Co-Training for Semi-supervised learning (cont.)

Machine Learning – 10701/15781

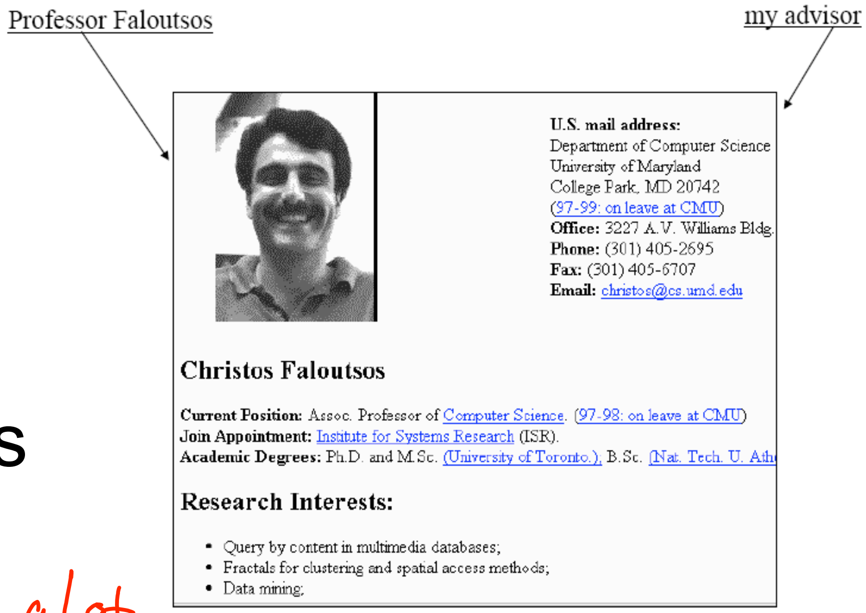
Carlos Guestrin

Carnegie Mellon University

April 23rd, 2007

Exploiting redundant information in semi-supervised learning

- Want to predict Y from features **X**
  - □ $f(\mathbf{X})$ ~~a~~ → Y
  - □ have some labeled data **L**
  - □ lots of unlabeled data **U**
- Co-training assumption: **X** is very expressive
  - □ $\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2)$
  - □ can learn
    - $g_1(\mathbf{X}_1)$ ~~a~~ → Y
    - $g_2(\mathbf{X}_2)$ ~~a~~ → Y



U.S. mail address:
Department of Computer Science
University of Maryland
College Park, MD 20742
(97-99: on leave at CMU)
Office: 3227 A.V. Williams Bldg.
Phone: (301) 405-2695
Fax: (301) 405-6707
Email: christos@cs.umd.edu

Christos Faloutsos

Current Position: Assoc. Professor of Computer Science. (97-98: on leave at CMU)
Join Appointment: Institute for Systems Research (ISR).
Academic Degrees: Ph.D. and M.Sc. (University of Toronto.); B.Sc. (Nat. Tech. U. Ath

Research Interests:

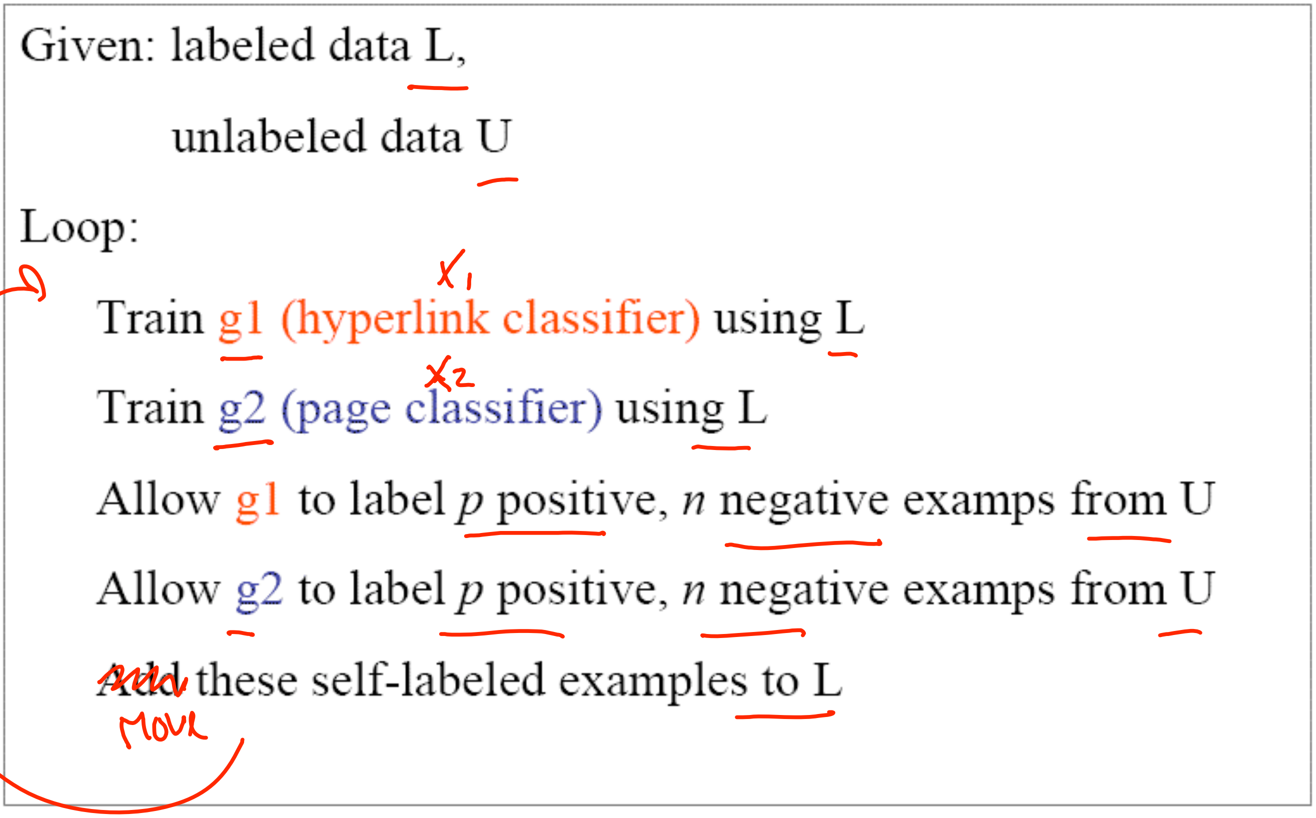
- Query by content in multimedia databases;
- Fractals for clustering and spatial access methods;
- Data mining;

can do alot with unlabeled data, especially if $X_1 \perp X_2 | Y$

Co-Training Algorithm

[Blum & Mitchell '99]

(example of the co-training principle)



```
Given: labeled data L,
        unlabeled data U
Loop:
                                $x_1$
    Train g1 (hyperlink classifier) using L
                              $x_2$
    Train g2 (page classifier) using L
    Allow g1 to label p positive, n negative examps from U
    Allow g2 to label p positive, n negative examps from U
    ~~Add~~ these self-labeled examples to L
    move
```

Understanding Co-Training: A simple setting

- Suppose $\mathbf{X}_1$ and $\mathbf{X}_2$ are discrete
  - □ $|\mathbf{X}_1| = |\mathbf{X}_2| = N$

possible values

if X_1 is described by n binary features, $N = 2^n$

- No label noise
- Without unlabeled data, how hard is it to learn g_1 (or g_2)?

$|H| = 2^N$

hypothesis space

training examples is dependent on

$g_1 \in H$

$\ln|H| = N \cdot \ln 2$

X_1

1 $\{+,-\}$

2 $\{+,-\}$

$\vdots$ $\vdots$

n $\{+,-\}$

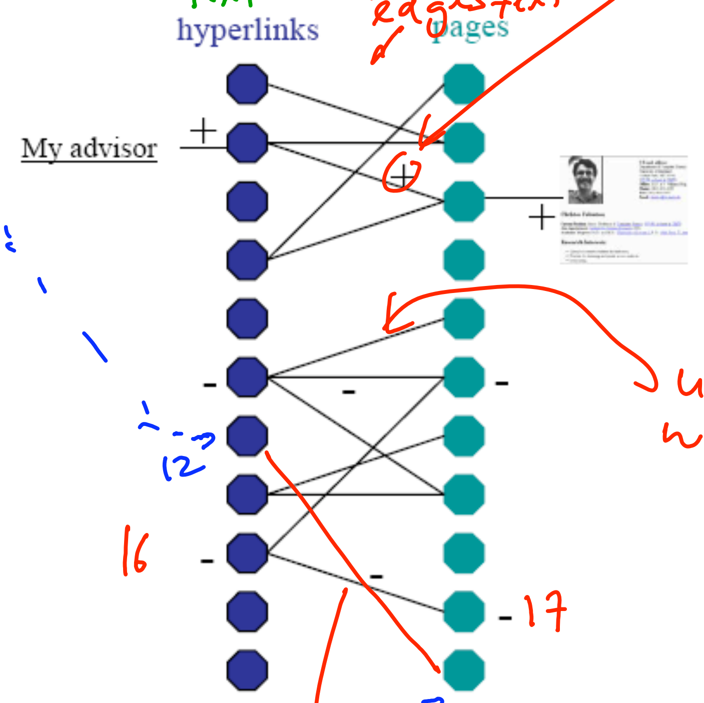
4

©2005-2007 Carlos Guestrin

Co-Training in simple setting – Iteration 0

you get ~~#~~ a web page
with $X_1 = 12$...
& $X_2 = 18$

X_1



set of webpages

form edges text on

labeled data

unlabeled webpage

X_2

N

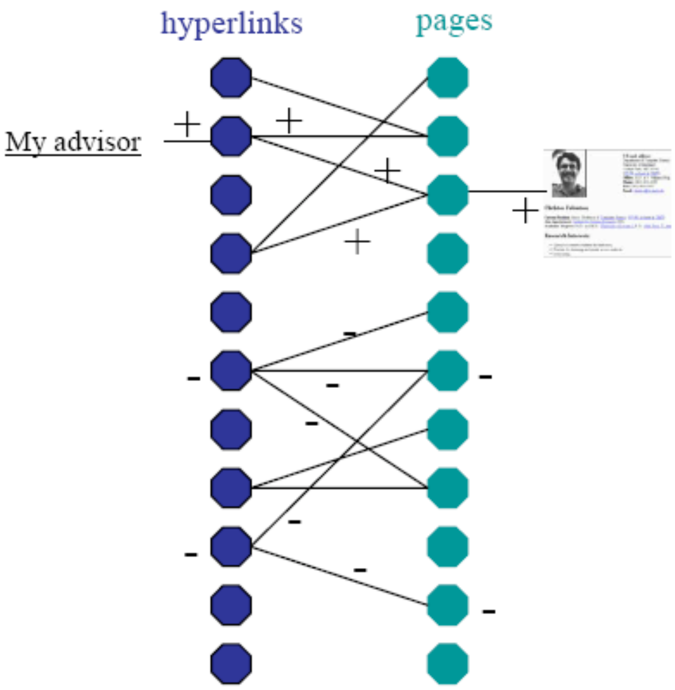
N

one webpage
$X_1 = 16$ & $X_2 = 17$

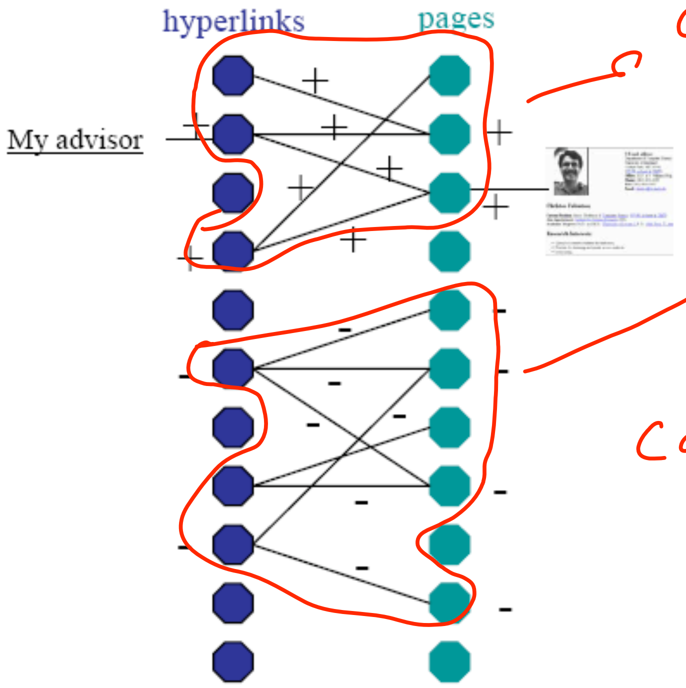
edge $X_1 = x_1$
~~&~~ to $X_2 = x_2$
means
x_1 & x_2
co occurred
on a
webpage

NO LABEL
NOISE

Co-Training in simple setting – Iteration 1



Co-Training in simple setting – after convergence



Connected component +

no edges between them, because no label noise

component -

Co-Training in simple setting – Connected components

- Suppose infinite **unlabeled** data
  - □ Co-training must have at least one labeled example in each connected component of L+U graph

component g_j

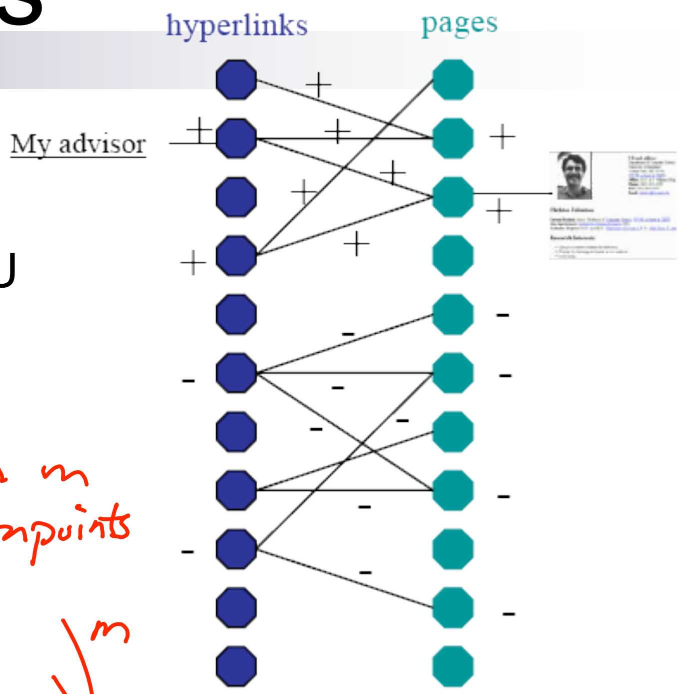
- What's probability of making an error?

with m datapoints

∃ connected component, where no data was labeled

test point x

$$E[\text{error}] = \sum_{g_j \in \text{components}} P(x \in g_j) \underbrace{\left(1 - P(x \in g_j)\right)^m}_{\text{no training data in } g_j}$$



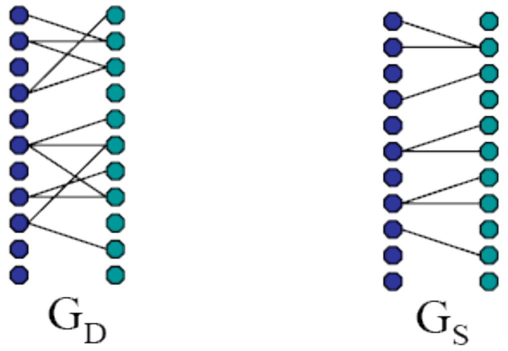
$$E[\text{error}] = \sum_j P(x \in g_j)(1 - P(x \in g_j))^m$$

Where g_j is the jth connected component of graph of L+U, m is number of labeled examples

- For k Connected components, how much labeled data?

How much unlabeled data?

Want to assure that connected components in the underlying distribution, G_D, are connected components in the observed sample, G_S



$O(\log(N)/\alpha)$ examples assure that with high probability, G_S has same connected components as G_D [Karger, 94]

N is size of G_D, α is min cut over all connected components of G_D

Co-Training theory

- Want to predict Y from features **X**
  - f(**X**) a Y
- Co-training assumption: **X** is very expressive
  - $\mathbf{X} = (\mathbf{X}_1,\mathbf{X}_2)$
  - want to learn $g_1(\mathbf{X}_1)$ a Y and $g_2(\mathbf{X}_2)$ a Y
- *Assumption*: $\exists\ g_1, g_2, \forall\ \mathbf{x}\ g_1(\mathbf{x}_1) = f(\mathbf{x}), g_2(\mathbf{x}_2) = f(\mathbf{x})$
- One co-training result [Blum & Mitchell ’99]
  - If
    - $(\mathbf{X}_1 \perp \mathbf{X}_2 \mid Y)$
    - g_1 & g_2 are PAC learnable from noisy data (and thus f)
  - Then
    - f is PAC learnable from weak initial classifier plus unlabeled data

What you need to know about co-training

- Unlabeled data can help supervised learning (a lot) when there are (mostly) independent redundant features
- One theoretical result:
  - If ($\mathbf{X}_1 \perp \mathbf{X}_2$ | Y) and g_1 & g_2 are PAC learnable from noisy data (and thus f)
  - Then f is PAC learnable from weak initial classifier plus unlabeled data
  - Disagreement between g_1 and g_2 provides bound on error of final classifier
- Applied in many real-world settings:
  - Semantic lexicon generation [Riloff, Jones 99] [Collins, Singer 99], [Jones 05]
  - Web page classification [Blum, Mitchell 99]
  - Word sense disambiguation [Yarowsky 95]
  - Speech recognition [de Sa, Ballard 98]
  - Visual classification of cars [Levin, Viola, Freund 03]

Transductive SVMs

Machine Learning – 10701/15781

Carlos Guestrin

Carnegie Mellon University

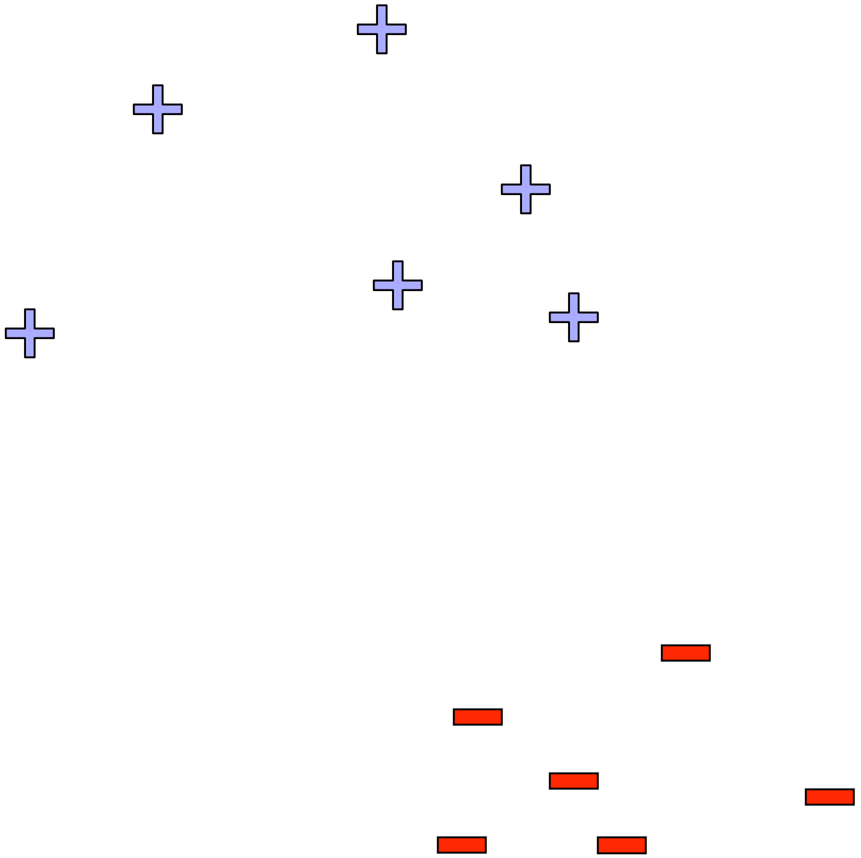
April 23rd, 2007

Semi-supervised learning and discriminative models

- We have seen semi-supervised learning for generative models
  - EM

- What can we do for discriminative models
  - Not regular EM
    - we can't compute P(x)
    - But there are discriminative versions of EM
  - Co-Training!
  - Many other tricks… let's see an example

Linear classifiers – Which line is better?



Data:

$$\left\langle x_1^{(1)}, \ldots, x_1^{(m)}, y_1 \right\rangle$$
$$\vdots$$
$$\left\langle x_n^{(1)}, \ldots, x_n^{(m)}, y_n \right\rangle$$

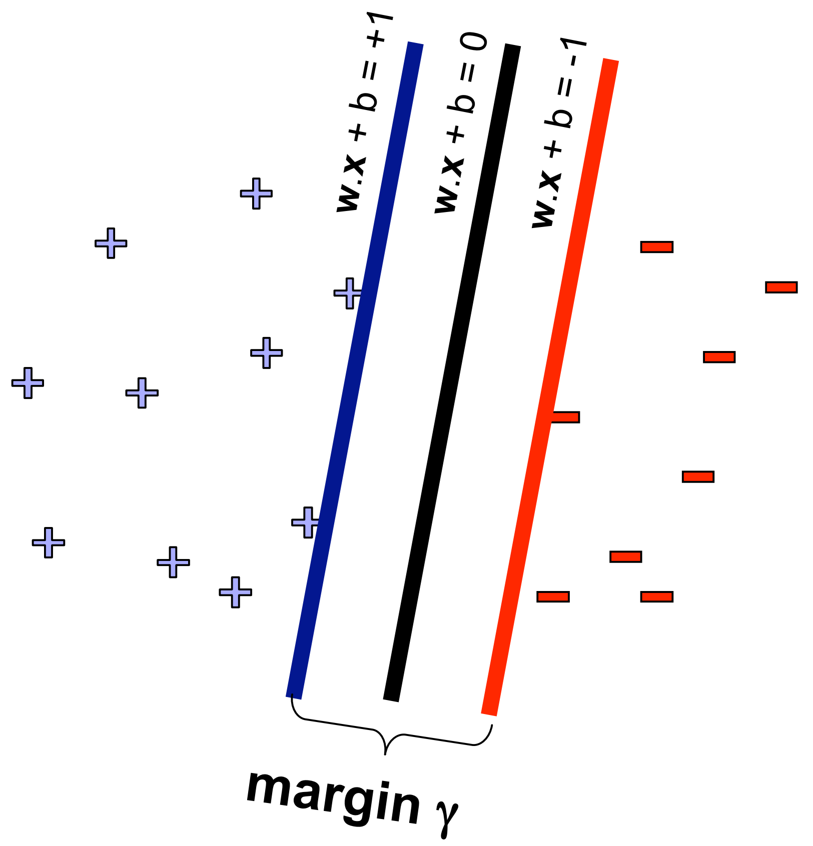
Example i:

$\left\langle x_i^{(1)}, \ldots, x_i^{(m)} \right\rangle$ — m features

$y_i \in \{-1, +1\}$ — class

w.x = $\sum_j w^{(j)} x^{(j)}$

Support vector machines (SVMs)

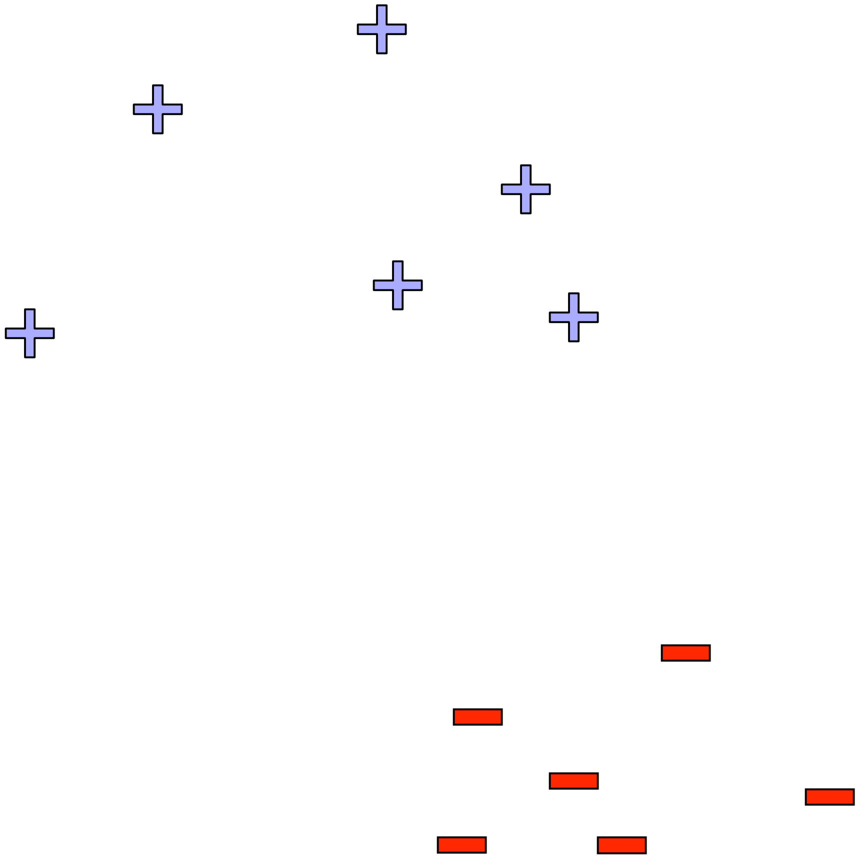


$$\text{minimize}_{\mathbf{w}} \quad \mathbf{w}.\mathbf{w}$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1, \ \forall j$$

- Solve efficiently by quadratic programming (QP)
  - Well-studied solution algorithms

- Hyperplane defined by support vectors

What if we have unlabeled data?



n$_L$ Labeled Data:

$$\left\langle x_1^{(1)}, \ldots, x_1^{(m)}, y_1 \right\rangle$$

$$\vdots$$

$$\left\langle x_n^{(1)}, \ldots, x_n^{(m)}, y_{n_L} \right\rangle$$

Example i:

$\left\langle x_i^{(1)}, \ldots, x_i^{(m)} \right\rangle$ — m features

$y_i \in \{-1, +1\}$ — class

n$_U$ Unlabeled Data:

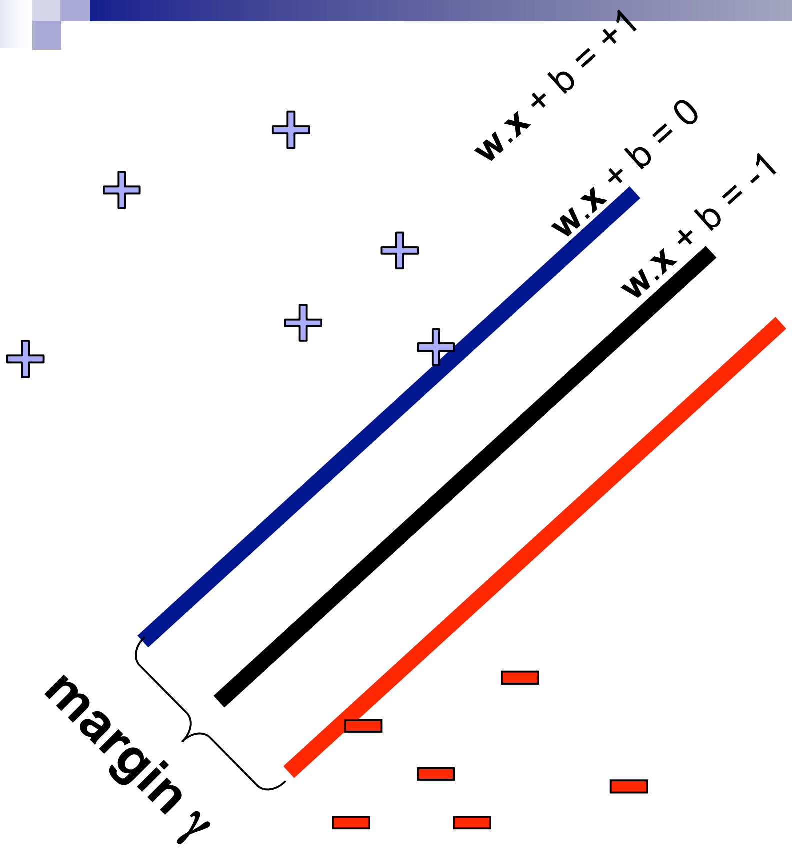
$$\left\langle x_1^{(1)}, \ldots, x_1^{(m)}, ? \right\rangle$$

$$\vdots$$

$$\left\langle x_n^{(1)}, \ldots, x_{n_U}^{(m)}, ? \right\rangle$$

$\mathbf{w}.\mathbf{x} = \sum_j w^{(j)} x^{(j)}$

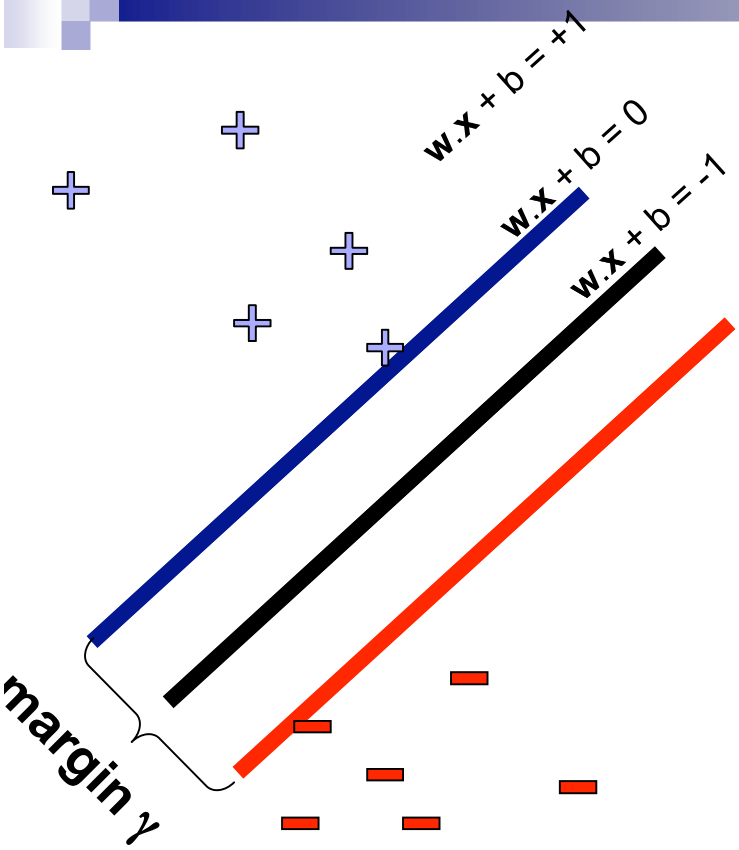
Transductive support vector machines (TSVMs)



$$\text{minimize}_{\mathbf{w}} \quad \mathbf{w}.\mathbf{w}$$

$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1, \ \ \forall j$$

Transductive support vector machines (TSVMs)



$$\text{minimize}_{\mathbf{w},\{\hat{y}_1,\ldots,\hat{y}_{n_U}\}} \quad \mathbf{w}.\mathbf{w}$$

$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1,\ \forall j = 1, \ldots, n_L$$

$$(\mathbf{w}.\mathbf{x}_u + b)\, \hat{y}_u \geq 1,\ \forall u = 1, \ldots, n_U$$

$$\hat{y}_u \in \{-1, +1\},\ \forall u = 1, \ldots, n_U$$

What's the difference between transductive learning and semi-supervised learning?

- Not much, and
- A lot!!!

- Semi-supervised learning:
  - labeled and unlabeled data ! learn **w**
  - use **w** on test data

- Transductive learning
  - same algorithms for labeled and unlabeled data, but…
  - unlabeled data is test data!!!

- You are learning on the test data!!!
  - OK, because you never look at the labels of the test data
  - can get better classification
  - but be very very very very very very very very careful!!!
    - never use test data prediction accuracy to tune parameters, select kernels, etc.

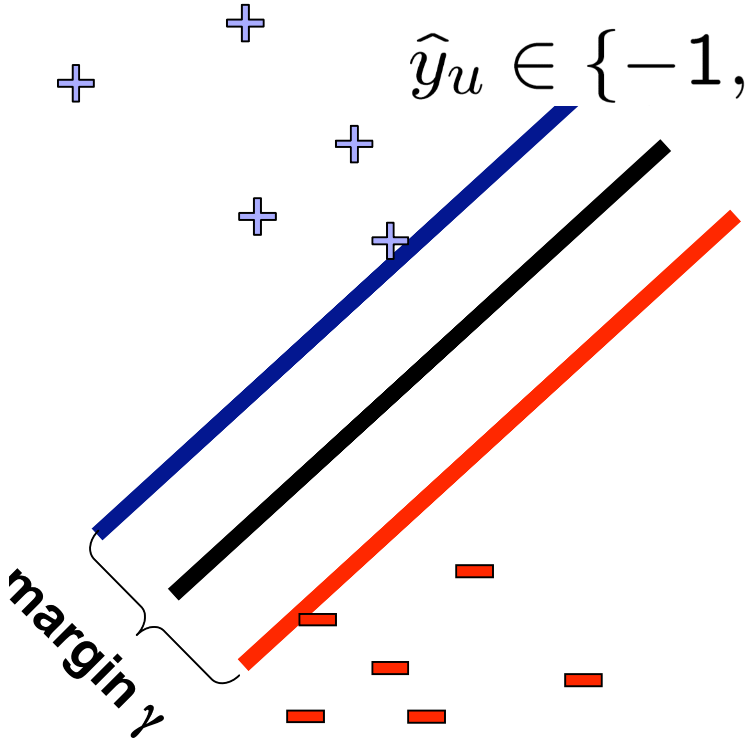
Adding slack variables

$$\text{minimize}_{\mathbf{w},\{\hat{y}_1,\ldots,\hat{y}_{n_U}\}} \quad \mathbf{w}.\mathbf{w}$$

$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 \qquad \forall j = 1, \ldots, n_L$$

$$(\mathbf{w}.\mathbf{x}_u + b)\, \hat{y}_u \geq 1 \qquad \forall u = 1, \ldots, n_U$$

$$\hat{y}_u \in \{-1, +1\}, \;\; \forall u = 1, \ldots, n_U$$



Transductive SVMs – now with slack variables! [Vapnik 98]

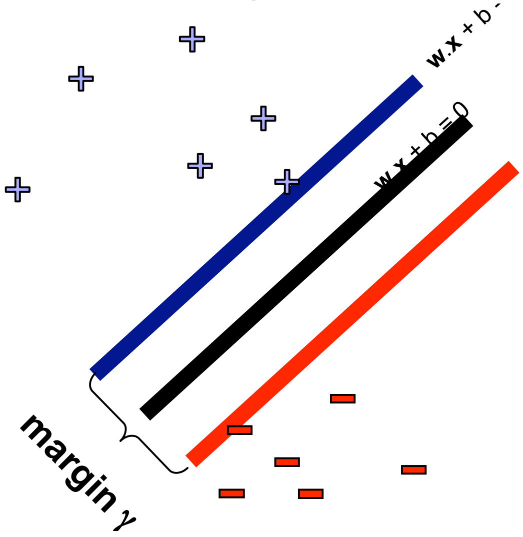
- Optimize $\mathbf{w}, \{\xi_1, \ldots, \xi_{n_L}\}, \{\widehat{y}_1, \ldots, \widehat{y}_{n_U}\}, \{\widehat{\xi}_1, \ldots, \widehat{\xi}_{n_U}\}$

$$\text{minimize} \quad \mathbf{w}.\mathbf{w} + C \sum_j \xi_j + \widehat{C} \sum_u \widehat{\xi}_u$$

$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 - \xi_j, \ \forall j = 1, \ldots, n_L$$

$$(\mathbf{w}.\mathbf{x}_u + b)\, \widehat{y}_u \geq 1 - \widehat{\xi}_u, \ \forall u = 1, \ldots, n_u$$

$$\widehat{y}_u \in \{-1, +1\}, \ \forall u = 1, \ldots, n_u$$



Learning Transductive SVMs is hard!

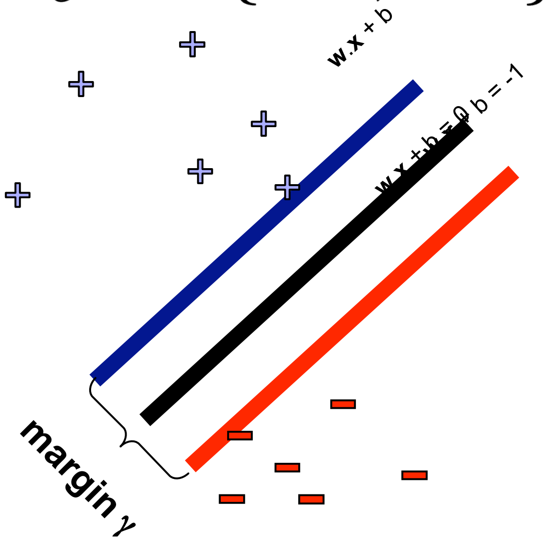
- Optimize $\mathbf{w}, \{\xi_1, \ldots, \xi_{n_L}\}, \{\widehat{y}_1, \ldots, \widehat{y}_{n_U}\}, \{\widehat{\xi}_1, \ldots, \widehat{\xi}_{n_U}\}$

minimize $\quad \mathbf{w}.\mathbf{w} + C \sum_j \xi_j + \widehat{C} \sum_u \widehat{\xi}_u$

$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 - \xi_j,\ \forall j = 1, \ldots, n_L$$

$$(\mathbf{w}.\mathbf{x}_u + b)\, \widehat{y}_u \geq 1 - \widehat{\xi}_u,\ \forall u = 1, \ldots, n_u$$

$$\widehat{y}_u \in \{-1, +1\},\ \forall u = 1, \ldots, n_u$$



- Integer Program
  - □ NP-hard!!!
  - □ Well-studied solution algorithms, but will not scale up to very large problems

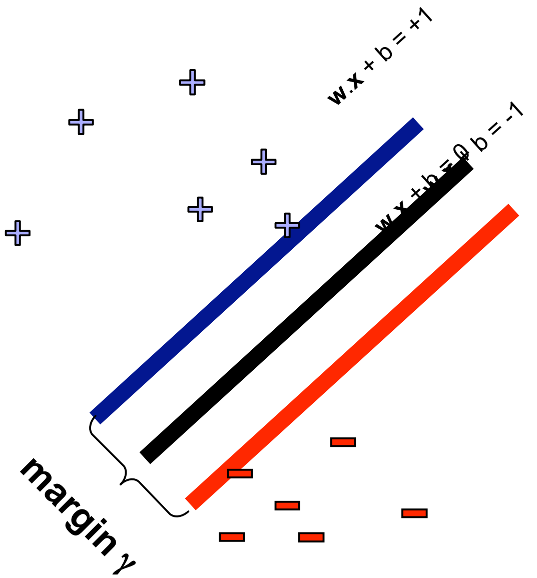
A (heuristic) learning algorithm for Transductive SVMs [Joachims 99]

minimize $\mathbf{w}.\mathbf{w} + C\sum_j \xi_j + \widehat{C}\sum_u \widehat{\xi}_u$

$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 - \xi_j,\ \ \forall j = 1, \ldots, n_L$$

$$(\mathbf{w}.\mathbf{x}_u + b)\, \widehat{y}_u \geq 1 - \widehat{\xi}_u,\ \ \forall u = 1, \ldots, n_u$$

$$\widehat{y}_u \in \{-1, +1\},\ \ \forall u = 1, \ldots, n_u$$



- If you set $\widehat{C}$ to zero → ignore unlabeled data
- Intuition of algorithm:
  - start with small $\widehat{C}$
  - add labels to some unlabeled data based on classifier prediction
  - slowly increase $\widehat{C}$
  - keep on labeling unlabeled data and re-running classifier

Some results classifying news articles – from [Joachims 99]

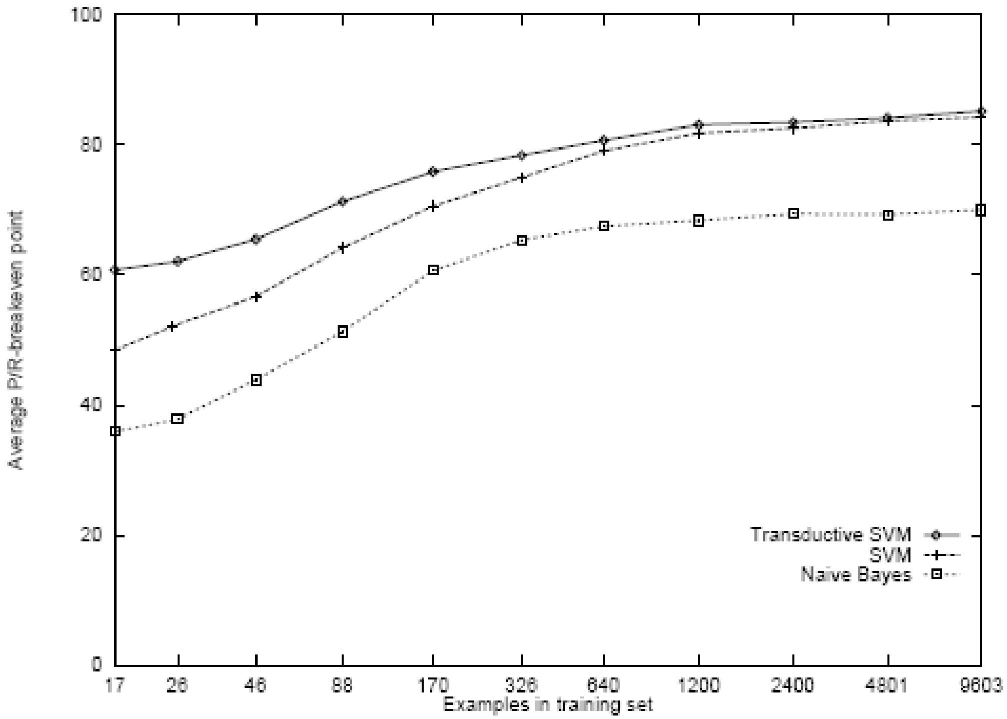


Figure 6: Average P/R-breakeven point on the Reuters dataset for different training set sizes and a test set size of 3,299.

What you need to know about transductive SVMs

- What is transductive v. semi-supervised learning
- Formulation for transductive SVM
  - can also be used for semi-supervised learning
- Optimization is hard!
  - Integer program
- There are simple heuristic solution methods that work well here

Dimensionality reduction

Machine Learning – 10701/15781

Carlos Guestrin

Carnegie Mellon University

April 23rd, 2007

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:$\mathbf{X} \mapsto Y$
  - $\mathbf{X}$=<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

Simple greedy **forward** feature selection algorithm

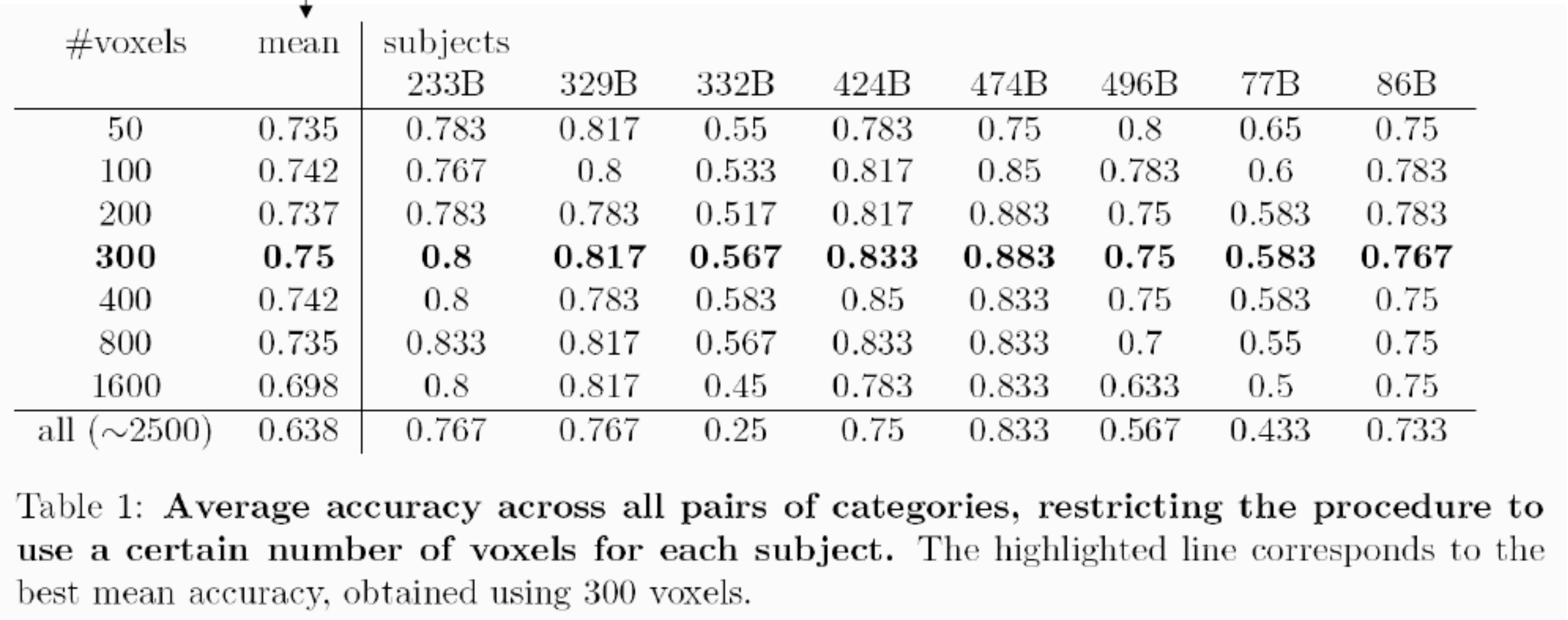
- Pick a dictionary of features
  - e.g., polynomials for linear regression
- Greedy heuristic:
  - Start from empty (or simple) set of features $F_0 = \emptyset$
  - Run learning algorithm for current set of features F_t
    - Obtain h_t
  - Select **next best feature X_i**
    - e.g., X_j that results 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 $\mathbf{X_i}$**
    - 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]

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	0.698	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.

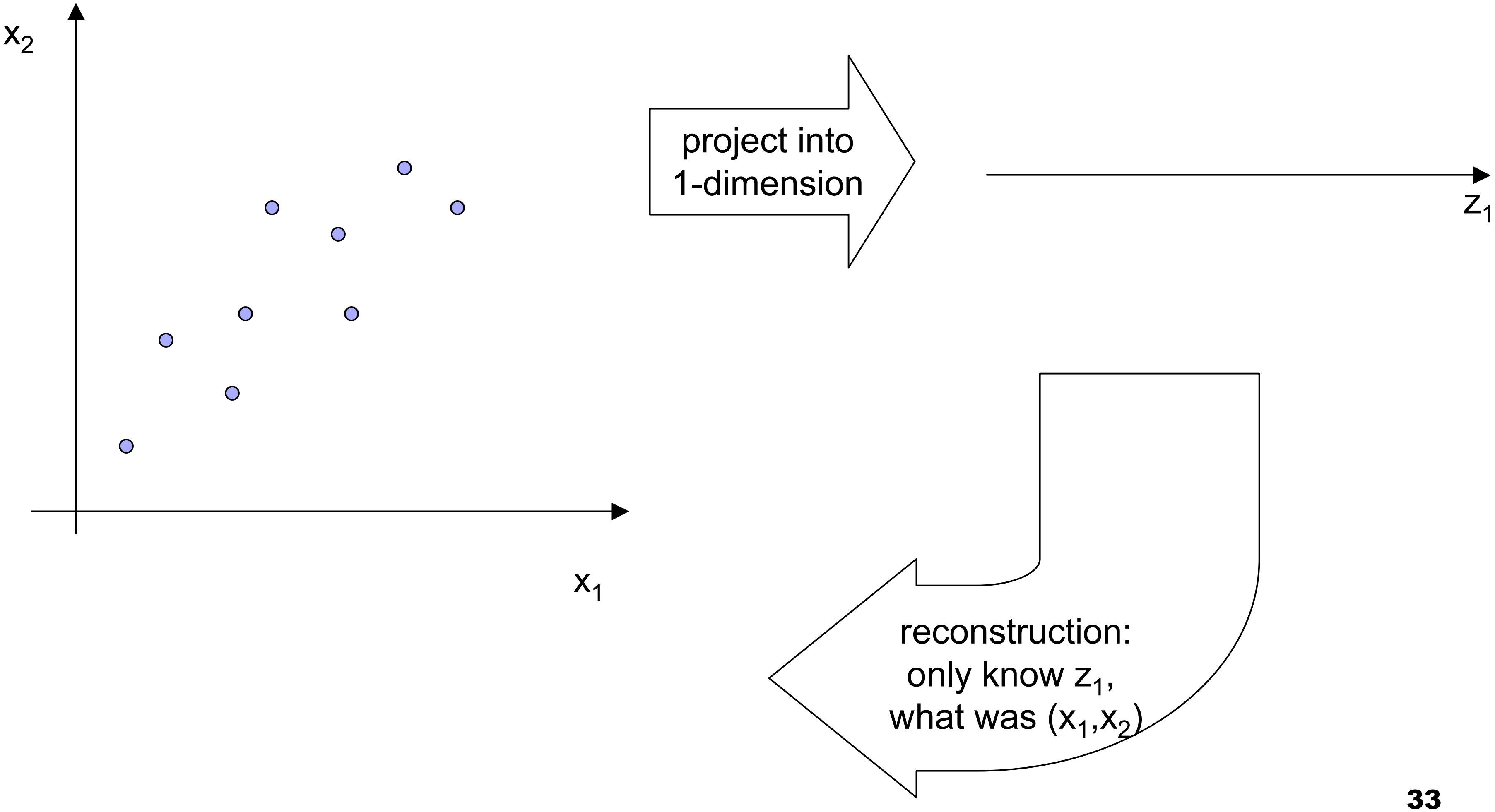
Voxels scored by p-value of regression to predict voxel value from the task

Lower dimensional projections

- Rather than picking a subset of the features, we can new 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

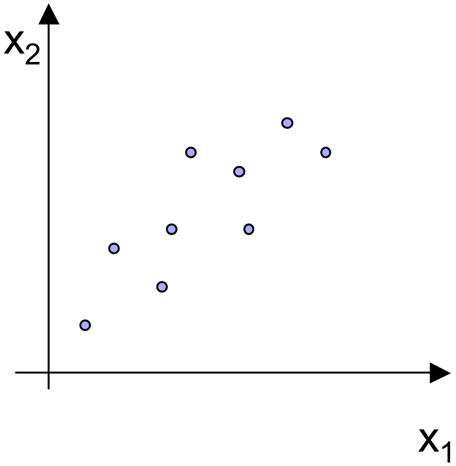
Linear projections, a review

- Project a point into a (lower dimensional) space:
  - **point**: $\mathbf{x} = (x_1,\ldots,x_n)$
  - **select a basis** – set of basis vectors – $(\mathbf{u}_1,\ldots,\mathbf{u}_k)$
    - we consider orthonormal basis:
      - $\mathbf{u}_i \cdot \mathbf{u}_i = 1$, and $\mathbf{u}_i \cdot \mathbf{u}_j = 0$ for $i \neq j$
  - **select a center** – $\bar{\mathbf{x}}$, defines offset of space
  - **best coordinates** in lower dimensional space defined by dot-products: $(z_1,\ldots,z_k)$, $z_i = (\mathbf{x}-\bar{\mathbf{x}})\cdot\mathbf{u}_i$
    - 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…m
- Will represent each point as a projection:
  - □ $\hat{\mathbf{x}}^i = \bar{\mathbf{x}} + \sum_{j=1}^{k} z_j^i \mathbf{u}_j$ where: $\bar{\mathbf{x}} = \frac{1}{m}\sum_{i=1}^{m} \mathbf{x}^i$ and $z_j^i = \mathbf{x}^i \cdot \mathbf{u}_j$
- PCA:
  - □ 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

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

- 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:

Reconstruction error and covariance matrix

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

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

Minimizing reconstruction error and eigen vectors

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

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

- Eigen vector:

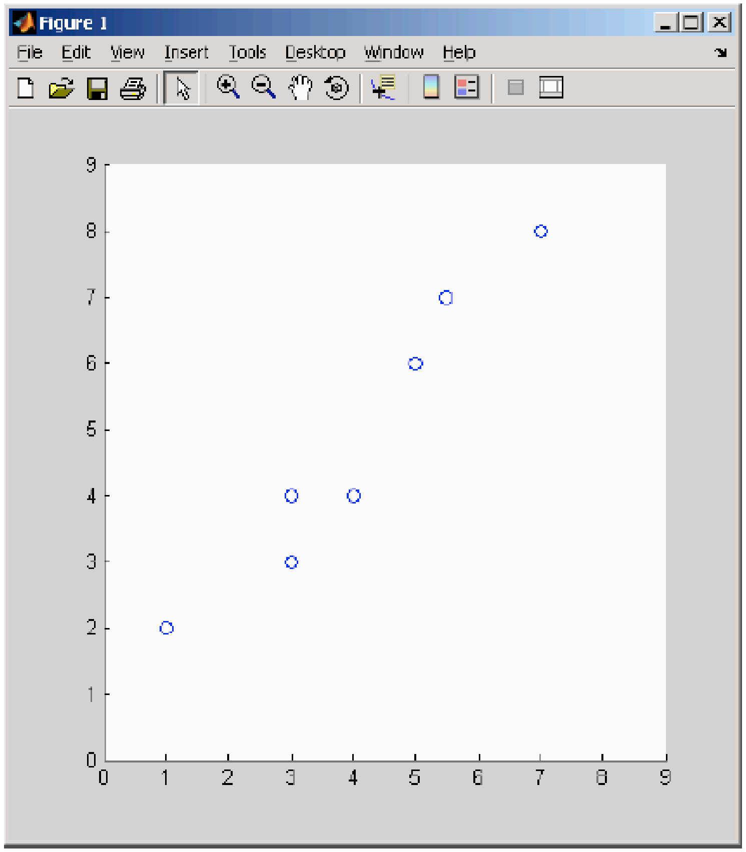
- Minimizing reconstruction error equivalent to picking ($\mathbf{u}_{k+1},\ldots,\mathbf{u}_n$) to be eigen vectors with smallest eigen values

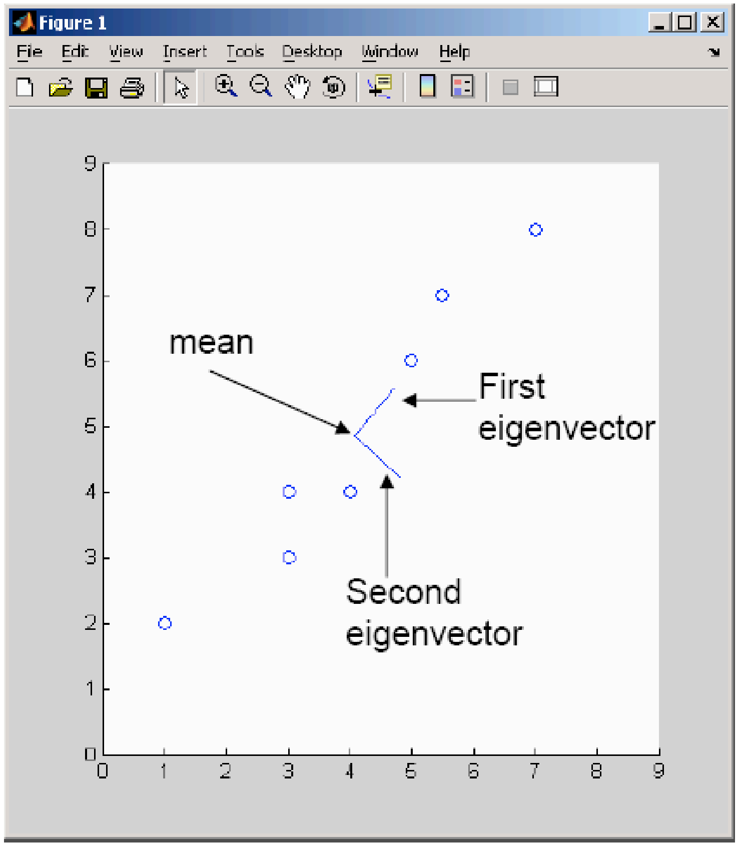
Basic PCA algoritm

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

PCA example

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

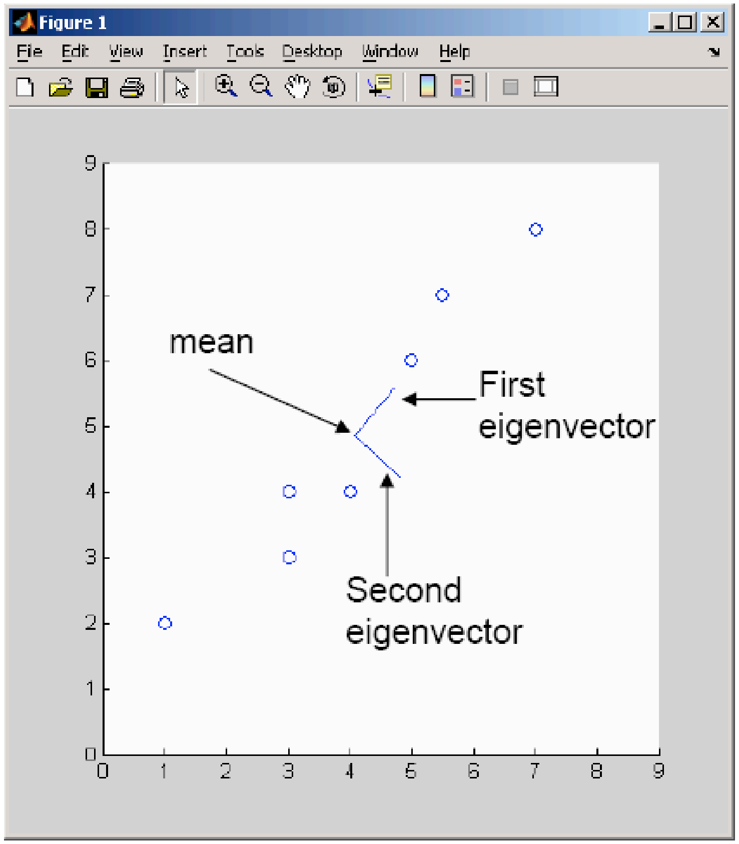


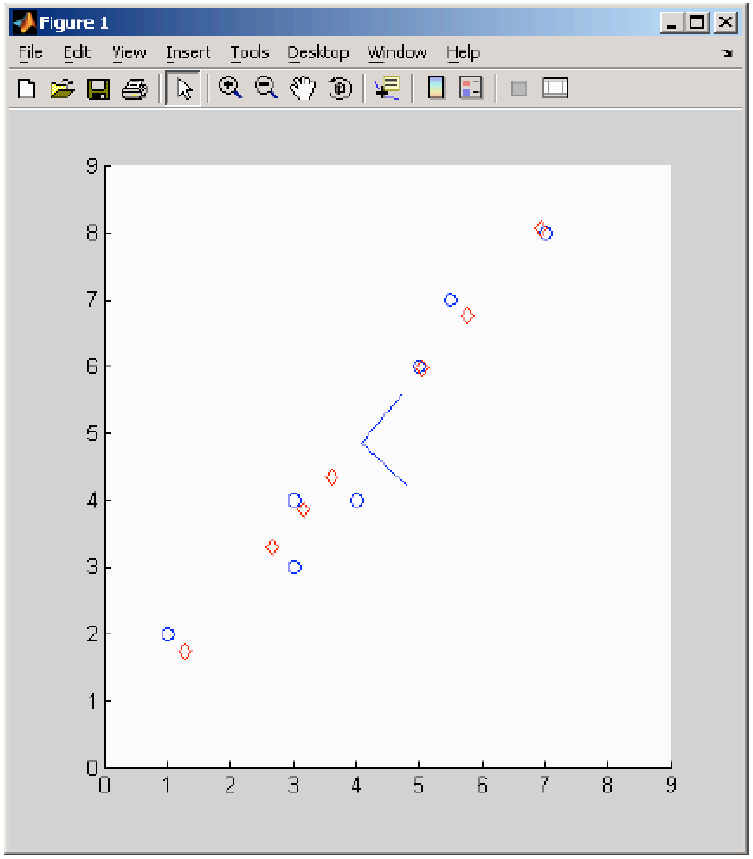


PCA example – reconstruction

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

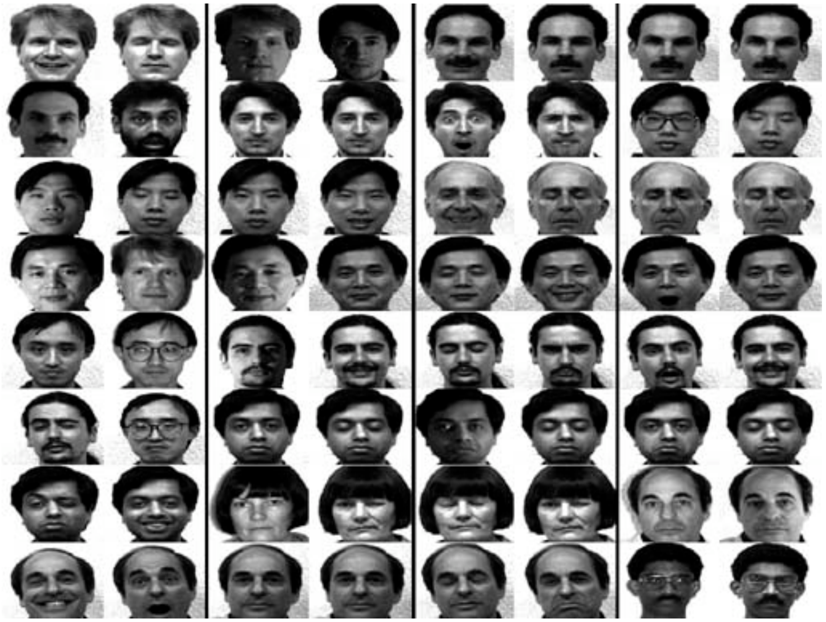
only used first principal component



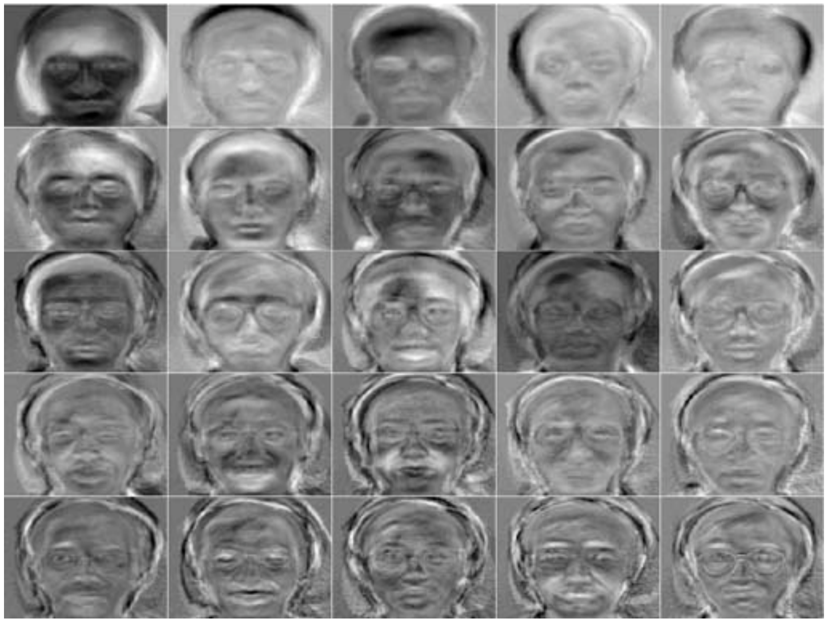


Eigenfaces [Turk, Pentland '91]

- Input images:

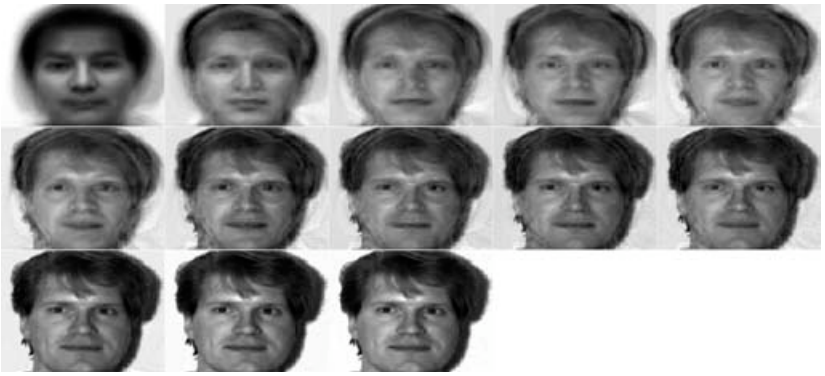


- Principal components:



Eigenfaces reconstruction

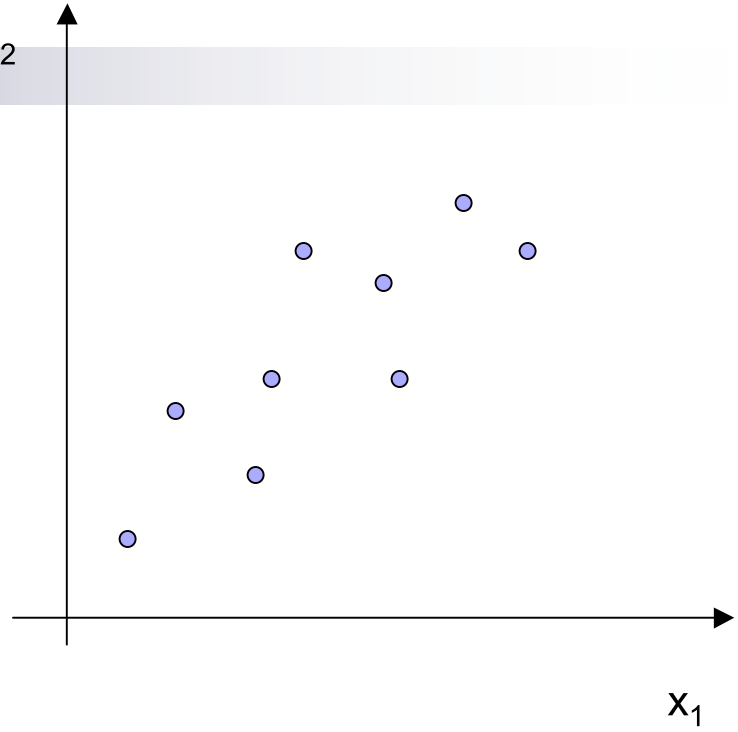
- Each image corresponds to adding 8 principal components:



Relationship to Gaussians

- PCA assumes data is Gaussian
  - □ $\mathbf{x} \sim N(\bar{\mathbf{x}};\Sigma)$
- Equivalent to weighted sum of simple Gaussians:

$$\mathbf{x} = \bar{\mathrm{x}} + \sum_{j=1}^{n} z_j \mathbf{u}_j; \qquad z_j \sim N(0;\sigma_j^2)$$



- Selecting top k principal components equivalent to lower dimensional Gaussian approximation:

$$\mathbf{x} \approx \bar{\mathrm{x}} + \sum_{j=1}^{k} z_j \mathbf{u}_j + \varepsilon; \qquad z_j \sim N(0;\sigma_j^2)$$

  - □ $\varepsilon \sim N(0;\sigma^2)$, where σ^2 is defined by error_k

Scaling up

- Covariance matrix can be really big!
  - □ Σ is n by n
  - □ 10000 features ! |Σ|
  - □ finding eigenvectors is very slow…

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

SVD

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

PCA using SVD algoritm

- 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** become:

Using PCA for dimensionality reduction in classification

- Want to learn f:**X**↦Y
  - □ **X**=<X_1,…,X_n>
  - □ but some features are more important than others

- **Approach**: Use PCA on **X** to select a few important features

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