spend more time exploiting.

Two types of RL:

- **Model Based RL**. Try to understand the underlying dynamics of the world.
- **Model Free RL**. learn what to do without understanding the dynamics. Learn how to act. "Don't solve a harder problem than the problem at hand." Can approximate the state space that are appealing.

2.5.1 Model Based RL

$N_{s,a}$: Track how often we took action a in state s.

$R_{s,a}^{total}$: total reward from doing a in state s.

$N_{s,a,s'}$: transition model - how often did we "land" in state s' after taking action a in state s.

Build up the model by taking these ratios:

$$\hat{R}_{s,a} = \frac{R_{s,a}^{total}}{N_{s,a}} \tag{37}$$

$$\hat{P}_{s,s'}^{(a)} = \frac{N_{s,a,s'}}{N_{s,a}} \tag{38}$$

so we can now plan using Value or Policy Iteration.

To start, a prior could be seeded rather than 0 for the initial counts. That prior would be down weighted as evidence is collected.

Assumes that the reward is always coming from the same *distribution* but not necessarily the same reward every time.

If we didn't explore, then these counts would stay small. Balance greed vs learning.

The most common method of balancing is ε greedy. Most of the time, the optimal thing is done, but a certain ε percentage of time exploration is performed. There should be a schedule for ε so it decreases as everything has been explored.

Building the policies and doing planning is not free (recall $O(N^3)$). So a new policy or value iteration should not necessarily be done every action. Memoize the policy to save recalculating time.

2.5.2 Model Free RL

Typically based on *Q-learning*, which tells you optimal action to take in any given state. Provides policy without understanding dynamics or rewards.

"Why store a model, when we can just keep asking 'The World' for a noisy estimate? Kind of like a stochastic gradient descent."

From Bellman, we have a recursion.

$$Q_{s,a} = R_{s,a} + \gamma \sum_{s'} p(s' \mid s, a) \max_{a'} Q_{s',a'} \tag{39}$$

$$= R_{s,a} + \gamma \mathbb{E}_{s'} \left[\max_{a'} Q_{s',a'} \right] \tag{40}$$

$$= \mathbb{E}_{s'} \left[R_{s,a} + \gamma \max_{a'} Q_{s',a'} \right] \tag{41}$$

As the world gives us updated sample rewards and actions, they are used to update Q.

$$Q_{s,a} \leftarrow Q_{s,a} + \alpha \left[(r - \gamma \max_{a'} Q_{s',a'}) - Q_{s,a} \right]$$

where:

r is the reward we actually got

α is a weight learning rate placed on the new values ($\in [0, 1)$).

When dealing with very large state spaces, we can use a function approximator from functions of the state to get Q values. The approximator can be a linear regressor or any other function we've already seen

Assuming a huge state space: e.g. $s \in \mathbb{R}$, approximate Q:

$$\hat{Q}(s, a) = w_0^{(a)} + \phi(s)^\intercal \mathbf{w}^{(a)}$$

We're just doing gradient descent on the Q's so we can find the weights to provide a "good enough" Q function.

(DeepMind basically used Deep Learning + Model Free RL to solve their Atari game player).

2.6 Partially Observable MDP

(See pomdp Class Notes)

POMDP relaxes a different set of assumptions than Reinforcement Learning. (Recall: RL assumes you do not know the model, transition, reward function and dynamics). In POMDP, the current state is unknown (hence the term *Partially Observable*). In POMDP, the transition and reward functions are known. Only noisy observations are provided.

This table summarizes the differences:

Item	**MDP**	**RL**	**POMDP**
State	Known	Known	Unknown
Actions	Known	Known	Known
Transition Model	Known	Unknown	Known
Reward	Known	Unknown	Known
Policy	Unknown	Unknown	Unknown

Not knowing the state is much harder than not knowing Transition Model and Reward. Knowing the full history may be necessary so the amount needed to store might grow without bound. POMDPs may never converge, unlike RL where the world will eventually be understood.

This is the most realistic case compared to MDP and RL. Observations are always noisy in the real world. "You don't have a perfect model of your chess opponent. The odometry on a robot isn't perfect. You check the weather before leaving the house – this is a real life information gathering action."

So starting like MDP:

States: $\mathcal{S} = \{1, 2, 3, ...N\}$

Actions: $\mathcal{A} = \{1, 2, 3, ...M\}$

Transition Model $P(s' \mid s, a)$

Reward Function: $R(s, a)$

POMDP has a discrete set of *observations* and an *observation model*:

Observations : $O = \{1, 2, 3, ..., J\}$

Observation model: $p(O_j \mid S_n)$.

"We are going to take some action and get some observation. We're going to use that observation and the history of observations and actions to come up with a distribution of what state we are actually in." As observations (and actions) accumulate, we determine the probability of the current state. For now, we assume we know the Observation Model. (In practice, we'll never be given the Observation Model in the real world. Real observations are likely not discrete).

2.6.1 Belief State

Bayes Rule will provide the means to go from observation distribution to the possible state. *Belief state* is an estimate – a Bernoulli distribution – over a set of possible states.

Belief State: $B \in [0,1]$; $b(s_t) = P(s_t \mid o_1, \ldots, o_t, a_1, \ldots, a_{t-1})$

Three approaches used to solve POMDPs (using the assumption that this is a discrete belief state).

1. *piecewise linearly convex* different regions of the $[0,1]$ state space are "owned" by different linear functions. Over time, some of these functions can be pruned out. FMI, see lecture 19 video, 42:00.
2. *point approximation* Some observations may be very informative (and not noisy) so simply use this observation rather than the distribution.
3. *function approximation.* Use regression or a neural net as a different mechanism to determine state in lieu of distribution.

2.6.2 Policy Iteration Methods

Finding optimal policy is too hard so approximate the policy.

Finite Memory Policy keep only the last K observations to avoid overloading memory. $\mathcal{S} = O \times O \times O \ldots K$ times.

Finite state controller use a finite state machine to model the states of the agent based on the observations. Nodes represent the state, with edges as actions. A small set of actions can represent the policy. If finite state machine is in a long line (as opposed to a spread out graph), then moving to one end of the line representing being more certain of the state. Being in the middle means the current state is more ambiguous.

Modifying the environment to store gathered information is an example of *stigmergy*. Natural examples of this include ants leaving pheromones along a path, or even a person leaving "to-do notes" around. Future observations are modified by the agent's impact on the environment. Intelligent agents working together need to learn how store and share information.

2.7 Sources

1. Lecture 16, April 2, 2014 MDP
2. Lecture 17, April 7, 2014 Value Iteration

3. Lecture 18, April 9, 2014 Policy Iteration
4. Lecture 19, April 14, 2014 POMDP
5. Course notes - MDP
6. Course notes - policyiter
7. Course notes - RL
8. Course notes - POMDP
9. Bishop 9.0-9.2
10. Section 9 review
11. Section 10 review
12. CS229 Reinforcement Learning and Control

3 Expectation Maximization

3.1 Mixture Models

Mixture models is a latent variable model. This is revisiting clustering. Recall K-means clustering. Lloyd's algorithm is a special case of max likelihood learning in a Gaussian mixture model.

Data: $\{x_n\}_{n=1}^N$ no labels, just features $x_n \in \mathbb{R}^D$

K Clusters: a set of mean vectors $\{\mu_k\}_{k=1}^K$, where $\mu_k \in \mathbb{R}^D$. Cluster centers so they need to look like the data.

Responsibility vectors, one hot coded, all zeros except for the cluster it belongs to.
$\{\mathbf{r}_n\}_{n=1}^N$,where $r_n = [0\,0\,0\,0\,0\,1\ldots 0]^\intercal$

Loss function, or distortion measure: (how good are each of the cluster centers at reconstructing the data they belong to)

$$\mathbf{J}(\{\mu_k\},\{\mathbf{r}_n\}) = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}||\mathbf{x}_n - \mu_k||_2^2$$

use r_nk to select the slice we care about and penalize the data that is far from the cluster center.

Recall: Lloyd's algorithm is a coordinate descent in the r_{nk}'s and the μ's.

Update r_{nk}:

$$r_{nk} = \begin{cases} 1, & \text{if } k = \text{argmin}_{k'} ||\mathbf{x}_n - \mu_k||_2^2 \\ 0, & \text{otherwise} \end{cases}$$

Update μ_k:

$$\mu_k = \frac{\sum_{n=1}^{N} r_{nk}\mathbf{x}_n}{\sum_{n=1}^{N} r_{nk}}$$

A datum belongs to only one cluster even if ambiguous. It can be made more precise by a *soft K-means* clustering.

3.1.1 Soft K-means

Instead of one-hot coding, make the responsibility vector "soft" r sum to 1. Each element will be a proportion of belonging to that cluster.

$$r_{nk} \geq 0, \sum_{k=1}^{K} r_{nk} = 1$$

Need an update to Lloyd's algorithm to set r_{nk}. If distance is close, r_{nk} will be close to 1. As distance is bigger, r_{nk} will be smaller:

$$r_{nk} = \frac{\exp\left\{-\frac{\beta}{2}||\mathbf{x}_n - \mu_k||_2^2\right\}}{\sum_{k'=1}^{K} \exp\left\{-\frac{\beta}{2}||\mathbf{x}_n - \mu_k||_2^2\right\}}$$

The updates of μ_k stay the same. β is a parameter to define the "softness" or "hardness". If β is large, a single term will dominate. If β is small, the distribution will be flatter.

3.1.2 Mixture of Gaussians

(See Bishop 9.2)

The underlying objective function for soft K-means. It permits a "fancier" distribution from a set of simpler distributions. Created a weighted sum of Gaussians.

$$\pi_1 + \pi_2 + \pi_3 = 1 \tag{42}$$

$$p(x) = \pi_1\mathcal{N}(\mu_1, \sigma_1^2) + \pi_2\mathcal{N}(\mu_2, \sigma_2^2) + \pi_3\mathcal{N}(\mu_3, \sigma_3^2) \tag{43}$$

Could be more complicated than Gaussians, like dice rolls, text, Poisson distribution...provides modularity.

More generally, there is a set of component distributions (or densities or PMF's), $p(x \mid \theta)$

K mixture components: $\{\theta_k\}_{k=1}^K$

K mixture weights: $\{\pi_k\}_{k=1}^K, \pi_k \geq 0, \sum_{k=1}^K \pi_k = 1$

Resulting distribution:

$$p(x \mid \{\theta_k, \pi_k\}_{k=1}^K) = \sum_{k=1}^K \pi_k p(x \mid \theta_k)$$

Any continuous density could be modeled like this to an arbitrarily complex level.

3.1.3 Generative Latent Variable Models

To generate an x from a mixture model

1. Pick a k from π (decide what member we will belong to)
2. Pick one x from the k^{th} component density, $x \sim p(x' \mid \theta_k)$

Can used to discover a *latent variable* or hidden structure in data. Unsupervised learning could considered like supervised learning without the labels. A lot of models, like PCA and Factor Analysis, could be based on a generative underlying dynamic.

Start with low dimensional data, and then find the underlying higher dimensional data, e.g. topic modeling, NLP (where a topic is a distribution over words). Recommender systems like Probabilistic Matrix Factorization, is also related. Applicable to Social and Network Analysis or digital humanities. "Map what you can see to a model of what you can't see in order to understand the dynamics."

Latent Dirichlet Allocation (LDA) helps find structure in corpora, images, etc.

3.1.4 Expectation Maximization Intro

We'll use θ as above consisting of K means and covariances: $\{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^K$

Mixture Weights: π_k

The density will be a weighted sum of these Gaussian models, known as a *Gaussian Mixture Model*:

$$p(\mathbf{x} \mid \pi, \{\mu_k, \Sigma_k\}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x} \mid \mu_k, \Sigma_k)$$

Maximum Likelihood is the best general way of fitting data and finding μ and Σ.

$$\ln p(\{\mathbf{x}_n\} \mid \pi, \{\mu_k, \Sigma_k\} = \sum_{n=1}^{N} \ln \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n \mid \mu_k, \Sigma_k)$$

which is a mess... due to $\ln \sum$. Taking the gradient descent would be a problem because the π's are constrained (summed to 1) and Σ must be symmetric positive definite. Instead use *Expectation Maximization*.

Expectation Maximization (EM) is a two step process that creates and revises estimates of 1) latent variables explicitly ("E-step") then 2) improves parameters ("M-step").

3.1.5 Simplest Possible Gaussian Mixture Model

Assumptions:

Known $\pi_k = \frac{1}{k}$ which is uniform

Known $\mathbf{\Sigma} = \frac{1}{\beta}\mathbb{I}_D$ (also uniform)

Unknown $\{\mu_k\}_{k=1}^K$

Log likelihood:

$$\ln p(\{\mathbf{x}_n\} \mid \{\mu_k\}_{k=1}^K) = \sum_{n=1}^{N} \ln \sum_{k=1}^{K} \frac{1}{k} \mathcal{N}(\mathbf{x}_n \mid \mu_k, \frac{1}{\beta}\mathbb{I}_D)$$

Same variance, isotropic (spherical) Gaussian.

(For the derivation, see lecture 20, 1:12)

After gradient descent, the resulting softened Lloyd's Algorithm will use:

$$\mu_k = \frac{\sum_n r_{nk}\mathbf{x}_n}{\sum_n r_{nk}}$$

3.2 Expectation Maximization in Depth

"EM is a core algorithm of probabilistic modeling. Perform maximum likelihood learning in probabilistic models that have latent variables. [Effectively EM is] coordinate ascent of the likelihood while introducing an approximation that makes things tractable and easy."

Recall: Most EM uses Gaussian Mixture Models, which consist of

K mixture components: $\{\theta_k\}_{k=1}^K$

K mixture weights: $\{\pi_k\}_{k=1}^K, \pi_k \geq 0, \sum_{k=1}^K \pi_k = 1$

(See Lecture 21, 8:00 for a diagram)

3.3 Hidden Markov Models

3.4 Sources

1. Lecture 20 April 16, 2014 Mixture Models
2. Lecture 21 Expectation Maximization
3. Lecture 22 Hidden Markov Models
4. Bishop 9.3
5. Bishop 13.0-13.2
6. Section 11
7. CS229 Mixture of Gaussians
8. CS229 EM Notes