Dimensionality Reduction
PCA

Machine Learning – 10701/15781
Carlos Guestrin
Carnegie Mellon University
November 30th, 2009
Dimensionality reduction

- Input data may have thousands or millions of dimensions!
  - e.g., text data has

- **Dimensionality reduction**: represent data with fewer dimensions
  - easier learning – fewer parameters
  - visualization – hard to visualize more than 3D or 4D
  - discover "intrinsic dimensionality" of data
    - high dimensional data that is truly lower dimensional
Feature selection

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

- **Approach**: select subset of features to be used by learning algorithm
  - Score each feature (or sets of features)
  - Select set of features with best score
Simple greedy **forward** feature selection algorithm

- Pick a dictionary of features
  - e.g., polynomials for linear regression

- Greedy heuristic:
  - Start from empty (or simple) set of features $F_0 = \emptyset$
  - Run learning algorithm for current set of features $F_t$
    - Obtain $h_t$
  - Select **next best feature** $X_i$
    - e.g., $X_j$ that results in lowest cross-validation error learner when learning with $F_t \cup \{X_j\}$
  - $F_{t+1} \leftarrow F_t \cup \{X_i\}$
  - Recurse
Simple greedy **backward** feature selection algorithm

- Pick a dictionary of features
  - e.g., polynomials for linear regression

- Greedy heuristic:
  - Start from all features $F_0 = F$
  - Run learning algorithm for current set of features $F_t$
    - Obtain $h_t$
  - Select next worst feature $X_i$
    - e.g., $X_j$ that results in lowest cross-validation error learner when learning with $F_t - \{X_j\}$
  - $F_{t+1} \leftarrow F_t - \{X_i\}$
  - Recurse
Impact of feature selection on classification of fMRI data [Pereira et al. ’05]

<table>
<thead>
<tr>
<th>#voxels</th>
<th>mean</th>
<th>subjects</th>
<th>23B</th>
<th>329B</th>
<th>332B</th>
<th>424B</th>
<th>474B</th>
<th>496B</th>
<th>77B</th>
<th>86B</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.735</td>
<td></td>
<td>0.783</td>
<td>0.817</td>
<td>0.55</td>
<td>0.783</td>
<td>0.75</td>
<td>0.8</td>
<td>0.65</td>
<td>0.75</td>
</tr>
<tr>
<td>100</td>
<td>0.742</td>
<td></td>
<td>0.707</td>
<td>0.8</td>
<td>0.533</td>
<td>0.817</td>
<td>0.85</td>
<td>0.783</td>
<td>0.6</td>
<td>0.783</td>
</tr>
<tr>
<td>200</td>
<td>0.737</td>
<td></td>
<td>0.783</td>
<td>0.783</td>
<td>0.517</td>
<td>0.817</td>
<td>0.883</td>
<td>0.75</td>
<td>0.583</td>
<td>0.783</td>
</tr>
<tr>
<td>300</td>
<td>0.75</td>
<td></td>
<td>0.8</td>
<td>0.817</td>
<td>0.567</td>
<td>0.833</td>
<td>0.883</td>
<td>0.75</td>
<td>0.583</td>
<td>0.767</td>
</tr>
<tr>
<td>400</td>
<td>0.742</td>
<td></td>
<td>0.8</td>
<td>0.783</td>
<td>0.583</td>
<td>0.85</td>
<td>0.833</td>
<td>0.75</td>
<td>0.583</td>
<td>0.75</td>
</tr>
<tr>
<td>800</td>
<td>0.735</td>
<td></td>
<td>0.833</td>
<td>0.817</td>
<td>0.567</td>
<td>0.833</td>
<td>0.833</td>
<td>0.7</td>
<td>0.55</td>
<td>0.75</td>
</tr>
<tr>
<td>1600</td>
<td>0.698</td>
<td></td>
<td>0.8</td>
<td>0.817</td>
<td>0.45</td>
<td>0.783</td>
<td>0.833</td>
<td>0.633</td>
<td>0.5</td>
<td>0.75</td>
</tr>
<tr>
<td>all (~2500)</td>
<td>0.638</td>
<td></td>
<td>0.767</td>
<td>0.767</td>
<td>0.25</td>
<td>0.75</td>
<td>0.833</td>
<td>0.567</td>
<td>0.433</td>
<td>0.733</td>
</tr>
</tbody>
</table>

Table 1: Average accuracy across all pairs of categories, restricting the procedure to use a certain number of voxels for each subject. The highlighted line corresponds to the best mean accuracy, obtained using 300 voxels.

Voxels scored by p-value of regression to predict voxel value from the task.
Lower dimensional projections

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

Let’s see this in the unsupervised setting.

- just $X$, but no $Y$
Linear projection and reconstruction

project into 1-dimension

reconstruction: only know $z_1$, what was $(x_1, x_2)$
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:
      - $u_i \cdot u_i = 1$, and $u_i \cdot u_j = 0$ for $i \neq j$
  - **select a center** – $\bar{x}$, defines offset of space
  - **best coordinates** in lower dimensional space defined by dot-products: $(z_1, \ldots, z_k)$, $z_i = (x - \bar{x}) \cdot u_i$
    - minimum squared error
PCA finds projection that minimizes reconstruction error

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

\[
\hat{x}^i = \bar{x} + \sum_{j=1}^{k} z_j^i \mathbf{u}_j \quad \text{where:} \quad \bar{x} = \frac{1}{m} \sum_{i=1}^{m} \mathbf{x}^i \quad \text{and} \quad z_j^i = (\mathbf{x}^i - \bar{x}) \cdot \mathbf{u}_j
\]

- PCA:
  - Given \( k \cdot n \), find \((\mathbf{u}_1, \ldots, \mathbf{u}_k)\) minimizing reconstruction error:

\[
\text{error}_k = \sum_{i=1}^{m} (\mathbf{x}^i - \hat{x}^i)^2
\]
Understanding the reconstruction error

- Note that $\mathbf{x}^i$ can be represented exactly by n-dimensional projection:
  \[ \mathbf{x}^i = \bar{\mathbf{x}} + \sum_{j=1}^{n} z_j^i \mathbf{u}_j \]

- Rewriting error:

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

\[ z_j^i = (\mathbf{x}^i - \bar{\mathbf{x}}) \cdot \mathbf{u}_j \]

- Given $k<n$, find $(\mathbf{u}_1, \ldots, \mathbf{u}_k)$ minimizing reconstruction error:

\[ \text{error}_k = \sum_{i=1}^{m} (\mathbf{x}^i - \hat{\mathbf{x}}^i)^2 \]
Reconstruction error and covariance matrix

\[
error_k = \sum_{i=1}^{m} \sum_{j=k+1}^{n} [u_j \cdot (x^i - \bar{x})]^2
\]

\[
\Sigma = \frac{1}{m} \sum_{i=1}^{m} (x^i - \bar{x})(x^i - \bar{x})^T
\]
Minimizing reconstruction error and eigen vectors

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

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

- Eigen vector:

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

- Start from m by n data matrix $X$
- **Recenter**: subtract mean from each row of $X$
  - $X_c \leftarrow X - \bar{X}$
- Compute covariance matrix:
  - $\Sigma \leftarrow \frac{1}{m} X_c^T X_c$
- Find **eigen vectors and values** of $\Sigma$
- **Principal components**: $k$ eigen vectors with highest eigen values
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
Eigenfaces [Turk, Pentland ’91]

- Input images:
- Principal components:
Eigenfaces reconstruction

- Each image corresponds to adding 8 principal components:
Scaling up

- Covariance matrix can be really big!
  - $\Sigma$ is $n$ by $n$
  - 10000 features! $|\Sigma|$
  - finding eigenvectors is very slow…

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

Write $X = W S V^T$

- $X \leftarrow$ data matrix, one row per datapoint
- $W \leftarrow$ 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:
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

©2005-2009 Carlos Guestrin
Thus far this semester

- Regression:

- Classification:

- Density estimation:
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
  - 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

- Action space:
  - Joint action $a = \{a_1, \ldots, a_n\}$ for all agents
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 $P(x'|x,a)$
Some states and actions are better than others

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

- Reward function:
  - Total reward $R(\mathbf{x})$
Markov Decision Process (MDP) Representation

- **State space:**
  - Joint state $\mathbf{x}$ of entire system

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

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

- **Transition model:**
  - Dynamics of the entire system $P(\mathbf{x}'|\mathbf{x}, \mathbf{a})$
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

\[
\text{(reward now)} + \gamma \text{(reward in 1 time step)} + \gamma^2 \text{(reward in 2 time steps)} + \gamma^3 \text{(reward in 3 time steps)} + \ldots
\]

\[
\text{(infinite sum)}
\]
The Academic Life

Define:

\( V_A \) = Expected discounted future rewards starting in state A

\( V_B \) = Expected discounted future rewards starting in state B

\( V_T \) = " " " " " " " " " " T

\( V_S \) = " " " " " " " " " " S

\( V_D \) = " " " " " " " " " " D

How do we compute \( V_A, V_B, V_T, V_S, V_D \) ?

Assume Discount Factor \( \gamma = 0.9 \)
Computing the Future Rewards of an Academic

Assume Discount Factor $\gamma = 0.9$
Policy

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
Value of Policy

Value: $V_\pi(x)$

Expected long-term reward starting from $x$

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

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!
Simple approach for computing the value of a policy: Iteratively

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

- 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 \( \|V_{t+1} - V_t\|_\infty < \varepsilon \)
    - means that \( \|V_\pi - V_{t+1}\|_\infty < \varepsilon/(1-\gamma) \)
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!
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: $\pi^*(x)$

Optimal policy:

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

\[ V_{\pi^*}(x) \geq V_{\pi}(x), \ \forall x, \ \forall \pi \]
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 \( ||V_{t+1} - V_t||_\infty < \varepsilon \)
  - means that \( ||V^* - V_{t+1}||_\infty < \varepsilon/(1-\gamma) \)
A simple example

You run a startup company. In every state you must choose between Saving money or Advertising.

\[ \gamma = 0.9 \]

©2005-2009 Carlos Guestrin
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

$$V_{t+1}(x) = \max_a R(x, a) + \gamma \sum_{x'} P(x'|x, a)V_t(x')$$
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)