Recommended reading:
Bishop, Chapters 3.6, 8.6
Shlens PCA tutorial
Wall et al. 2003 (PCA applied to gene expression data)
Announcements

- Thursday’s recitation:
  - Semi-supervised learning and PCA
  - (Of course), final will cover all material including last part of the semester (semi-supervised learning, dimensionality reduction, reinforcement learning…)
Lower dimensional projections

- Rather than picking a subset of the features, we can generate new features that are combinations of existing features.

Let’s see this in the unsupervised setting

- just \( X \), but no \( Y \)
Liner projection and reconstruction

\[ U = a_1 x_1 + a_2 x_2 + a_3 x_3 + \ldots \]

project into 1-dimension

only have \( z \), what was \((x_1, x_2)\)

reconstruction: only know \( z_1 \), and \( u_1 \)
Principal component analysis – basic idea

- Project n-dimensional data into k-dimensional space while preserving information:
  - e.g., project space of 10000 words into 3-dimensions
  - e.g., project 3-d into 2-d

- Choose projection with minimum reconstruction error
Linear projections, a review

- Project a point into a (lower dimensional) space:
  - **point**: \( x = (x_1, \ldots, x_n) \)
  - **select a basis** – set of basis vectors – \( (u_1, \ldots, u_k) \)
    - we consider orthonormal basis: \( u_i \cdot u_i = 1, \text{ and } u_i \cdot u_j = 0 \text{ for } i \neq j \)
  - **select a center** – \( \bar{x} \), defines offset of space
  - **best coordinates** in lower dimensional space defined by dot-products: \( (z_1, \ldots, z_k), \quad z_i = (x - \bar{x}) \cdot u_i \)
    - minimum squared error
PCA finds projection that minimizes reconstruction error

- Given m data points: \( x^i = (x_1^i, \ldots, x_n^i), i=1\ldots m \)
- Will represent each point as a projection:

\[
\hat{x}^i = \bar{x} + \sum_{j=1}^{k} z_j^i u_j
\]

where:
\[
\bar{x} = \frac{1}{m} \sum_{i=1}^{m} x^i
\]
and
\[
z_j^i = x^i \cdot u_j
\]

- PCA:

Given \( k \leq n \), find \((u_1, \ldots, u_k)\)

minimizing reconstruction error:

\[
\text{error}_k = \sum_{i=1}^{m} (x^i - \hat{x}^i)^2
\]
Minimizing reconstruction error and eigen vectors

- Minimizing reconstruction error equivalent to picking orthonormal basis \((u_1, \ldots, u_n)\) minimizing:

\[
\text{error}_k = \sum_{j=k+1}^{n} u_j^T \Sigma u_j
\]

- Eigen vector:

\[
u: \quad \Sigma u = \lambda u \quad \Rightarrow \quad u^T \Sigma u = \lambda u^T u = \lambda
\]

- Minimizing reconstruction error equivalent to picking \((u_{k+1}, \ldots, u_n)\) to be eigen vectors with smallest eigen values

\[
\begin{align*}
\text{sort!} & \quad u_1, u_2, \ldots \quad \text{take eigen vectors} \\
\lambda_1, \lambda_2 & \quad \text{values of } \Sigma \\
\lambda_1 > \lambda_{i+1} & \quad \text{pick top k} \\
\end{align*}
\]

\[
\Rightarrow \min \sum_{j=k+1}^{n} u_j^T \Sigma u_j
\]

\[
= \min \sum_{j=k+1}^{n} \lambda_j
\]

\[
\Rightarrow \text{ignore vectors with low eigen values}
\]
Basic PCA algorithm

- Start from m by n data matrix $X$
- **Recenter**: subtract mean from each row of $X$
  - $X_c \leftarrow X - \bar{X}
  - $\bar{X} = \frac{1}{m} \sum_i x_i$
  - $X_c$ has mean $\emptyset$
  - $x^u = (x^u_1, ..., x^u_n)$
  - $x^v = (x^v_1, ..., x^v_n)$
- Compute covariance matrix:
  - $\Sigma \leftarrow X_c^T X_c$
  - $\sigma_{ij} = \frac{1}{m} \sum_u \sum_v x^u_i \cdot x^v_j$
- Find **eigen vectors and values** of $\Sigma$
- **Principal components**: k eigen vectors with highest eigen values
  - New features are linear combination of old features
  - $u^j = \sum_i a_i x_i$
PCA example

\[ \hat{x}^i = \bar{x} + \sum_{j=1}^{k} z_j^i u_j \]
PCA example – reconstruction

\[ \hat{x}^i = \bar{x} + \sum_{j=1}^{k} z_j^i u_j \]

only used first principal component

![Figure 1](image)
Eigenfaces [Turk, Pentland ’91]

- Input images:
- Principal components:

\[ \chi^i = (\text{pixel}) \]
Eigenfaces reconstruction

- Each image corresponds to adding 8 principal components:
Relationship to Gaussians

- PCA assumes data is Gaussian
  - \( x \sim N(\mu; \Sigma) \)

- Equivalent to weighted sum of simple Gaussians:
  \[
  x = \bar{x} + \sum_{j=1}^{n} z_j u_j; \quad z_j \sim N(0; \sigma_j^2)
  \]

- Selecting top \( k \) principal components equivalent to lower dimensional Gaussian approximation:
  \[
  x \approx \bar{x} + \sum_{j=1}^{k} z_j u_j + \varepsilon; \quad z_j \sim N(0; \sigma_j^2)
  \]

- \( \varepsilon \sim N(0; \sigma^2) \), where \( \sigma^2 \) is defined by error \( k \)
Scaling up

- Covariance matrix can be really big!
  - $\Sigma$ is $n$ by $n$
  - 10000 features $\rightarrow |\Sigma| = 10,000 \times 10,000$
  - finding eigenvectors is very slow…

- Use singular value decomposition (SVD)
  - finds to $k$ eigenvectors
  - great implementations available, e.g., Matlab svd
SVD

Write $X = U S V^T$

- $X \leftarrow$ data matrix, one row per datapoint
- $U \leftarrow$ weight matrix, one row per datapoint – coordinate of $x^i$ in eigenspace
- $S \leftarrow$ singular value matrix, diagonal matrix
  - in our setting each entry is eigenvalue $\lambda_j$
- $V^T \leftarrow$ singular vector matrix
  - in our setting each row is eigenvector $v_j$
PCA using SVD algorithm

- Start from m by n data matrix $X$
- **Recenter**: subtract mean from each row of $X$
  - $X_c \leftarrow X - \bar{X}$
- Call SVD algorithm on $X_c$ – ask for **k** singular vectors
- **Principal components**: $k$ singular vectors with highest singular values (rows of $V^T$)
  - **Coefficients** become:
    - $x^i = \bar{x} + \sum z_j u_j \cdot v_j$
    - $z_j = (x^i - \bar{x}) \cdot v_j$
    - $z_j = u_j \cdot x_j$
Using PCA for dimensionality reduction in classification

- Want to learn $f : \mathbf{X} \rightarrow Y \in \{ +1, -1 \}$
  - $\mathbf{X} = \langle X_1, \ldots, X_n \rangle$
  - but some features are more important than others

- **Approach**: Use PCA on $\mathbf{X}$ to select a few important features

\[
\text{Instead } f(x) \rightarrow y
\]

\[
\text{Learn } f(\mathbf{z}_1, \ldots, \mathbf{z}_k) \rightarrow y
\]

\[
\mathbf{x} \approx \mathbf{x} + \sum_{i=1}^{k} \mathbf{z}_i \cdot w_j
\]

Suppose learn linear classifier $(\mathbf{w}, b)$

Decision rule? for a test point $\mathbf{x}$:

- Compute projection of $\mathbf{x}_{\text{new}}$: $\mathbf{z}_{\text{new}} = (\mathbf{x}_{\text{new}} - \mathbf{x}) \cdot w_j$

- $\sum_{j} \mathbf{z}_{\text{new}} \cdot w_j + b \geq 0$
PCA for classification can lead to problems...

- Direction of maximum variation may be unrelated to "discriminative" directions:

- PCA often works very well, but sometimes must use more advanced methods:
  - e.g., Fisher linear discriminant
What you need to know

- Dimensionality reduction
  - why and when it’s important

- Simple feature selection

- Principal component analysis
  - minimizing reconstruction error
  - relationship to covariance matrix and eigenvectors
  - using SVD
  - problems with PCA
Markov Decision Processes (MDPs)

Machine Learning – 10701/15781
Carlos Guestrin
Carnegie Mellon University
April 26th, 2006

Reading:
Kaelbling et al. 1996 (see class website)
Announcements

■ Project:
  ■ Poster session: Friday May 5th 2-5pm, NSH Atrium
    ■ please arrive a little early to set up
  ■ Paper: Monday May 8th by noon to Monica Hopes – Wean Hall 4616
    ■ maximum of 8 pages, NIPS format

■ FCEs!!!!
  ■ Please, please, please, please, please, please, please, please give us your feedback, it helps us improve the class! 😊
    ■ http://www.cmu.edu/fce
Thus far this semester

- Regression: $X \rightarrow \mathbb{R}$

- Classification: $X \rightarrow Y \in \{1, 2, 3, 4\}$

- Density estimation: $X \rightarrow Y \in [0, 1]$
Learning to act

- Reinforcement learning
- An agent
  - Makes sensor observations
  - Must select action
  - Receives rewards
    - positive for “good” states
    - negative for “bad” states

[Ng et al. '05]
Learning to play backgammon
[Tesauro '95]

- Combines reinforcement learning with neural networks
- Played 300,000 games against itself
- Achieved grandmaster level!
Roadmap to learning about reinforcement learning

- When we learned about Bayes nets:
  - First talked about formal framework:
    - representation
    - inference \( P(X|\epsilon) \)
  - Then learning for BNs

- For reinforcement learning:
  - Formal framework
    - Markov decision processes
  - Then learning
Real-time **Strategy** Game

- **Peasants** collect resources and build
- **Footmen** attack enemies
- **Buildings** train peasants and footmen
States and actions

- **State space:**
  - Joint state \( x \) of entire system
    \[ x \rightarrow \text{how many peasants, footmen...} \]
    \[ \text{where they are} \]
    \[ \text{how much gold, wood...} \]

- **Action space:**
  - Joint action \( a = \{a_1, \ldots, a_n\} \) for all agents
    \[ a \rightarrow \text{peasant I collect wood} \]
    \[ \text{"I " gold} \]
    \[ \text{footmen I go west} \]
States change over time

- Like an HMM, state changes over time
- Next state depends on current state and action selected
  - e.g., action=“build castle” likely to lead to a state where you have a castle
- Transition model:
  - Dynamics of the entire system $P(x'|x,a)$
  - $P(x'|x,a) = P(x^{t+1}|x^t,a)$
    - **HMM**: $P(x^{t+1}|x^t)$
    - **MDP**: $P(x^{t+1}|x^t,a)$
Some states and actions are better than others

- Each state $x$ is associated with a reward
  - positive reward for successful attack
  - negative for loss

- Reward function:
  - Total reward $R(x)$

$$R = \sum_{x} \left[ \begin{array}{c} 2.8 \\ -10000 \\ 300 \end{array} \right]$$
Discounted Rewards

An assistant professor gets paid, say, 20K per year. How much, in total, will the A.P. earn in their life?

\[ 20 + 20 + 20 + 20 + 20 + \ldots = \text{Infinity} \]

What’s wrong with this argument?
Discounted Rewards

“A reward (payment) in the future is not worth quite as much as a reward now.”

- Because of chance of obliteration
- Because of inflation

Example:

Being promised $10,000 next year is worth only 90% as much as receiving $10,000 right now.

Assuming payment $n$ years in future is worth only $(0.9)^n$ of payment now, what is the AP’s Future Discounted Sum of Rewards?
Discount Factors

People in economics and probabilistic decision-making do this all the time.

The “Discounted sum of future rewards” using discount factor $\gamma$ is

$(\text{reward now}) + \gamma (\text{reward in 1 time step}) + \gamma^2 (\text{reward in 2 time steps}) + \gamma^3 (\text{reward in 3 time steps}) + \ldots$ (infinite sum)

\[= \frac{20}{1-\gamma} = \frac{-20}{0.1} = -200\]
Define:

\[ V_A = \text{Expected discounted future rewards starting in state A} \]
\[ V_B = \text{Expected discounted future rewards starting in state B} \]
\[ V_T = \text{Expected discounted future rewards starting in state T} \]
\[ V_S = \text{Expected discounted future rewards starting in state S} \]
\[ V_D = \text{Expected discounted future rewards starting in state D} \]

How do we compute \( V_A, V_B, V_T, V_S, V_D \) ?
Computing the Future Rewards of an Academic

Assume Discount Factor $\gamma = 0.9$

- $V_B = 60 + \gamma [0.6 V_B + 0.2 V_T + 0.2 V_D]$
- $V_S = 10 + \gamma [0.7 V_S + 0.3 V_D]$
- $V_T = 400 + \gamma [0.3 V_D + 0.7 V_T]$

$V_D = 0$

$V_T = \frac{400}{1 - 0.7 \gamma}$
Joint Decision Space

Markov Decision Process (MDP) Representation:

- **State space:**
  - Joint state $x$ of entire system

- **Action space:**
  - Joint action $a = \{a_1, \ldots, a_n\}$ for all agents

- **Reward function:**
  - Total reward $R(x,a)$
    - sometimes reward can depend on action

- **Transition model:**
  - Dynamics of the entire system $P(x'|x,a)$
Policy

Policy: \( \pi(x) = a \)

At state \( x \), action \( a \) for all agents

\( \pi(x_0) = \text{both peasants get wood} \)

\( \pi(x_1) = \text{one peasant builds barrack, other gets gold} \)

\( \pi(x_2) = \text{peasants get gold, footmen attack} \)
Value of Policy

Value: \( V_\pi(x) \)

Expected long-term reward starting from \( x \)

\[
V_\pi(x_0) = \mathbb{E}_\pi[R(x_0) + \gamma R(x_1) + \gamma^2 R(x_2) + \gamma^3 R(x_3) + \gamma^4 R(x_4) + \cdots]
\]

Future rewards discounted by \( \gamma \in [0,1) \)
Computing the value of a policy

$V_\pi(x_0) = E_\pi[R(x_0) + \gamma R(x_1) + \gamma^2 R(x_2) + \gamma^3 R(x_3) + \gamma^4 R(x_4) + \cdots]$  

- **Discounted value of a state:**
  - value of starting from $x_0$ and continuing with policy $\pi$ from then on

  $$V_\pi(x_0) = E_\pi[R(x_0) + \gamma R(x_1) + \gamma^2 R(x_2) + \gamma^3 R(x_3) + \cdots]$$

  $$= E_\pi[\sum_{t=0}^{\infty} \gamma^t R(x_t)]$$

- **A recursion!**
Computing the value of a policy 1 – the matrix inversion approach

\[ V_\pi(x) = R(x) + \gamma \sum_{x'} P(x' | x, a = \pi(x)) V_\pi(x') \]

- Solve by simple matrix inversion:
Computing the value of a policy 2 – iteratively

\[ V_\pi(x) = R(x) + \gamma \sum_{x'} P(x' \mid x, a = \pi(x)) V_\pi(x') \]

- If you have 1,000,000 states, inverting a 1,000,000x1,000,000 matrix is hard!
- Can solve using a simple convergent iterative approach: (a.k.a. dynamic programming)
  - Start with some guess \( V_0 \)
  - Iteratively say:
    - \( V_{t+1} = R + \gamma P_\pi V_t \)
  - Stop when \( \|V_{t+1} - V_t\|_\infty \leq \varepsilon \)
    - means that \( \|V_\pi - V_{t+1}\|_\infty \leq \varepsilon/(1-\gamma) \)
But we want to learn a Policy

- So far, told you how good a policy is…
- But how can we choose the best policy???

- Suppose there was only one time step:
  - world is about to end!!!
  - select action that maximizes reward!

Policy: \( \pi(x) = a \)

At state \( x \), action \( a \) for all agents

\( \pi(x_0) = \) both peasants get wood

\( \pi(x_1) = \) one peasant builds barrack, other gets gold

\( \pi(x_2) = \) peasants get gold, footmen attack
Another recursion!

- Two time steps: address tradeoff
  - good reward now
  - better reward in the future
Unrolling the recursion

- Choose actions that lead to best value in the long run
  - Optimal value policy achieves optimal value $V^*$

$$V^*(x_0) = \max_{a_0} R(x_0, a_0) + \gamma E_{a_0} \left[ \max_{a_1} R(x_1) + \gamma^2 E_{a_1} \left[ \max_{a_2} R(x_2) + \cdots \right] \right]$$
Bellman equation

- Evaluating policy $\pi$:

$$V_\pi(x) = R(x) + \gamma \sum_{x'} P(x' \mid x, a = \pi(x)) V_\pi(x')$$

- Computing the optimal value $V^*$ - Bellman equation

$$V^*(x) = \max_a R(x, a) + \gamma \sum_{x'} P(x' \mid x, a) V^*(x')$$
Optimal Long-term Plan

Optimal value function $V^*(x)$

Optimal Policy: $\pi^*(x)$

$$Q^*(x, a) = R(x, a) + \gamma \sum_{x'} P(x'|x, a)V^*(x')$$

Optimal policy:

$$\pi^*(x) = \arg \max_a Q^*(x, a)$$
Interesting fact – Unique value

\[ V^*(x) = \max_a R(x, a) + \gamma \sum_{x'} P(x'|x, a)V^*(x') \]

- **Slightly surprising fact**: There is only one \( V^* \) that solves Bellman equation!
  - there may be many optimal policies that achieve \( V^* \)
- **Surprising fact**: optimal policies are good everywhere!!!

\[ V_{\pi^*}(x) \geq V_{\pi}(x), \quad \forall x, \; \forall \pi \]
Solving an MDP

Solve Bellman equation

Optimal value $V^*(x)$

Optimal policy $\pi^*(x)$

$$V^*(x) = \max_a R(x, a) + \gamma \sum_{x'} P(x'| x, a) V^*(x')$$

Bellman equation is non-linear!!!

Many algorithms solve the Bellman equations:

- Policy iteration [Howard ‘60, Bellman ‘57]
- Value iteration [Bellman ‘57]
- Linear programming [Manne ‘60]
- …
Value iteration (a.k.a. dynamic programming) – the simplest of all

\[ V^*(x) = \max_a R(x, a) + \gamma \sum_{x'} P(x'| x, a)V^*(x') \]

- Start with some guess \( V_0 \)
- Iteratively say:
  \[ V_{t+1}(x) = \max_a R(x, a) + \gamma \sum_{x'} P(x'| x, a)V_t(x') \]
- Stop when \( \|V_{t+1} - V_t\|_\infty \leq \epsilon \)
  - \( \epsilon \) means that \( \|V^* - V_{t+1}\|_\infty \leq \epsilon/(1-\gamma) \)
A simple example

You run a startup company.

In every state you must choose between Saving money or Advertising.
Let’s compute $V_t(x)$ for our example

\[ V_{t+1}(x) = \max_a R(x, a) + \gamma \sum_{x'} P(x'|x, a)V_t(x') \]
Let’s compute $V_t(x)$ for our example

\[
V_{t+1}(x) = \max_a R(x, a) + \gamma \sum_{x'} P(x'|x, a)V_t(x')
\]
Policy iteration – Another approach for computing $\pi^*$

- Start with some guess for a policy $\pi_0$
- Iteratively say:
  - evaluate policy: $V_t(x) = R(x, a = \pi_t(x)) + \gamma \sum_{x'} P(x'|x, a = \pi_t(x))V_t(x')$
  - improve policy: $\pi_{t+1}(x) = \max_a R(x, a) + \gamma \sum_{x'} P(x'|x, a)V_t(x')$

- Stop when
  - policy stops changing
    - usually happens in about 10 iterations
  - or $||V_{t+1}-V_t||_\infty \leq \varepsilon$
    - means that $||V^*-V_{t+1}||_\infty \leq \varepsilon/(1-\gamma)$
Policy Iteration & Value Iteration: Which is best ???

It depends.

- Lots of actions? Choose Policy Iteration
- Already got a fair policy? Policy Iteration
- Few actions, acyclic? Value Iteration

Best of Both Worlds:

Modified Policy Iteration [Puterman]

...a simple mix of value iteration and policy iteration

3rd Approach

Linear Programming
LP Solution to MDP

Value computed by linear programming:

\[
\text{minimize: } \sum_x V(x) \geq R(x, a) + \gamma \sum_{x'} P(x'|x, a)V(x')
\]

- One variable \( V(x) \) for each state
- One constraint for each state \( x \) and action \( a \)
- Polynomial time solution
What you need to know

- What’s a Markov decision process
  - state, actions, transitions, rewards
  - a policy
  - value function for a policy
    - computing $V_\pi$

- Optimal value function and optimal policy
  - Bellman equation

- Solving Bellman equation
  - with value iteration, policy iteration and linear programming
Acknowledgment

- This lecture contains some material from Andrew Moore’s excellent collection of ML tutorials:
  - [http://www.cs.cmu.edu/~awm/tutorials](http://www.cs.cmu.edu/~awm/tutorials)