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
  - \( f(X) \rightarrow Y \)
  - have some labeled data \( L \)
  - lots of unlabeled data \( U \)

- Co-training assumption: X is very expressive
  - \( X = (X_1, X_2) \)
  - can learn
    - \( g_1(X_1) \rightarrow Y \)
    - \( g_2(X_2) \rightarrow Y \)

\[ \text{can do a lot with unlabeled data, especially if } X_1 \perp X_2 | Y \]
Co-Training Algorithm
[Blum & Mitchell ’99]

Given: labeled data \( L \), unlabeled data \( U \)

Loop:
- Train \( g_1 \) (hyperlink classifier) using \( L \)
- Train \( g_2 \) (page classifier) using \( L \)
- Allow \( g_1 \) to label \( p \) positive, \( n \) negative examps from \( U \)
- Allow \( g_2 \) to label \( p \) positive, \( n \) negative examps from \( U \)
- Add these self-labeled examples to \( L \)

(example of the Co-training principle)
Understanding Co-Training: A simple setting

- Suppose $X_1$ and $X_2$ are discrete
  - $|X_1| = |X_2| = N$ (number of possible values)

- No label noise

- Without unlabeled data, how hard is it to learn $g_1$ (or $g_2$)?

\[ |H| = 2^N \]  \# training examples

\[ \{0, -3\} \quad g_i \in H \]

\[ \{1, -3\} \]

\[ \vdots \]

\[ \{n, -3\} \]

\[ |\ln|H| = N \cdot \ln 2 \]
Co-Training in simple setting – Iteration 0

You get a web page with $X_1 = 12$ and $X_2 = 18$. You get the text of hyperlinks for two textons. The set of webpages are labeled data.

One webpage $X_1 = 16$ and $X_2 = 17$.

My advisor means $X_1$ and $X_2$ co-occur on a webpage.

No label noise.

Edge $X_1 = x_1$ to $X_2 = x_2$.
Co-Training in simple setting – Iteration 1
Co-Training in simple setting – after convergence

Connected component

No edge 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

- What’s probability of making an error?
  - For k Connected components, how much labeled data?

\[
E[\text{error}] = \sum_{j} P(x \in g_j) \left( 1 - P(x \in g_j) \right)^m
\]

Where \( g_j \) is the \( j \)th connected component of graph of L+U, \( m \) is number of labeled examples.
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 $G_D$, $\alpha$ 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
  - \( X = (X_1, X_2) \)
  - want to learn \( g_1(X_1) \) a Y and \( g_2(X_2) \) a Y

**Assumption:** \( \exists g_1, g_2, \forall x \ g_1(x_1) = f(x), g_2(x_2) = f(x) \)

- One co-training result [Blum & Mitchell '99]
  - If
    - \( (X_1 \perp X_2 \mid Y) \)
      - \( g_1 \) & \( g_2 \) 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 \((X_1 \perp X_2 \mid Y)\) 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]
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:

\[ \langle x_1^{(1)}, \ldots, x_1^{(m)}, y_1 \rangle \]
\[ \vdots \]
\[ \langle x_n^{(1)}, \ldots, x_n^{(m)}, y_n \rangle \]

Example i:

\[ \langle x_i^{(1)}, \ldots, x_i^{(m)} \rangle \quad \text{— m features} \]
\[ y_i \in \{-1, +1\} \quad \text{— class} \]

\[ w \cdot x = \sum_j w^{(j)} x^{(j)} \]
Support vector machines (SVMs)

\[ w \cdot x + b = +1 \]
\[ w \cdot x + b = 0 \]
\[ w \cdot x + b = -1 \]

- Margin \( \gamma \)
- Solve efficiently by quadratic programming (QP)
  - Well-studied solution algorithms
- Hyperplane defined by support vectors

\[
\minimize_w \quad w \cdot w \\
\left( w \cdot x_j + b \right) y_j \geq 1, \quad \forall j
\]
What if we have unlabeled data?

\[ \mathbf{w} \cdot \mathbf{x} = \sum_j w^{(j)} x^{(j)} \]

**n_L Labeled Data:**
\[ \langle x_1^{(1)}, \ldots, x_1^{(m)}, y_1 \rangle \]
\[ \vdots \]
\[ \langle x_n^{(1)}, \ldots, x_n^{(m)}, y_{n_L} \rangle \]

**Example i:**
\[ \langle x_i^{(1)}, \ldots, x_i^{(m)} \rangle \quad \text{— m features} \]
\[ y_i \in \{-1, +1\} \quad \text{— class} \]

**n_U Unlabeled Data:**
\[ \langle x_1^{(1)}, \ldots, x_1^{(m)}, ? \rangle \]
\[ \vdots \]
\[ \langle x_n^{(1)}, \ldots, x_n^{(m)}, ? \rangle \]
Transductive support vector machines (TSVMs)

\[ \min_w \quad w \cdot w \]

\[ (w \cdot x_j + b) y_j \geq 1, \quad \forall j \]
Transductive support vector machines (TSVMs)

\[ w \cdot x + b = +1 \]
\[ w \cdot x + b = -1 \]
\[ w \cdot x + b = 0 \]

margin \( \gamma \)

\[
\begin{align*}
\text{minimize}_{w, \{\hat{y}_1, \ldots, \hat{y}_{n_U}\}} & \quad w \cdot w \\
\left( w \cdot x_j + b \right) y_j & \geq 1, \quad \forall j = 1, \ldots, n_L \\
\left( w \cdot x_u + b \right) \hat{y}_u & \geq 1, \quad \forall u = 1, \ldots, n_U \\
\hat{y}_u & \in \{-1, +1\}, \quad \forall u = 1, \ldots, n_U
\end{align*}
\]
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 very very very very very careful!!!
  - never use test data prediction accuracy to tune parameters, select kernels, etc.
Adding slack variables

$$\begin{align*}
\text{minimize}_{w, \{\hat{y}_1, \ldots, \hat{y}_{n_U}\}} & \quad w \cdot w \\
(w \cdot x_j + b) y_j & \geq 1 \quad \forall j = 1, \ldots, n_L \\
(w \cdot x_u + b) \hat{y}_u & \geq 1 \quad \forall u = 1, \ldots, n_U \\
\hat{y}_u & \in \{-1, +1\}, \quad \forall u = 1, \ldots, n_U
\end{align*}$$
Transductive SVMs – now with slack variables! [Vapnik 98]

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

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

$(w . x_j + b) y_j \geq 1 - \xi_j, \ \forall j = 1, \ldots, n_L$

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

$\tilde{y}_u \in \{-1, +1\}, \ \forall u = 1, \ldots, n_u$
Learning Transductive SVMs is hard!

Optimize \( w, \{ \xi_1, \ldots, \xi_{n_L} \}, \{ \hat{y}_1, \ldots, \hat{y}_{n_U} \}, \{ \tilde{\xi}_1, \ldots, \tilde{\xi}_{n_U} \} \)

minimize \( w \cdot w + C \sum_j \xi_j + \tilde{C} \sum_u \tilde{\xi}_u \)

\( (w \cdot x_j + b) y_j \geq 1 - \xi_j, \ \forall j = 1, \ldots, n_L \)

\( (w \cdot x_u + b) \hat{y}_u \geq 1 - \tilde{\xi}_u, \ \forall u = 1, \ldots, n_u \)

\( \hat{y}_u \in \{-1, +1\}, \ \forall u = 1, \ldots, n_u \)

- 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 \( w \cdot w + C \sum_j \xi_j + \tilde{C} \sum_u \tilde{\xi}_u \)

\[
\left( w \cdot x_j + b \right) y_j \geq 1 - \xi_j, \quad \forall j = 1, \ldots, n_L
\]

\[
\left( w \cdot x_u + b \right) \tilde{y}_u \geq 1 - \tilde{\xi}_u, \quad \forall u = 1, \ldots, n_u
\]

\( \tilde{y}_u \in \{-1, +1\}, \quad \forall u = 1, \ldots, n_u \)

- If you set \( \tilde{C} \) to zero → ignore unlabeled data
- Intuition of algorithm:
  - start with small \( \tilde{C} \)
  - add labels to some unlabeled data based on classifier prediction
  - slowly increase \( \tilde{C} \)
  - 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 \( f: X \rightarrow Y \)
  - \( X = <X_1, \ldots, X_n> \)
  - 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

©2005-2007 Carlos Guestrin
Simple greedy \textbf{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>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>233B</td>
</tr>
<tr>
<td>50</td>
<td>0.735</td>
<td>0.783</td>
</tr>
<tr>
<td>100</td>
<td>0.742</td>
<td>0.767</td>
</tr>
<tr>
<td>200</td>
<td>0.737</td>
<td>0.783</td>
</tr>
<tr>
<td>300</td>
<td>0.75</td>
<td>0.8</td>
</tr>
<tr>
<td>400</td>
<td>0.742</td>
<td>0.8</td>
</tr>
<tr>
<td>800</td>
<td>0.735</td>
<td>0.833</td>
</tr>
<tr>
<td>1600</td>
<td>0.698</td>
<td>0.8</td>
</tr>
<tr>
<td>all (~2500)</td>
<td>0.638</td>
<td>0.767</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 create 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**: \( x = (x_1, \ldots, x_n) \)
  - **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: \( x^i = (x_1^i, \ldots, x_n^i), i=1\ldots m \)
- Will represent each point as a projection:

\[
\hat{x}^i = \bar{x} + \sum_{j=1}^{k} z_j^i u_j \quad \text{where:} \quad \bar{x} = \frac{1}{m} \sum_{i=1}^{m} x^i \quad \text{and} \quad z_j^i = x^i \cdot u_j
\]

- PCA:
  - Given k·n, find \((u_1, \ldots, u_k)\) minimizing reconstruction error:

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

- Note that $x^i$ can be represented exactly by n-dimensional projection:

$$x^i = \bar{x} + \sum_{j=1}^{n} z_j^i u_j$$

- Rewriting error:

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

Given $k \cdot n$, find $(u_1, ..., u_k)$ minimizing reconstruction error:

$$error_k = \sum_{i=1}^{m} (x^i - \hat{x}^i)^2$$
Reconstruction error and covariance matrix

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

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

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

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

- 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 $X$
  - $X_c \leftarrow X - \overline{X}$
- Compute covariance matrix:
  - $\Sigma \leftarrow X_c^T X_c$
- 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:
Relationship to Gaussians

- PCA assumes data is Gaussian
  \[ \mathbf{x} \sim \mathcal{N}(\mathbf{x}; \Sigma) \]

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

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

- \( \epsilon \sim \mathcal{N}(0; \sigma^2) \), where \( \sigma^2 \) is defined by error_k
Scaling up

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

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

Write $X = U \Sigma V^T$

- $X \leftarrow$ data matrix, one row per datapoint
- $U \leftarrow$ weight matrix, one row per datapoint – coordinate of $x_i$ in eigenspace
- $\Sigma \leftarrow$ 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 an 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 \( V^\top \))
  \[ \text{Coefficients} \] become:
Using PCA for dimensionality reduction in classification

- Want to learn $f: X \rightarrow Y$
  - $X = <X_1, \ldots, X_n>$
  - 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