Feature selection and dimensionality reduction

Machine Learning

Slides adapted from Tom Mitchell

The big picture

So far in the semester, we have seen

- What is learning? How to measure learnability?
  - Learning = generalization to previously unseen examples
  - Different ways of formally defining this idea (PAC, Bayesian)
- Different learning algorithms
  - Derived from different ways of defining learning
  - Empirical loss minimization
- Dealing with missing labels

But where do those #&@# features come from?

Overview

- Feature selection
  - Criteria for identifying good features
- Dimensionality reduction
  - Principal Components Analysis and the like

Why is dimensionality reduction useful?

Several reasons

- Don't know which features are good up front, so add them all and try to figure out which ones are good later

- Sample complexity bounds depend on dimensionality
  - Higher dimensionality $\Rightarrow$ possibly worse generalization

- Visualizing, interpreting and storing high dimensional data is hard

Dimensionality reduction is another tool. Use it, but know what assumptions are baked into the specific methods

Overview

- Feature selection
  - Criteria for identifying good features

- Dimensionality reduction
  - Principal Components Analysis and the like

Supervised feature selection

Suppose we have a large library of N features that we think are interesting, but we want to find a "good" subset

The setting

- Each example is a feature vector $(x_1, x_2, \cdots, x_N)$
- And we have a training set $\{(\mathbf{x}, y)\}$

Approaches

- Score each feature or subsets of features. (How?)
- Search for the best subsets of features automatically during learning (How?)

Strategies for scoring features

- Cross-validation accuracy using only one feature or a subset of features

- Mutual information between feature x_i and the label y

$$I(X_i, Y) = \sum_k \sum_y P(x_i = k, Y = y) \log \frac{P(X_i = k, Y = y)}{P(x_i = k)P(Y = y)}$$

- Domain specific criteria
  - You might know something about your problem that allows you to score features

- Other approaches exist…

Iterative methods for choosing features

- Forward search
  - Start with one or no features
  - Greedily add more features, recalculating scores at each iteration
- Backward search
  - Start with all features
  - Greedily remove the lowest scoring features one-by-one and recalculate scores

Important: Only use the training set for feature selection. Never the test set!

Regularization as feature selection

A L2 norm penalty to the objective forces features to get low weights, thus implicitly selecting features

Ideally, we would like to find weight vector with as many zeros as possible

- Irrelevant features will automatically get discarded
- This is equivalent to adding a L0 norm as a regularizer, L0 minimization is intractable

Instead use an *L1 regularizer*

$$\min_{\mathbf{w}} ||\mathbf{w}||_1 + \text{loss}(\mathbf{w}) \quad \text{where} \quad ||\mathbf{w}||_1 = \sum_i |w_i|$$

Overview

- Feature selection
  - Criteria for identifying good features

- Dimensionality reduction
  - Principal Components Analysis and the like

Mapping inputs to lower dimensions

- Feature selection: Choosing a subset of features
- Instead, we can map features to lower dimensional space
  - New features are effectively functions of the original features
- Does not consider class labels, only the instances

Principal components analysis

The intuition

- Given points in d dimensional space, find a projection to a lower dimensional space such that as much information as possible is preserved
  - Eg: Best 2 dimensional approximation for three dimensional data

- Choose a projection that minimizes the squared error in reconstructing the original data

PCA: Minimizing Reconstruction Error

Suppose the data consists m examples that are d dimensional vectors

$$\mathbf{x} = (x_1, x_2, \cdots, x_d)$$

Goal: For a n < d, find a new set of basis vectors $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_n$ such that the following reconstruction error is minimized

$$\text{Error} = \sum_{i=1}^{m} ||\mathbf{x}_i - \hat{\mathbf{x}}_i||^2$$

where $\hat{\mathbf{x}}_i = \bar{\mathbf{x}} + \sum_{j=1}^{M} z_j \mathbf{u}_j$

Original data point

Reconstructed data point

Mean of original data

Coefficient of projection

Intuition behind optimization

If n = d, then reconstruction is easy. Just set all u's to be unit basis vectors in the original d-dimensional space and all the coefficients to ones

The goal is to minimize the reconstruction error caused by all the "missing" dimensions

Solving the optimization problem on paper gives us an eigenvalue problem

Recall: Eigenvalues and Eigenvectors

A matrix A has an eigenvector u with eigenvalue λ if

$$Au = \lambda u$$

Nearly all standard matrix libraries have routines for eigen-decomposition

Low dimensional representation via PCA

1. Compute Σ = co-variance of the examples

$$\Sigma_{i,j} = \left(\mathbf{x}_i - \bar{\mathbf{x}}\right)\left(\mathbf{x}_j - \bar{\mathbf{x}}\right)^T$$

2. Find its eigenvalues and eigenvectors

3. Principal components $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_n$ = the n eigenvectors with largest eigenvalues

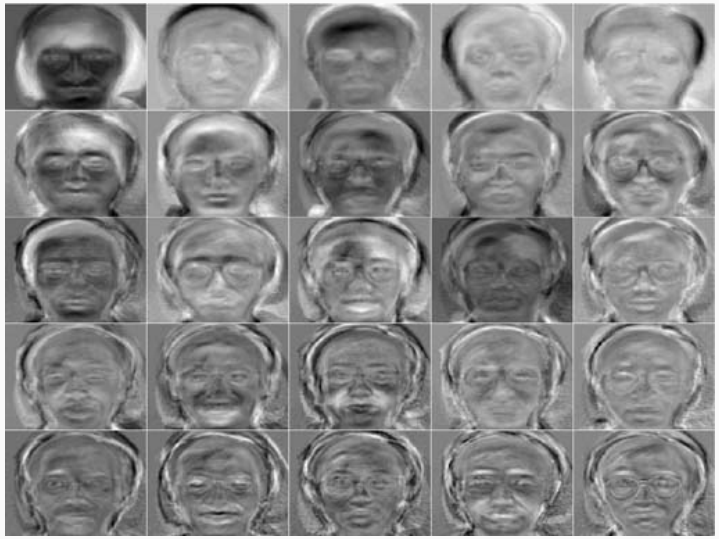
4. Transform an d dimensional example $\mathbf{x}$ to the following n dimensional one: $(\mathbf{x}^T\mathbf{u}_1, \mathbf{x}^T\mathbf{u}_2, \cdots \mathbf{x}^T\mathbf{u}_n)$

For efficiency, can use SVD. Incremental algorithms for SVD are more efficient

Example: Eigenfaces

Faces

Eigenfaces (with 8 principal components)



http://www.cs.princeton.edu/~cdecoro/eigenfaces/

PCA for dimensionality reduction

- Best explains variance in the data
  - Many implementations available
  - Better to use SVD to get eigenvectors if your dimensionality is large
- Only linear projections
  - Kernel PCA transforms input points to higher dimensional spaces before doing projections

PCA for dimensionality reduction

- Best explains variance in the data
  - Many implementations available
  - Better to use SVD to get eigenvectors if your dimensionality is large
- Only linear projections
  - Kernel PCA transforms input points to higher dimensional spaces before doing projections
- Other methods:
  - Neural networks can be seen as doing dimensionality reduction
    - Supervised, non-linear, could be prone to local minimia
    - Very very successful in recent years