CS 2750 Machine Learning
Lecture 19

Dimensionality reduction
Feature selection

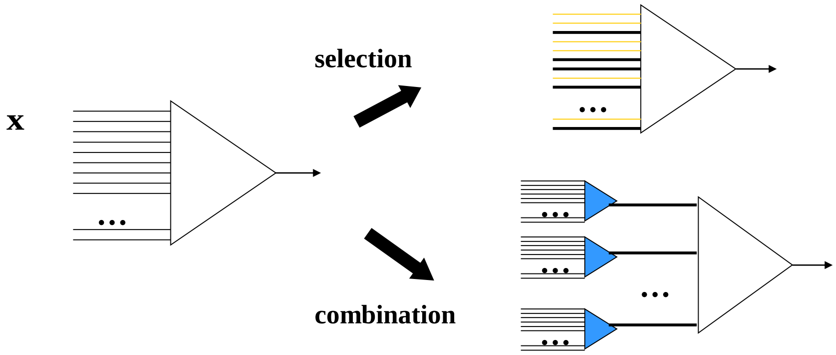
Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Dimensionality reduction. Motivation.

- **ML methods are sensitive to the dimensionality *d* of data**
- **Question:** Is there a lower dimensional representation of the data that captures well its characteristics?
- **Objective of dimensionality reduction:**
  - **Find a lower dimensional representation of data**
- **Two learning problems:**
  - **Supervised** $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), .., (\mathbf{x}_n, , y_n)\}$
    $\mathbf{x}_i = (x_i^1, x_i^2, .., x_i^d)$
  - **Unsupervised** $D = \{\mathbf{x}_1, \mathbf{x}_2, .., \mathbf{x}_n\}$
    $\mathbf{x}_i = (x_i^1, x_i^2, .., x_i^d)$
- **Goal: replace** $\mathbf{x}_i = (x_i^1, x_i^2, .., x_i^d)$
  with $\mathbf{x}_i{}'$ **of dimensionality d'< d**

Dimensionality reduction

- **Solutions**:
  - **Selection of a smaller subset** of inputs (features) from a large set of inputs; train classifier on the reduced input set
  - **Combination of high dimensional inputs** to a smaller set of features $\phi_k(\mathbf{x})$; train classifier on new features



Task-dependent feature selection

Assume: Classification problem:
- $\mathbf{x}$ – input vector, y - output

Objective: Find a subset of inputs/features that gives/preserves most of the output prediction capabilities

Selection approaches:
- **Filtering approaches**
  - Filter out features with small predictive potential
  - Done before classification; typically uses univariate analysis
- **Wrapper approaches**
  - Select features that directly optimize the accuracy of the multivariate classifier
- **Embedded methods**
  - Feature selection and learning closely tied in the method
  - Regularization methods, decision tree methods

Feature selection through filtering

Assume:

Classification problem:

x – input vector, y - output

- **How to select the features/inputs?**

For each input x_i

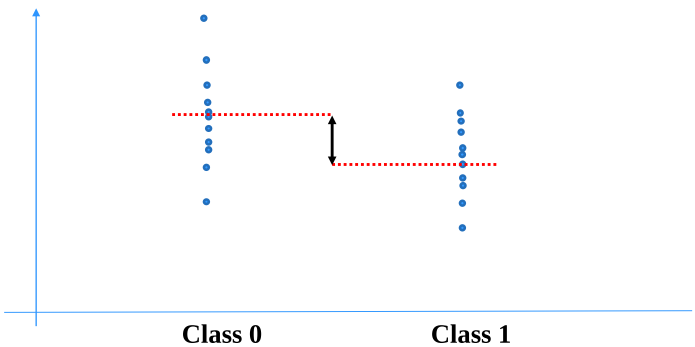
- Calculate **a score** reflecting how well x_i predicts the output y alone

– Pick the inputs with the best scores

(or equivalently eliminate/filter the inputs with the worst scores)

Feature scoring for classification

- **Scores for measuring the differential expression**
  - **T-Test score** (Baldi & Long)
    - Based on the test that two groups come from the same population
    - Null hypothesis: **is mean of class 0 = mean of class 1**

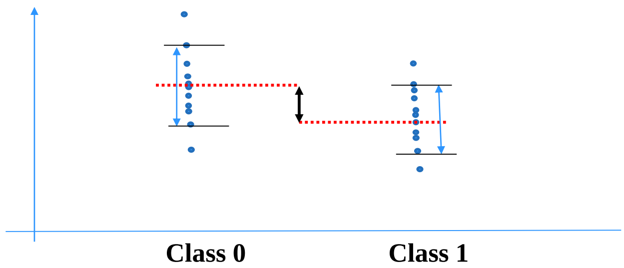


Feature scoring for classification

Scores for measuring the differential expression

- **Fisher Score**

$$Fisher(i) = \frac{(\mu_i^{(+)} - \mu_i^{(-)})^2}{\sigma_i^{(+)2} + \sigma_i^{(-)2}}$$



- **AUROC score:** Area under Receiver Operating Characteristic curve

Feature scoring

- **Correlation coefficients**
  - **Measures linear dependences**

$$\rho(x_k, y) = \frac{Cov(x_k, y)}{\sqrt{Var(x_k)Var(y)}}$$

- **Mutual information**
  - **Measures dependences**
  - **Needs discretized input values**

$$I(x_k, y) = \sum_i \sum_j \tilde{P}(x_k = j, y = i) \log_2 \frac{\tilde{P}(x_k = j, y = i)}{\tilde{P}(x_k = j)\tilde{P}(y = i)}$$

Feature/input dependences

Univariate score assumptions:

- Only one input and its effect on *y* is incorporated in the score
- Effects of two features on *y* are considered to be independent

Correlation based feature selection

- A partial solution to the above problem
- **Idea:** good feature subsets contain features that are highly correlated with the class but independent of each other
- **Assume a set of features S of size *d*.** Then

$$M(S) = \frac{d\bar{r}_{yx}}{\sqrt{d + d(d+1)\bar{r}_{xx}}}$$

- Average correlation between x and class y $\bar{r}_{yx}$
- Average correlation between pairs of xs $\bar{r}_{xx}$

Feature selection: low sample size

Problems:

- **Many inputs and low sample size**
  - if many random features, and not many instances we can learn from, the features with a good differentially expressed score may arise simply by chance
  - The probability of this happening can be quite large
- Techniques to address the problem:
  - reduce **FDR** (False discovery rate) and
  - **FWER** (Family wise error).

Feature selection: wrappers

Wrapper approach:

- The input/feature selection is driven by the prediction accuracy of the classifier (regressor) we actually want to built

How to find the appropriate feature subset S?

- For *d* inputs/features there are 2^d different feature subsets
- **Idea: Greedy search in the space of classifiers**
  - Gradually add features improving the quality of the model
  - Gradually remove features that effect the accuracy the least
  - Score should reflect the accuracy of the classifier (error) and also prevent overfitting
- **Standard way to measure the quality of the model:**
  - Internal cross-validation (k-fold cross validation)

Internal cross-validation

- **Split train set: to internal train and test sets**
- **Internal train set: train different models** (defined e.g. on different subsets of features)
- **Internal test set/s**: **estimate the generalization error** and select the best model among possible models
- **Internal cross-validation (*k*-fold):**
  - Divide the train data into *m* equal partitions (of size *N/k*)
  - Hold out one partition for validation, train the classifiers on the rest of data
  - Repeat such that every partition is held out once
  - The estimate of the generalization error of the learner is the mean of errors of on all partitions

Feature selection: wrappers

- **Example: Greedy (forward) search:**
  - Assume a **logistic regression model**

Start with a simple model: $p(y=1|\mathbf{x},\mathbf{w}) = g(w_o)$

Choose feature x_i with the best error (in the internal step)

$$p(y=1|\mathbf{x},\mathbf{w}) = g(w_o + w_i x_i)$$

Choose feature x_j with the best error (in the internal step)

$$p(y=1|\mathbf{x},\mathbf{w}) = g(w_o + w_i x_i + w_j x_j)$$

Etc.

When to stop ?

Goal: Stop adding features when the internal error on the data stops improving

Embedded methods

Feature selection + model learning done jointly

- **Examples of embedded methods:**
  - **Regularized models**
    - Models of higher complexity are explicitly penalized leading to 'virtual' removal of inputs from the model
    - **Covers:**
      - Regularized logistic/linear regression
      - Support vector machines
        - » Optimization of margins penalizes nonzero weights

$$\underbrace{J_n(\mathbf{w},D)}_{\text{Function to optimize}} = \underbrace{L(\mathbf{w},D)}_{\text{Loss function (fit of the data)}} + \underbrace{R(\mathbf{w})}_{\text{Regularization penalty}}$$

  - **CART/Decision trees**

Unsupervised dimensionality reduction

- **Is there a lower dimensional representation of the data that captures well its characteristics?**
- **Assume:**
  - We have data $D = \{\mathbf{x_1}, \mathbf{x_2}, .., \mathbf{x_N}\}$ such that
    $\mathbf{x}_i = (x_i^1, x_i^2, .., x_i^d)$
  - Assume the dimension d of the data point $\boldsymbol{x}$ is very large
  - We want to analyze $\boldsymbol{x}$, there is no class label y
- **Our goal:**
  - **Find a lower dimensional representation of data of dimension $d' < d$**

Principal component analysis (PCA)

Objective: We want to replace a high-dimensional input vector with a lower dimension vector (obtained by combining inputs)

- Different from the feature subset selection !!!

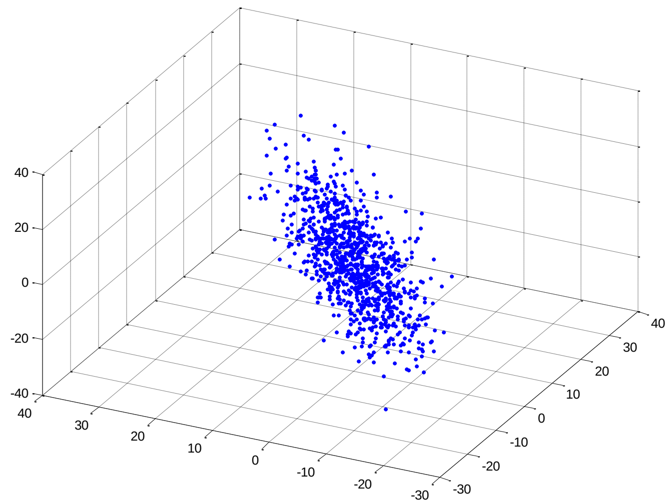
PCA:

- A linear transformation of d dimensional input x to M dimensional feature vector z such that $M < d$

$$\mathbf{z} = \mathbf{A}\mathbf{x}$$

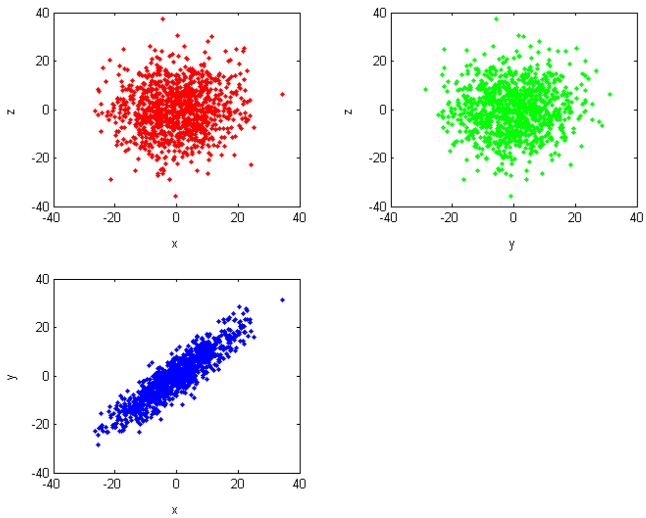
- Many different transformations exists, which one to pick?
- PCA –selects the linear transformation for which **the retained variance is maximal**
- Or, equivalently it is the linear transformation for which the sum of squares reconstruction cost is minimized

PCA: example



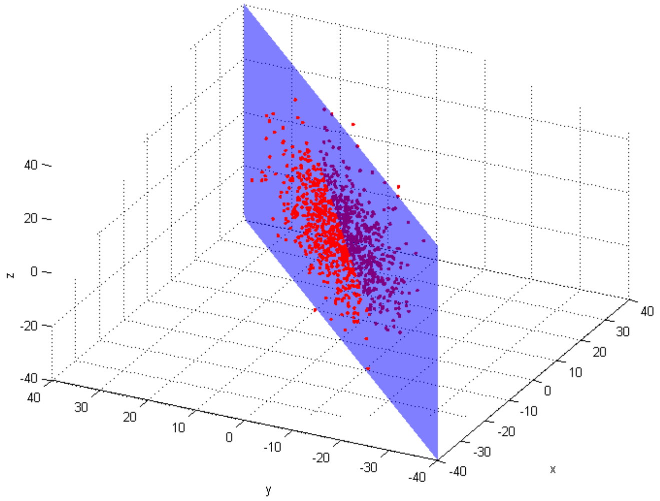
PCA

Projections to different axis



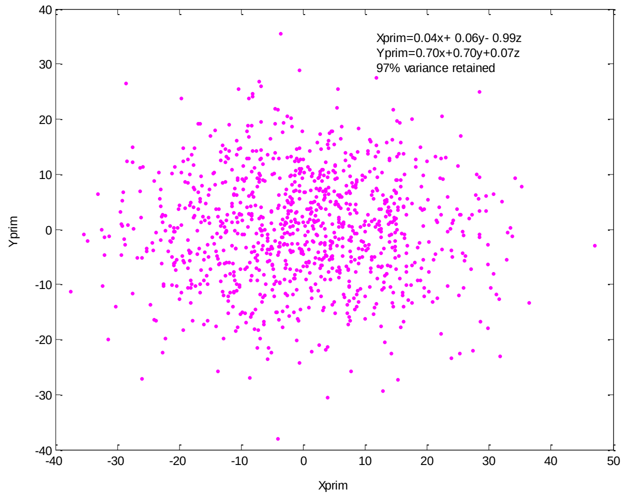
PCA

- PCA projection to the 2 dimensional space



PCA

- PCA projection to the 2 dimensional space



Principal component analysis (PCA)

- **PCA:**
  - linear transformation of a d dimensional input **x** to M dimensional vector **z** such that $M < d$ under which the retained variance is maximal. **Remember:** no y is needed
- **Fact:**
  - A vector **x** can be represented using a set of orthonormal vectors **u** (basis vectors)

$$\mathbf{x} = \sum_{i=1}^{d} z_i \mathbf{u}_i$$

  - Leads to transformation of coordinates (from **x** to **z** using **u**'s)

$$z_i = \mathbf{u}_i^T \mathbf{x} \qquad \mathbf{z} = \mathbf{U}\mathbf{x} \qquad \mathbf{U} = \begin{bmatrix} \mathbf{u}_1^T \\ \mathbf{u}_2^T \\ .. \\ \mathbf{u}_d^T \end{bmatrix}$$

Principal component analysis (PCA)

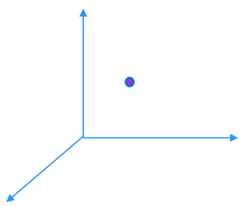
- **Fact:** A vector **x** can be represented using a set of orthonormal vectors **u** (basis vectors)

$$\mathbf{x} = \sum_{i=1}^{d} z_i \mathbf{u}_i$$

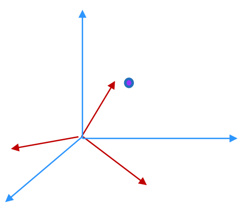
  - Leads to transformation of coordinates

    (from **x** to **z** using **u**'s)

$$z_i = \mathbf{u}_i^T \mathbf{x} \qquad \mathbf{z} = \mathbf{U}\mathbf{x} \qquad \mathbf{U} = \begin{bmatrix} \mathbf{u}_1^T \\ \mathbf{u}_2^T \\ .. \\ \mathbf{u}_d^T \end{bmatrix}$$



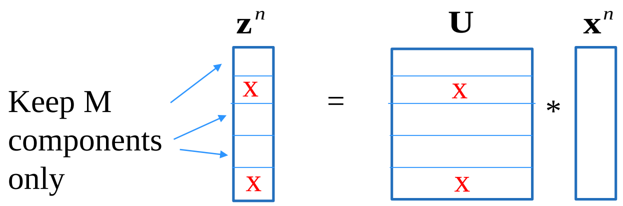
Standard bases:
(1,0,0); (0,1,0); (0,0,1)



New bases: $\mathbf{u}_1$, $\mathbf{u}_2$, $\mathbf{u}_3$

PCA

- **Idea:** represent d-dimensional $\mathbf{x}^n$ with an M-dimensional $\mathbf{z}^n$ formed by subset of z_i coordinates for the bases defined by $\mathbf{U}$.



- **Goal:** We want to find:

  (1) Basis vectors U and (2) a subset *of basis of size M* to keep
- **This effectively replaces $\mathbf{x}^n$ with its approximation $\tilde{\mathbf{x}}^n$**

$$\mathbf{x}^n = \sum_{i=1}^{d} z_i^n \mathbf{u}_i \quad \Rightarrow \quad \tilde{\mathbf{x}}^n = \sum_{i=1}^{M} z_i^n \mathbf{u}_i + \sum_{i=M+1}^{d} b_i \mathbf{u}_i$$

b_i - constant and fixed for all data-points

PCA

- **Goal:** We want to find: z_i

  Basis vectors U and a subset *of basis of size M* to keep

$$\mathbf{x}^n = \sum_{i=1}^{d} z_i^n \mathbf{u}_i \quad \Rightarrow \quad \tilde{\mathbf{x}}^n = \sum_{i=1}^{M} z_i^n \mathbf{u}_i + \sum_{i=M+1}^{d} b_i \mathbf{u}_i$$

b_i - constant and fixed for all data-points

- **How to choose the best set of basis vectors?**
  - We want the subset that gives the best approximation of data *x* in the dataset on average (we use least squares fit)

Error for data entry $\mathbf{x}^n$ $\quad \mathbf{x}^n - \tilde{\mathbf{x}}^n = \sum_{i=M+1}^{d} (z_i^n - b_i)\mathbf{u}_i$

Reconstruction error

$$E_M = \frac{1}{2}\sum_{n=1}^{N} \left\| \mathbf{x}^n - \tilde{\mathbf{x}}^n \right\| = \frac{1}{2}\sum_{n=1}^{N} \sum_{i=M+1}^{d} (z_i^n - b_i)^2$$

PCA

- **Differentiate the error function** with regard to all b_i and set equal to 0 we get:

$$b_i = \frac{1}{N}\sum_{n=1}^{N} z_i^n = \mathbf{u}_i^T \overline{\mathbf{x}} \qquad \overline{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}^n$$

- Then we can rewrite:

$$E_M = \frac{1}{2}\sum_{i=M+1}^{d} \mathbf{u}_i^T \boldsymbol{\Sigma} \mathbf{u}_i \qquad \boldsymbol{\Sigma} = \sum_{n=1}^{N} (\mathbf{x}^n - \overline{\mathbf{x}})(\mathbf{x}^n - \overline{\mathbf{x}})^T$$

- The error function is optimized when basis vectors satisfy:

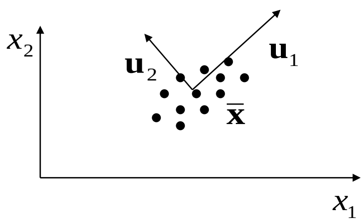
$$\boldsymbol{\Sigma}\mathbf{u}_i = \lambda_i \mathbf{u}_i \qquad E_M = \frac{1}{2}\sum_{i=M+1}^{d} \lambda_i$$

The best *M* basis vectors: discard vectors with *d*-*M* smallest eigenvalues (or keep vectors with M largest eigenvalues)

Eigenvector $\mathbf{u}_i$ – is called a **principal component**

PCA

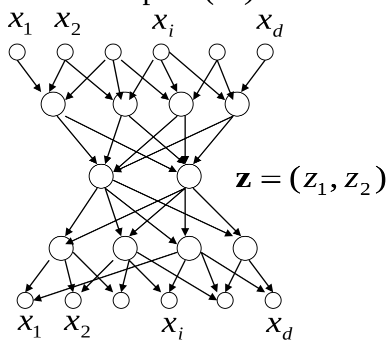
- Once eigenvectors $\mathbf{u}_i$ with largest eigenvalues are identified, they are used to transform the original *d*-dimensional data to *M* dimensions



- To find the "true" dimensionality of the data *d'* we can just look at eigenvalues that contribute the most (small eigenvalues are disregarded)
- **Problem:** PCA is a linear method. The "true" dimensionality can be overestimated. There can be non-linear correlations.
- **Modifications for nonlinearities:** kernel PCA

Dimensionality reduction with neural nets

- **PCA** is limited to linear dimensionality reduction
- To do non-linear reductions we can use neural nets
- **Auto-associative (or auto-encoder) network:** a neural network with the same inputs and outputs (x)



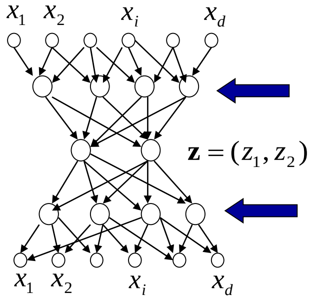
- The middle layer corresponds to the reduced dimensions

Dimensionality reduction with neural nets

- **Error criterion:**

$$E = \frac{1}{2}\sum_{n=1}^{N}\sum_{i=1}^{d}\left(y_i(x^n) - x^n\right)^2$$

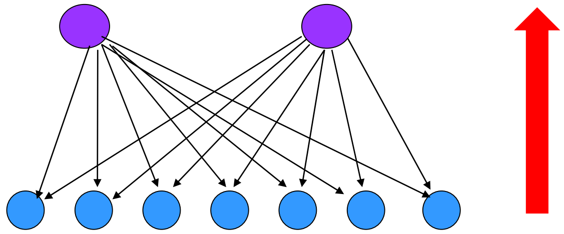
- Error measure tries to recover the original data through limited number of dimensions in the middle layer
- **Non-linearities** modeled through intermediate layers between the middle layer and input/output
- If no intermediate layers are used the model replicates PCA optimization through learning



Latent variable models

- **Learning using unsupervised learning**
- **Dimensionality reduction via inference**

Latent variables (s): Dimensionality k

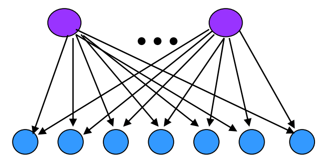


Dimensionality reduction via inference

Observed variables x: real valued vars
Dimensionality d

Cooperative vector quantizer

s: k binary vars



x: d real valued vars

Model:

Latent var s_i:

~ Bernoulli distribution
parameter: π_i

$$P(s_i \mid \pi_i) = \pi_i^{s_i}(1-\pi_i)^{1-s_i}$$

Observable variables x:

~ Normal distribution
parameters: **W, Σ**

$$P(\mathbf{x} \mid \mathbf{s}) = N(\mathbf{Ws}, \Sigma)$$

We assume $\Sigma = \sigma I$

$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} & .. & w_{1k} \\ w_{21} & & & \\ & .. & & \\ w_{d1} & .. & .. & w_{dk} \end{pmatrix}$$

Joint for one instance of x and s:

$$P(\mathbf{x},\mathbf{s} \mid \Theta) = (2\pi)^{-d/2}\sigma^{-d/2}\exp\left\{-\frac{1}{2\sigma^2}(\mathbf{x}-\mathbf{Ws})^T(\mathbf{x}-\mathbf{Ws})\right\}\prod_{i=1}^{k}\pi_i^{s_i}(1-\pi_i)^{(1-s_i)}$$

Dimensionality reduction through clustering

- **Clustering algorithms**
  - group together "similar" instances in the data sample
- **Dimensionality reduction based on clustering**:
  - Replace a high dimensional data entry with a cluster label
- **Problem:**
  - Determistic clustering gives only one label per input
  - May not be enough to represent the data for prediction
- **Solutions:**
  - Clustering over subsets of input data
  - Soft clustering (probability of a cluster is used directly)

Dimensionality reduction through clustering

- **Soft clustering** (e.g. mixture of Gaussians) attempts to cover all instances in the data sample with a small number of groups
  - Each group is more or less responsible for a data entry (responsibility – a posterior of a group given the data entry)

Mixture of G. responsibility

$$h_{il} = \frac{\pi_i p(x_l \mid y_l = i)}{\sum_{u=1}^{k} \pi_u p(x_l \mid y_l = u)}$$

- **Dimensionality reduction based on soft clustering**
  - Replace a high dimensional data with the set of group posteriors
  - Feed all posteriors to the learner e.g. linear regressor, classifier

Dimensionality reduction through clustering

- We can use the idea of soft clustering before applying regression/classification learning
- **Two stage algorithms**
  - Learn the clustering
  - Learn the classification
- Input clustering: $\mathbf{x}$ (high dimensional)
- Output clustering (Input classifier): $p(c = i \mid \mathbf{x})$
- Output classifier: y
- **Example: Networks with Radial Basis Functions (RBFs)**
- **Problem:**
  - Clustering learned based on $p(\mathbf{x})$ (disregards the target)
  - Prediction based on $p(y \mid x)$

Multidimensional scaling

- Find a lower dimensional space projection such that the distances among data points are preserved
- Used in visualization – d-diminensional data transformed to 3D or 2D
- **Dissimilarities before projection** $\delta_{i,j} = \left\| x_i - x_j \right\|$
- **Objective:** Optimize points and their coordinates by fitting the dissimilarities afterwards

$$\min_{\{x_1, x_2, \cdots x_n\}} \sum_{i<j} (\left\| x_i' - x_j' \right\| - \delta_{ij})^2$$