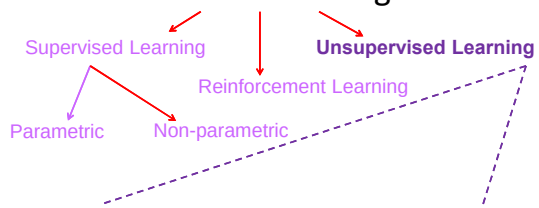
CSE 446
Dimensionality Reduction and PCA
Winter 2012

Slides adapted from Carlos Guestrin & Luke Zettlemoyer

Machine Learning

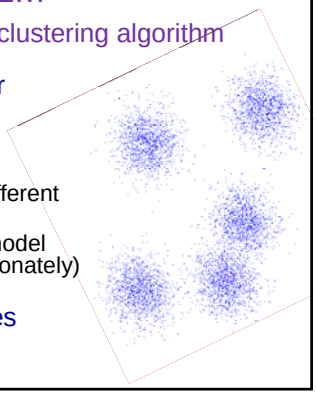


Fri	K-means & Agglomerative Clustering
Mon	**Expectation Maximization (EM)**
Wed	**Principle Component Analysis (PCA)**

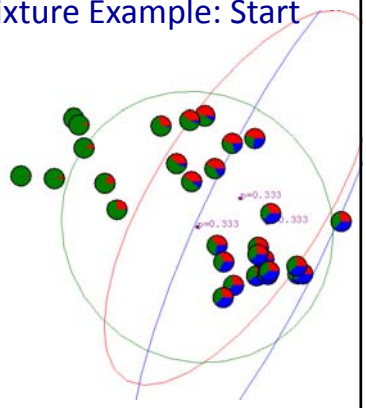
EM

another iterative clustering algorithm

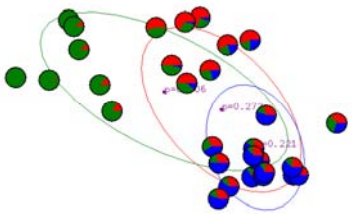
- Pick K random cluster models
- Alternate:
  - Assign data instances ***proportionately*** to different models
  - Revise each cluster model based on its (proportionately) assigned points
- Stop when no changes



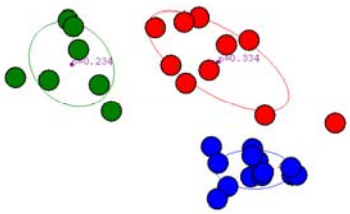
Gaussian Mixture Example: Start



After first iteration



After 20th iteration



Can we do EM with hard assignments?

(univariate case for simplicity)

Iterate: On the *t*'th iteration let our estimates be

$$\theta_t = \{ \mu_1^{(t)}, \mu_2^{(t)} \dots \mu_k^{(t)} \}$$

E-step

Compute "expected" classes of all datapoints

$$p(y = i | x_j, \mu_1 \dots \mu_k) \propto \exp\left(-\frac{1}{2\sigma^2} \|x_j - \mu_i\|^2\right) P(y = i)$$

M-step

Compute most likely new **μ**s given class expectations

$$\mu_i = \frac{\sum_{j=1}^{m} P(y = i | x_j) x_j}{\sum_{j=1}^{m} P(y = i | x_j)}$$

$$\mu_i = \frac{\delta(y = i, x_j)\, x_j}{\sum_{j=1}^{m} \delta(y = i, x_j)}$$

δ represents hard assignment to "most likely" or nearest cluster

Equivalent to k-means clustering algorithm!!!

Summary

- K-means for clustering:
  - algorithm
  - converges because it's coordinate ascent
- Hierarchical agglomerative clustering
- EM for mixture of Gaussians:
  - Viewed as coordinate ascent in parameter space (μ_i, Σ_i, π_i)
  - How to "learn" maximum likelihood parameters (locally max. like.) in the case of unlabeled data
  - Relation to K-means
    - Hard / soft clustering
    - Probabilistic model
- Remember, E.M. can (& ***does***) get stuck in local minima

Dimensionality Reduction

Input data may have thousands or millions of dimensions!

e.g., images have??? And text data has ???,

Dnl S. Wld s Thms J. Cbl / WRF Prfssr f Cmptr
Scnc nd ngnrng t th nvrsty f Wshngtn. ftr gng t schl
t Phllps cdmy, h rcvd bchlr's dgrs n bth Cmptr
Scnc nd Bchmstry t Yl nvrsty n 1982.

Are all those dimensions (letters, pixels) necessary?

Dimensionality Reduction

- Represent data with fewer dimensions! ☺
- Easier learning – fewer parameters
  - |Features| >> |training examples| ??
- Better visualization
  - Hard to understand more than 3D or 4D
- Discover "intrinsic dimensionality" of data
  - High dimensional data that is truly lower dimensional

Feature Selection

- Want to learn f:**X**→Y
  - **X**=<X_1,…,X_n>
  - 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

Greedy **forward** feature selection algorithm

- Pick a dictionary of features
  - e.g., polynomials for linear regression
- Greedy: Start from empty (or simple) set of features $F_0 = \varnothing$
  - Run learning algorithm for current set of features F_t
    Obtain h_t
  - Select **next best feature X_i**
    e.g., X_j giving lowest ***held-out error*** when learning with $F_t \cup \{X_j\}$
  - $F_{t+1} \leftarrow F_t \cup \{X_i\}$
  - Repeat

Greedy **backward** feature selection algorithm

- Pick a dictionary of features
  - e.g., polynomials for linear regression
- Greedy: Start with all features $F_0 = F$
  - Run learning algorithm for current set of features F_t
    Obtain h_t
  - Select **next worst feature X_i**
    X_i giving lowest held out-error learner when learning with F_t - {X_j}
  - $F_{t+1} \leftarrow F_t$ - {X_i}
  - Repeat

Impact of feature selection on classification of fMRI data [Pereira et al. '05]

Accuracy classifying category of word read by subject

#voxels	mean	subjects							
		233B	329B	332B	424B	474B	496B	77B	86B
50	0.735	0.783	0.817	0.55	0.783	0.75	0.8	0.65	0.75
100	0.742	0.767	0.8	0.533	0.817	0.85	0.783	0.6	0.783
200	0.737	0.783	0.783	0.517	0.817	0.883	0.75	0.583	0.783
300	**0.75**	**0.8**	**0.817**	**0.567**	**0.833**	**0.883**	**0.75**	**0.583**	**0.767**
400	0.742	0.8	0.783	0.583	0.85	0.833	0.75	0.583	0.75
800	0.735	0.833	0.817	0.567	0.833	0.833	0.7	0.55	0.75
1600	0.638	0.8	0.817	0.45	0.783	0.833	0.633	0.5	0.75
all (~2500)	0.638	0.767	0.767	0.25	0.75	0.833	0.567	0.433	0.733

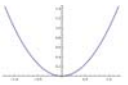
Table 1: **Average accuracy across all pairs of categories, restricting the procedure to use a certain number of voxels for each subject.** The highlighted line corresponds to the best mean accuracy, obtained using 300 voxels.

Voxels scored by p-value of regression to predict voxel value from the task

Feature Selection through Regularization

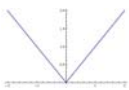
- Previously, we discussed regularization with a squared norm:

$$\hat{\theta} = \arg\min_{\theta} Loss(\theta; \mathcal{D}) + \lambda \sum_i \theta_i^2$$



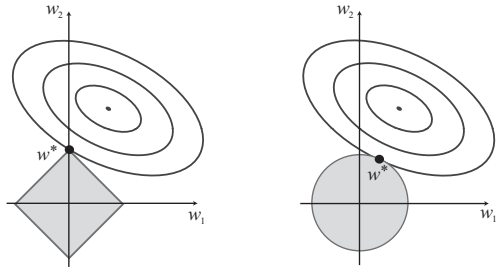
- What if we used an L_1 norm instead?

$$\hat{\theta} = \arg\min_{\theta} Loss(\theta; \mathcal{D}) + \lambda \sum_i |\theta_i|$$



- What about L_∞?
- These norms work, but are harder to optimize! And, it can be tricky to set λ!!!

Regularization



L1 L2

16

Lower Dimensional Projections

- Rather than picking a subset of the features, we can ***make new ones*** by combining existing features $x_1 \ldots x_n$

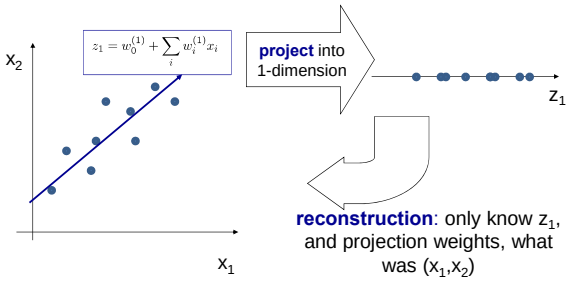
$$z_1 = w_0^{(1)} + \sum_i w_i^{(1)} x_i$$

$$\ldots$$

$$z_k = w_0^{(k)} + \sum_i w_i^{(k)} x_i$$

- New features are linear combinations of old ones
- Reduces dimension when k<n
- Let's see this in the unsupervised setting
  - just **X**, but no Y

Linear projection and reconstruction



reconstruction: only know z_1, and projection weights, 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

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

- Project a point into a (lower dimensional) space:
  - **point**: $\mathbf{x} = (x_1,\ldots,x_n)$
  - **select a basis** – set of unit (length 1) basis vectors $(\mathbf{u}_1,\ldots,\mathbf{u}_k)$
    - we consider orthonormal basis:
      - $\mathbf{u}_i \bullet \mathbf{u}_i = 1$, and $\mathbf{u}_i \bullet \mathbf{u}_j = 0$ for $i \neq j$
  - **select a center** – $\bar{\mathbf{x}}$, defines offset of space
  - **best coordinates** in lower dimensional space defined by dot-products: $(z_1,\ldots,z_k)$, $z_i = (\mathbf{x}-\bar{\mathbf{x}}) \bullet \mathbf{u}_i$

3D Data

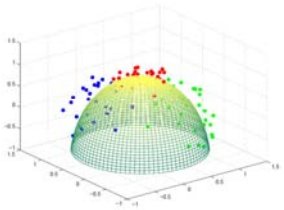


FIGURE 14.15. *Simulated data in three classes, near the surface of a half-sphere.*

Projection to 2D

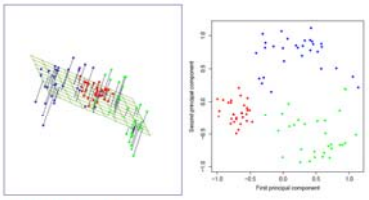


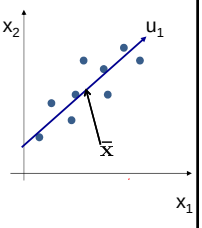
FIGURE 14.21. *The best rank-two linear approximation to the half-sphere data. The right panel shows the projected points with coordinates given by $\mathbf{U}_2\mathbf{D}_2$, the first two principal components of the data.*

PCA finds projection that minimizes reconstruction error

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

$$\hat{\mathbf{x}}^i = \bar{\mathbf{x}} + \sum_{j=1}^{k} z_j^i \mathbf{u}_j \qquad \bar{\mathbf{x}} = \frac{1}{m}\sum_{i=1}^{m} \mathbf{x}^i$$

- PCA: $z_j^i = (\mathbf{x}^i - \bar{\mathbf{x}}) \cdot \mathbf{u}_j$
  - Given k<n, find $(\mathbf{u}_1,\ldots,\mathbf{u}_k)$ minimizing reconstruction error:



$$error_k = \sum_{i=1}^{m} (\mathbf{x}^i - \hat{\mathbf{x}}^i)^2$$

Understanding the reconstruction error

$$\hat{\mathbf{x}}^i = \bar{\mathbf{x}} + \sum_{j=1}^{k} z_j^i \mathbf{u}_j \qquad z_j^i = (\mathbf{x}^i - \bar{\mathbf{x}}) \cdot \mathbf{u}_j$$

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

$$error_k = \sum_{i=1}^{m} (\mathbf{x}^i - \hat{\mathbf{x}}^i)^2$$

- 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:

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

$$= \sum_{i=1}^{m} \left(\sum_{j=k+1}^{n} z_j^i u_j \right)^2 = \sum_{i=1}^{m} \sum_{j=k+1}^{n} z_j^i u_j \cdot u_j z_j^i + \sum_{i=1}^{m} \sum_{j=k+1}^{n} \sum_{l=k+1, l\neq k}^{n} z_j^i u_j \cdot u_l z_l^i$$

$$= \sum_{i=1}^{m} \sum_{j=k+1}^{n} (z_j^i)^2$$

Aha! Error is sum of squared weights that would have been used for dimensions that are cut!!!!

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

Reconstruction error and covariance matrix

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

$$= \sum_{i=1}^{m}\sum_{j=k+1}^{n} u_j^T (x^i - \bar{x})(x^i - \bar{x})^T u_j$$

$$= \sum_{j=k+1}^{n} u_j^T \left[\sum_{i=1}^{m}(x^i - \bar{x})(x^i - \bar{x})^T\right] u_j$$

$$error_k = \sum_{j=k+1}^{n} u_j^T \Sigma u_j$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{x}^i - \bar{\mathbf{x}})(\mathbf{x}^i - \bar{\mathbf{x}})^T$$

Now, to find the u_j, we minimize:

$$u^T \Sigma u + \lambda(1 - u^T u)$$

Lagrange multiplier to ensure orthonormal

Take derivative, set equal to 0, …, solutions are eigenvectors

$$\Sigma u_i = \lambda_i u_i$$

Minimizing reconstruction error and eigen vectors

- Minimizing reconstruction error equivalent to picking orthonormal basis ($\mathbf{u}_1$,…,$\mathbf{u}_n$) minimizing:

$$error_k = m \sum_{j=k+1}^{n} \mathbf{u}_j^T \Sigma \mathbf{u}_j$$

- Solutions: eigen vectors

$$\Sigma u_i = \lambda_i u_i$$

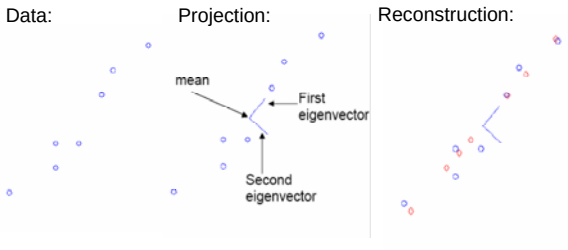
- So, minimizing reconstruction error equivalent to picking ($\mathbf{u}_{k+1}$,…,$\mathbf{u}_n$) to be eigen vectors with smallest eigen values
- And, our projection should be onto the ($\mathbf{u}_1$,…,$\mathbf{u}_k$) with the largest values

Basic PCA algorithm

- Start from m by n data matrix **X**
- **Recenter**: subtract mean from each row of **X**
  - $\mathbf{X_c} \leftarrow \mathbf{X} - \overline{\mathbf{X}}$
- **Compute covariance** matrix:
  - $\Sigma \leftarrow 1/m\ \mathbf{X_c}^T \mathbf{X_c}$
- Find **eigen vectors and values** of Σ
- **Principal components:** k eigen vectors with highest eigen values

PCA example

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



Handwriting

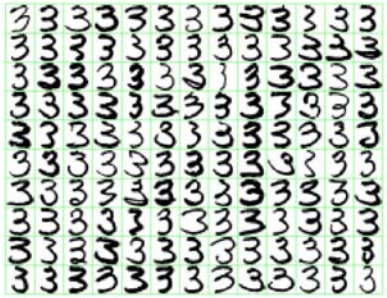
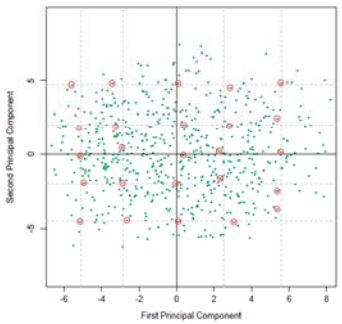
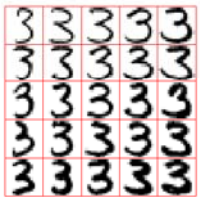


FIGURE 14.22. *A sample of 130 handwritten 3's shows a variety of writing styles.*

29

Two Principle Eigenvectors





30

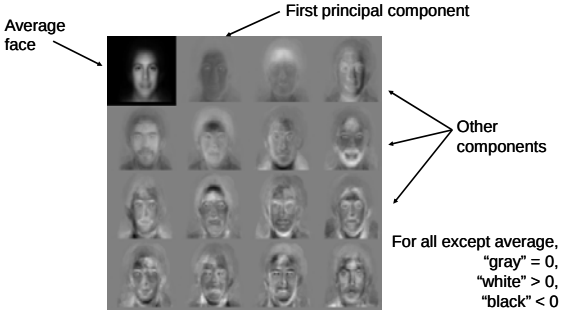
Eigenfaces [Turk, Pentland '91]

Input images:

- N images
- Each 50×50 pixels
- 2500 features

Misleading figure.
Best to think of as an N × 2500 matrix: |Examples| × |Features|

Reduce Dimensionality 2500 → 15



Using PCA for Compression

- Store each face as coefficients of projection onto first few principal components

$$\text{image} = \sum_{i=0}^{i_{\max}} a_i \text{Eigenface}_i$$

Using PCA for Recognition

- Compute projections of target image, compare to database ("nearest neighbor classifier")

Eigenfaces reconstruction

- Each image corresponds to adding together the principal components:

Scaling up

- Covariance matrix can be really big!
  - Σ is n×n
  - 10000 features can be common!
  - Finding eigenvectors is very slow…
- Use singular value decomposition (SVD)
  - Finds top k eigenvectors
  - Great implementations available, e.g., Matlab svd

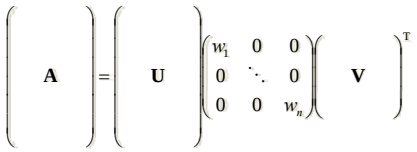
SVD

Write $\mathbf{X} = \mathbf{W}\,\mathbf{S}\,\mathbf{V}^T$

- $\mathbf{X}$ ← data matrix, one row per datapoint
- $\mathbf{W}$ ← weight matrix
  - one row per datapoint – coordinate of $\mathbf{x}^i$ in eigenspace
- $\mathbf{S}$ ← singular value matrix, diagonal matrix
  - in our setting each entry is eigenvalue λ_j
- $\mathbf{V}^T$ ← singular vector matrix
  - in our setting each row is eigenvector $\mathbf{v}_j$

SVD

notation change ☹



- Treat as black box: code widely available
  In Matlab: `[U,W,V]=svd(A,0)`

PCA using SVD algorithm

- Start from m by n data matrix **X**
- **Recenter**: subtract mean from each row of **X**
  - $\mathbf{X_c} \leftarrow \mathbf{X} - \overline{\mathbf{X}}$
- **Call SVD** algorithm on $\mathbf{X_c}$ – ask for k singular vectors
- **Principal components:** k singular vectors with highest singular values (rows of $\mathbf{V}^T$)
  - **Coefficients:** project each point onto the new vectors

Singular Value Decomposition (SVD)

- Handy mathematical technique that has application to many problems
- Given any $m \times n$ matrix **A**, algorithm to find matrices **U**, **V**, and **W** such that

  $\mathbf{A} = \mathbf{U}\mathbf{W}\mathbf{V}^T$

  U is $m \times n$ and orthonormal

  W is $n \times n$ and diagonal

  V is $n \times n$ and orthonormal

SVD

- The w_i are called the singular values of **A**
- If **A** is singular, some of the w_i will be 0
- In general *rank*(**A**) = number of nonzero w_i
- SVD is mostly unique (up to permutation of singular values, or if some w_i are equal)

What you need to know

- Dimensionality reduction
  - why and when it's important
- Simple feature selection
- Regularization as a type of feature selection
- Principal component analysis
  - minimizing reconstruction error
  - relationship to covariance matrix and eigenvectors
  - using SVD