# Foundations of Machine Learning

## Module 3: Instance Based Learning and Feature Selection

### Part A: Instance Based Learning



Sudeshna Sarkar

IIT Kharagpur

# Instance-Based Learning

- One way of solving tasks of approximating discrete or real valued target functions
- Have training examples: $(x_n, f(x_n))$, n=1..N.
- Key idea:
  - just store the training examples
  - when a test example is given then find the closest matches

# Inductive Assumption

- Similar inputs map to similar outputs
  - If not true => learning is impossible
  - If true => learning reduces to defining "*similar*"

- Not all similarities created equal
  - predicting a person's weight may depend on different attributes than predicting their IQ

# Basic k-nearest neighbor classification

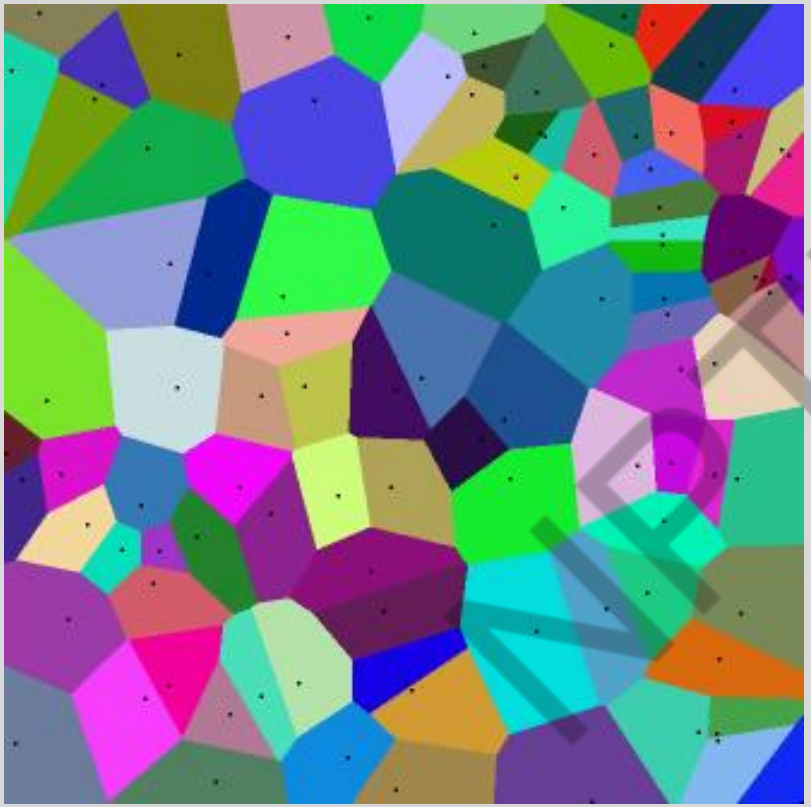
- Training method:
  - Save the training examples
- At prediction time:
  - Find the *k* training examples $(x_1,y_1),\ldots(x_k,y_k)$ that are closest to the test example *x*
  - Predict the most frequent class among those $y_i$'s.

- Example:

http://cgm.cs.mcgill.ca/~soss/cs644/projects/simard/

# What is the decision boundary?

Voronoi diagram



# Basic k-nearest neighbor classification

- Training method:
  - Save the training examples
- At prediction time:
  - Find the *k* training examples $(x_1,y_1),\ldots(x_k,y_k)$ that are closest to the test example *x*
  - Predict the most frequent class among those $y_i$'s.

- Improvements:
  - *Weighting* examples from the neighborhood
  - Measuring "*closeness*"
  - Finding "close" examples in a large training set *quickly*

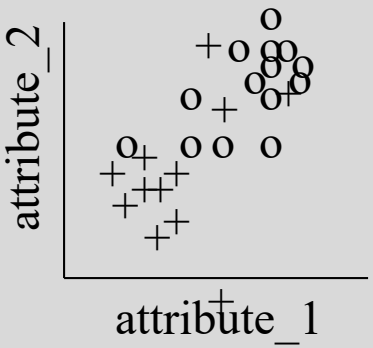
# k-Nearest Neighbor

$$Dist(c_1, c_2) = \sqrt{\sum_{i=1}^{N} \left(attr_i(c_1) - attr_i(c_2)\right)^2}$$

$$k - NearestNeighbors = \left\{k - MIN(Dist(c_i, c_{test}))\right\}$$

$$prediction_{test} = \frac{1}{k}\sum_{i=1}^{k} class_i \ (or \ \frac{1}{k}\sum_{i=1}^{k} value_i)$$

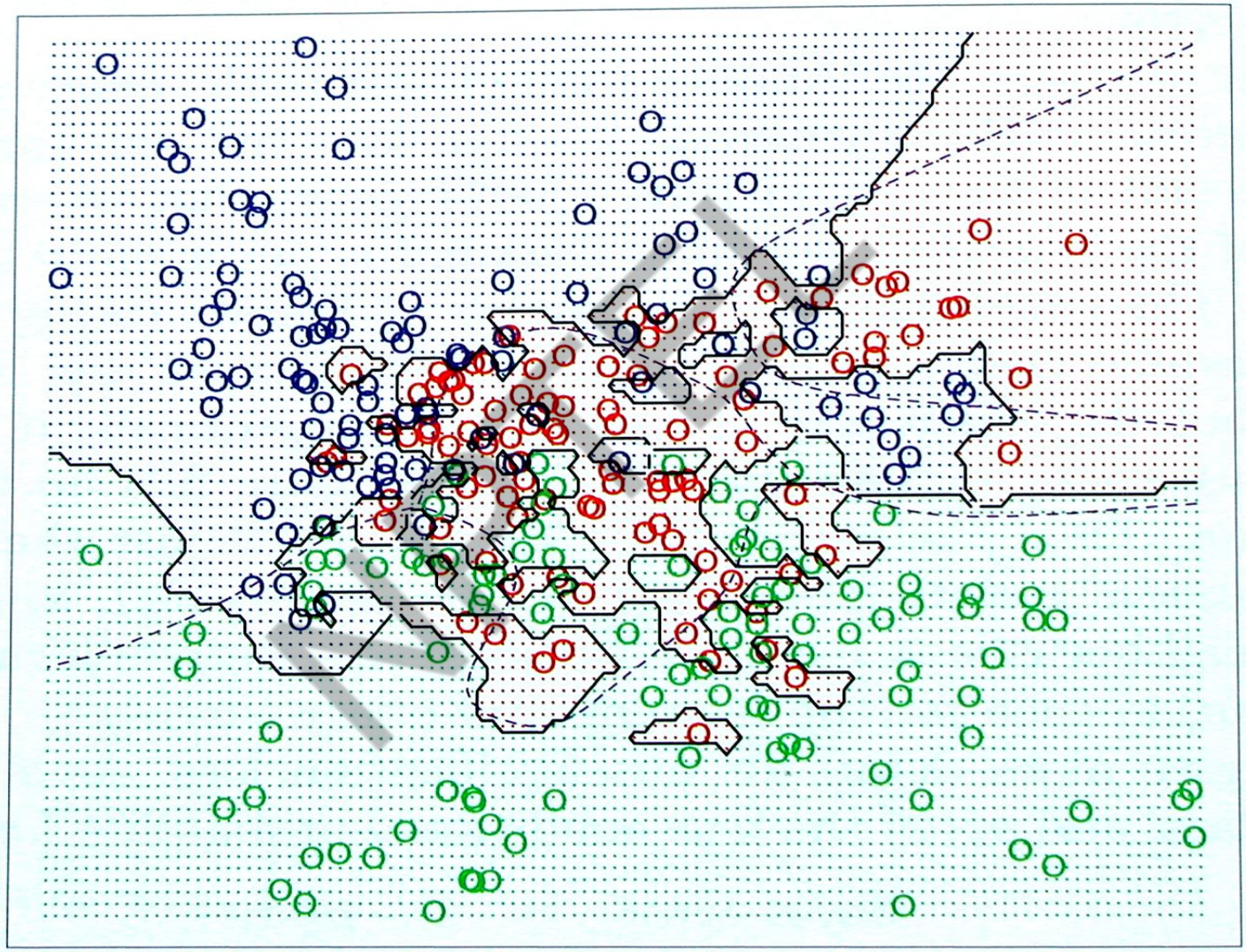
- Average of k points more reliable when:
  - noise in attributes
  - noise in class labels
  - classes partially overlap



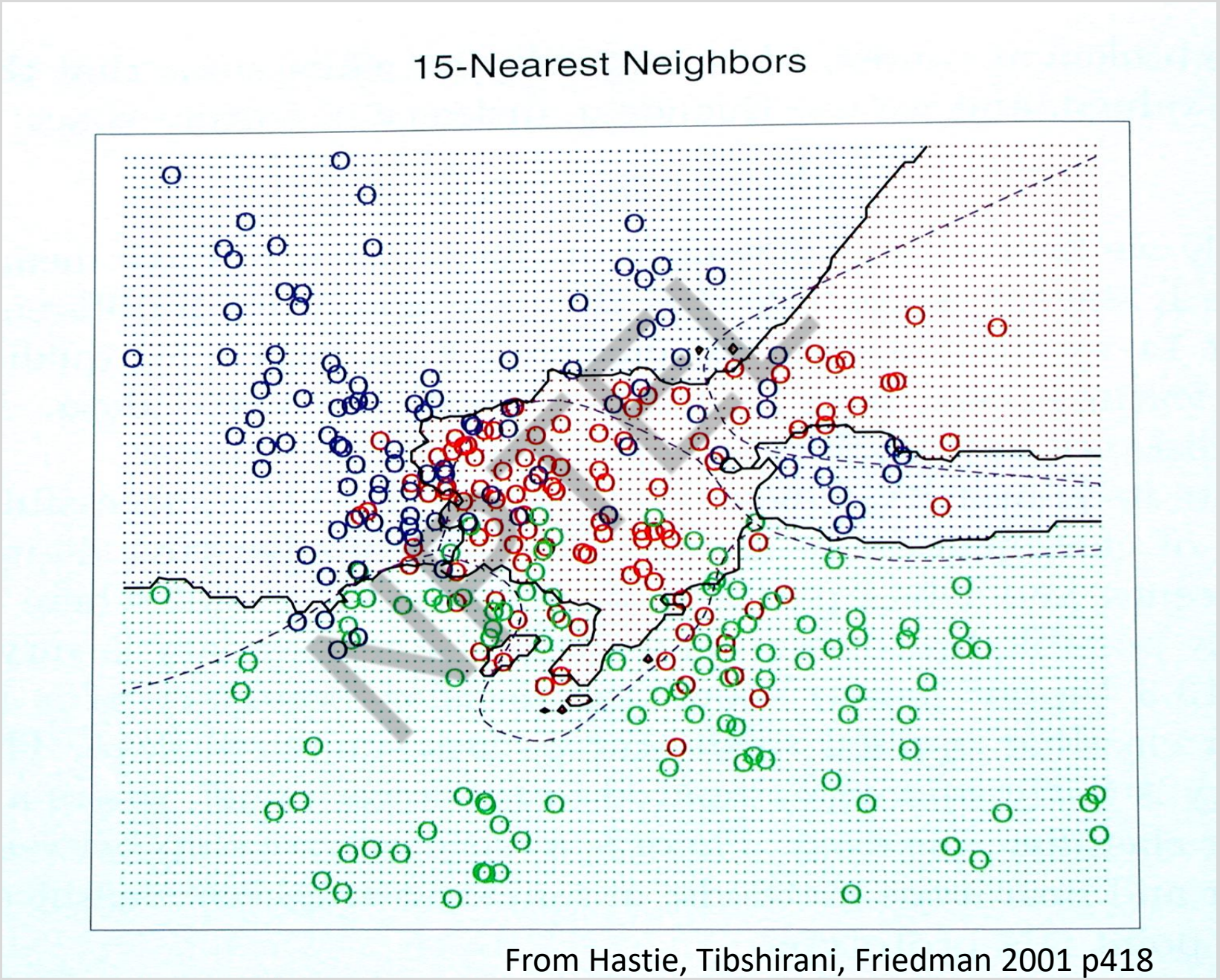
# How to choose “k”

- Large k:
  - less sensitive to noise (particularly class noise)
  - better probability estimates for discrete classes
  - larger training sets allow larger values of k
- Small k:
  - captures fine structure of problem space better
  - may be necessary with small training sets
- Balance must be struck between large and small k
- As training set approaches infinity, and k grows large, kNN becomes Bayes optimal

# 1-Nearest Neighbor

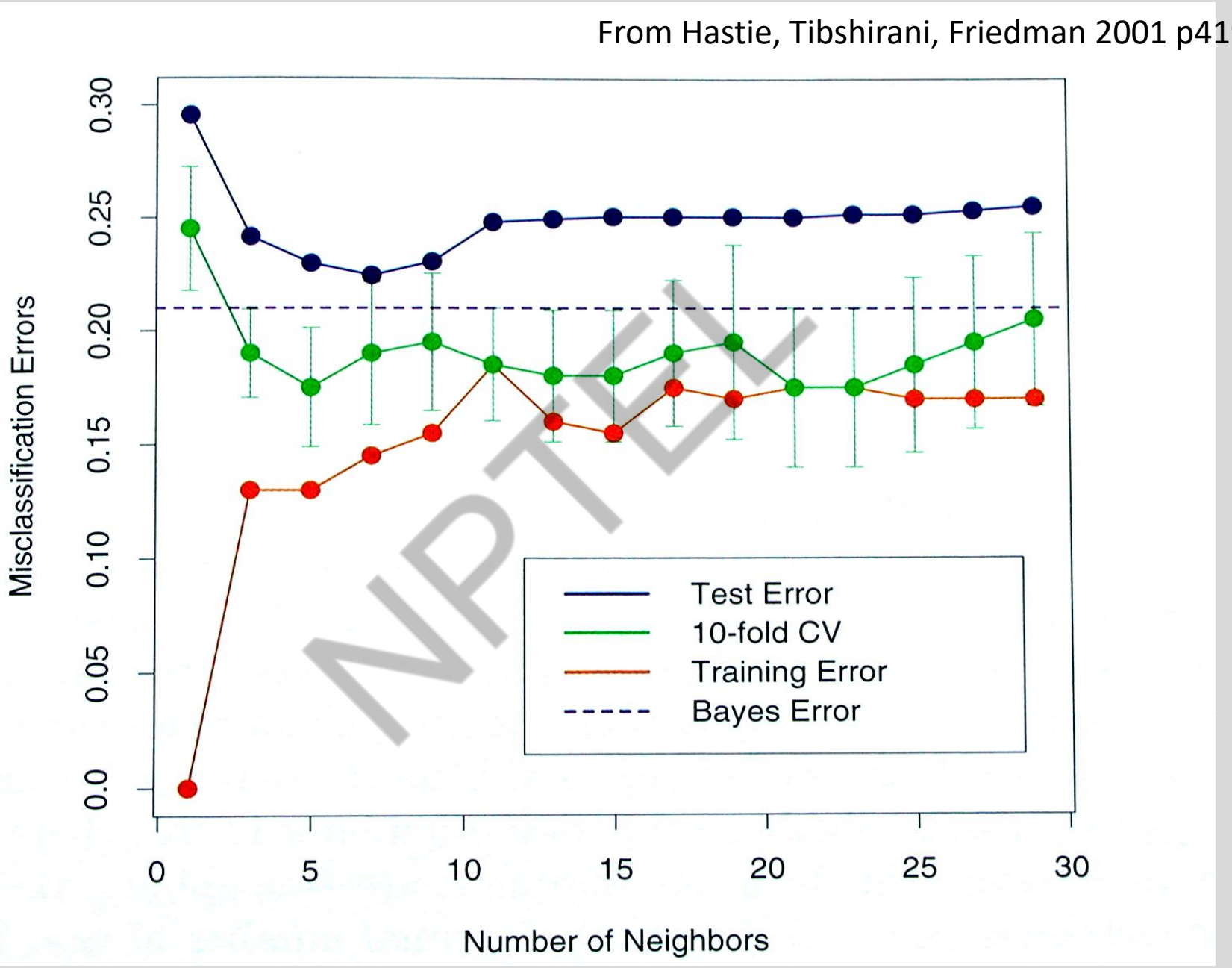


From Hastie, Tibshirani, Friedman 2001 p418




From Hastie, Tibshirani, Friedman 2001 p418

From Hastie, Tibshirani, Friedman 2001 p419



# Distance-Weighted kNN

- tradeoff between small and large k can be difficult
  - use large k, but more emphasis on nearer neighbors?

$$prediction_{test} = \frac{\sum_{i=1}^{k} w_i * class_i}{\sum_{i=1}^{k} w_i} \left( or \frac{\sum_{i=1}^{k} w_i * value_i}{\sum_{i=1}^{k} w_i} \right)$$

$$w_k = \frac{1}{Dist(c_k, c_{test})}$$

# Locally Weighted Averaging

- Let k = number of training points
- Let weight fall-off rapidly with distance

$$prediction_{test} = \frac{\sum_{i=1}^{k} w_i * class_i}{\sum_{i=1}^{k} w_i} \left( or \frac{\sum_{i=1}^{k} w_i * value_i}{\sum_{i=1}^{k} w_i} \right)$$

$$w_k = \frac{1}{e^{KernelWidth \cdot Dist(c_k, c_{test})}}$$

- KernelWidth controls size of neighborhood that has large effect on value (analogous to k)

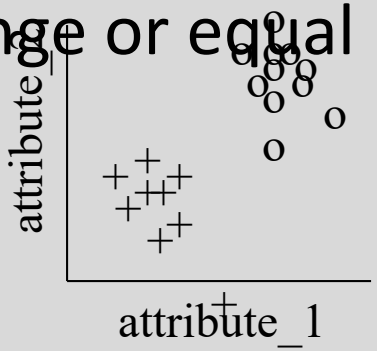
# Locally Weighted Regression

- All algs so far are strict averagers: interpolate, but can't extrapolate
- Do weighted regression, centered at test point, weight controlled by distance and KernelWidth
- Local regressor can be linear, quadratic, n-th degree polynomial, neural net, …
- Yields piecewise approximation to surface that typically is more complex than local regressor

# Euclidean Distance

$$D(c1,c2) = \sqrt{\sum_{i=1}^{N} \left(attr_i(c1) - attr_i(c2)\right)^2}$$

- gives all attributes equal weight?
  - only if scale of attributes and differences are similar
  - scale attributes to equal range or equal variance
- assumes spherical classes



# Euclidean Distance?


- if classes are not spherical?
- if some attributes are more/less important than other attributes?
- if some attributes have more/less noise in them than other attributes?

# Weighted Euclidean Distance

$$D(c1,c2)=\sqrt{\sum_{i=1}^{N} w_i \cdot \left(attr_i(c1)-attr_i(c2)\right)^2}$$

- large weights => attribute is more important
- small weights => attribute is less important
- zero weights => attribute doesn't matter

- Weights allow kNN to be effective with axis-parallel elliptical classes
- Where do weights come from?

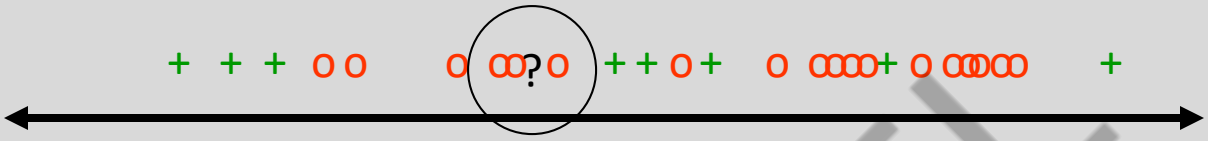
# Curse of Dimensionality

- as number of dimensions increases, distance between points becomes larger and more uniform
- if number of relevant attributes is fixed, increasing the number of less relevant attributes may swamp distance

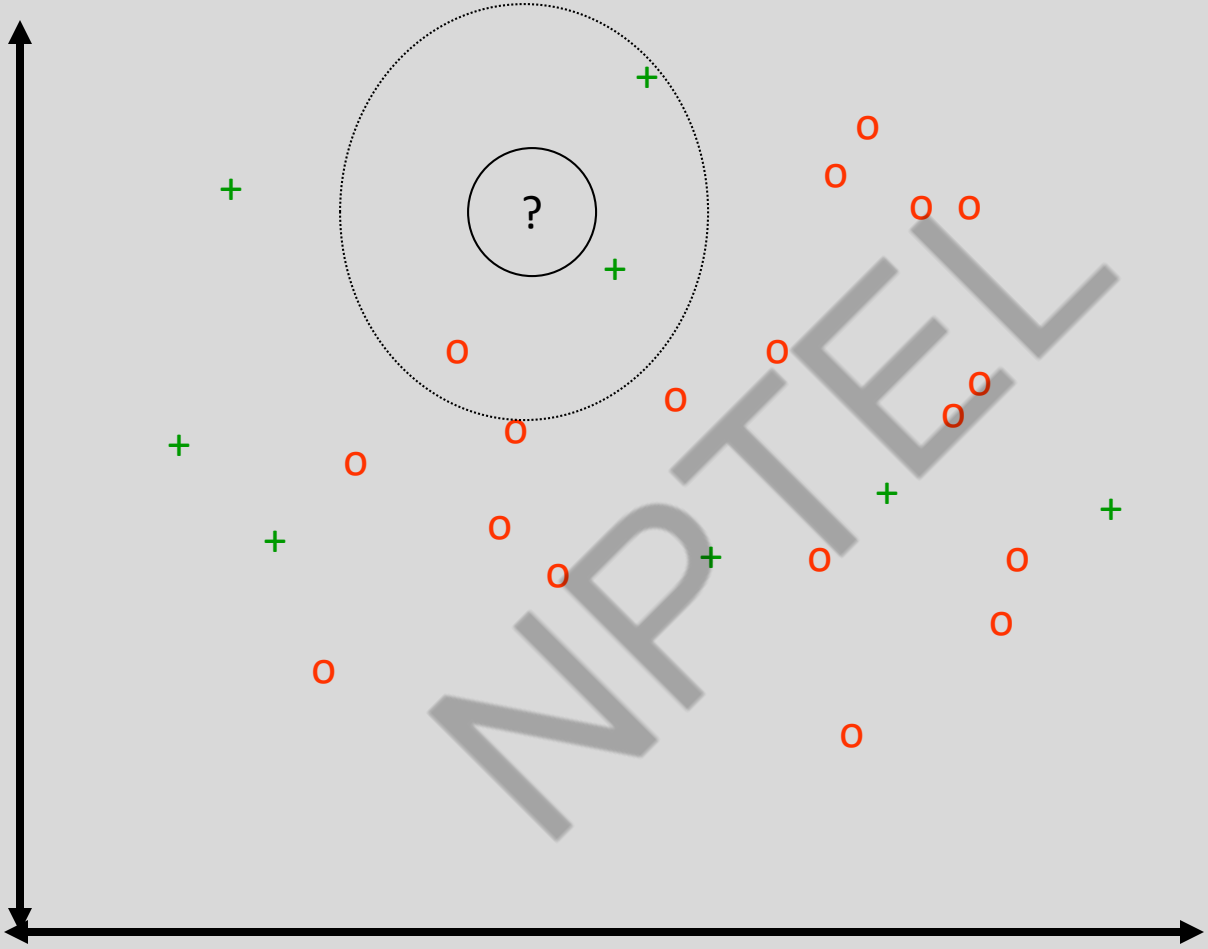
- when more irrelevant than relevant dimensions, distance becomes less reliable
- solutions: larger k or KernelWidth, feature selection, feature weights, more complex distance functions

$$D(c1,c2) = \sqrt{\sum_{i=1}^{relevant} \left(attr_i(c1) - attr_i(c2)\right)^2 + \sum_{j=1}^{irrelevant} \left(attr_j(c1) - attr_j(c2)\right)^2}$$

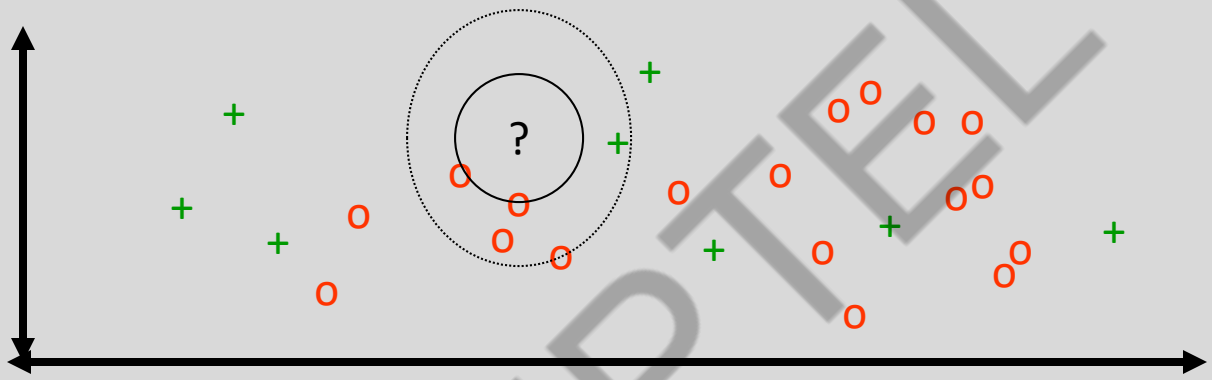
# K-NN and irrelevant features



# K-NN and irrelevant features



# K-NN and irrelevant features



# Ways of rescaling for KNN

Normalized L1 distance:

$$\Delta(X,Y) = \sum_{i=1}^{n} \delta(x_i, y_i)|$$

where:

$$\delta(x_i, y_i) = \begin{cases} abs(\frac{x_i - y_i}{max_i - min_i}) & \text{if numeric, else} \\ 0 & \text{if } x_i = y_i \\ 1 & \text{if } x_i \neq y_i \end{cases}$$

Scale by IG:

$$\Delta(X,Y) = \sum_{i=1}^{n} w_i \, \delta(x_i, y_i)$$

$$w_i = H(C) - \sum_{v \in V_i} P(v) \times H(C|v)$$

Modified value distance metric:

$$\delta(v_1, v_2) = \sum_{i=1}^{n} |P(C_i|v_1) - P(C_i|v_2)|$$

# Ways of rescaling for KNN

Dot product:

$$\Delta(X,Y) = dot_{max} - \sum_{i=1}^{n} w_i x_i y_i$$

Cosine distance:

$$\Delta(X,Y) = cos_{max} - \frac{\sum_{i=1}^{n} w_i x_i y_i}{\sqrt{\sum_{i=1}^{n} w_i x_i^2 \sum_{i=1}^{n} w_i y_i^2}}$$

TFIDF weights for text: for doc j, feature i: $x_i = tf_{i,j} * idf_i$ :

#occur. of term i in doc j

$$\mathrm{tf}_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

$$\mathrm{idf}_i = \log \frac{|D|}{|\{d_j : t_i \in d_j\}|}$$

#docs in corpus

#docs in corpus that contain term i

# Combining distances to neighbors

Standard KNN:

$$\hat{y} = \arg\max_y C(y, Neighbors(x))$$

$$C(y, D') \equiv |\{(x', y') \in D' : y' = y\}|$$

Distance-weighted KNN:

$$\hat{y} = \arg\max_y C(y, Neighbors(x))$$

$$C(y, D') \equiv \sum_{\{(x', y') \in D' : y' = y\}} (SIM(x, x'))$$

$$C(y, D') \equiv 1 - \prod_{\{(x', y') \in D' : y' = y\}} (1 - SIM(x, x'))$$

$$SIM(x, x') \equiv 1 - \Delta(x, x')$$

# Advantages of Memory-Based Methods

- Lazy learning: don't do any work until you know what you want to predict (and from what variables!)
  - never need to learn a global model
  - many simple local models taken together can represent a more complex global model
  - better focussed learning
  - handles missing values, time varying distributions, ...
- Very efficient cross-validation
- Intelligible learning method to many users
- Nearest neighbors support explanation and training
- Can use ***any*** distance metric: string-edit distance, …

# Weaknesses of Memory-Based Methods

- Curse of Dimensionality:
  - often works best with 25 or fewer dimensions
- Run-time cost scales with training set size
- Large training sets will not fit in memory
- Many MBL methods are strict averagers
- Sometimes doesn't seem to perform as well as other methods such as neural nets
- Predicted values for regression not continuous

# Foundations of Machine Learning

## Module 3: Instance Based Learning and Feature Reduction

## Part B: Feature Selection

Sudeshna Sarkar
IIT Kharagpur

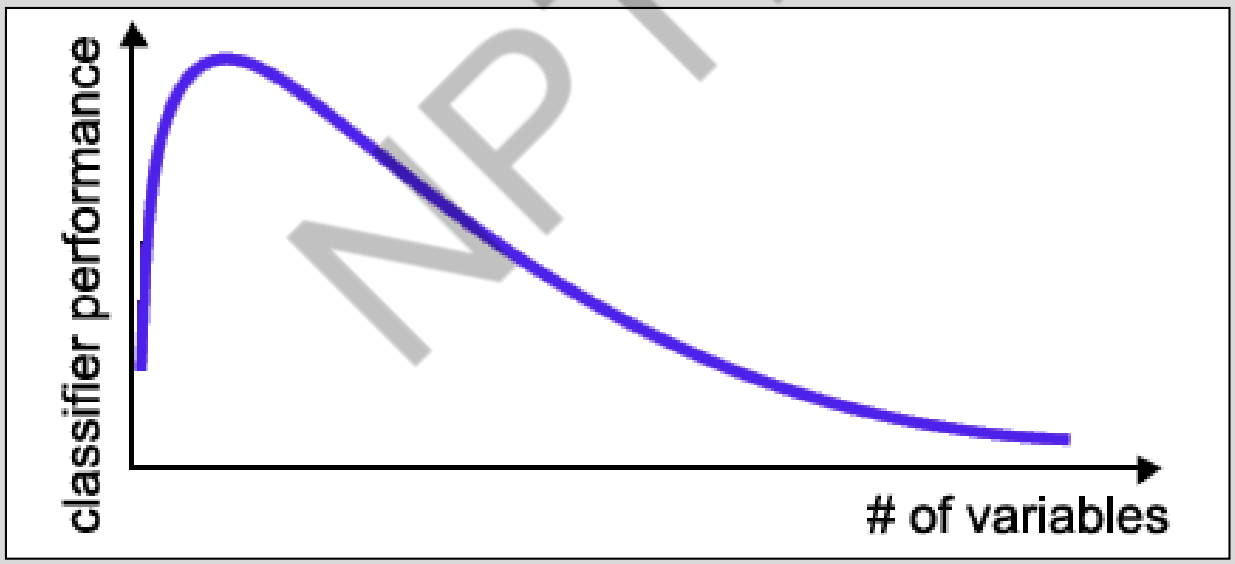
# Feature Reduction in ML

- The information about the target class is **inherent in the variables**.
- Naïve view:
  More features
  => More information
  => More discrimination power.
- In practice:
  **many reasons why this is not the case!**

# Curse of Dimensionality

- number of training examples is fixed

  => the classifier's performance usually will degrade for a large number of features!



# Feature Reduction in ML

- Irrelevant and
- redundant features
  - can confuse learners.

- Limited training data.
- Limited computational resources.
- **Curse of dimensionality**.

# Feature Selection

Problem of selecting some subset of features, while ignoring the rest

# Feature Extraction

- Project the original $x_i$ , $i$ =1,...,$d$ dimensions to new $k < d$ dimensions, $z_j$ , $j$ =1,...,$k$

Criteria for selection/extraction:
either improve or maintain the classification accuracy, simplify classifier complexity.

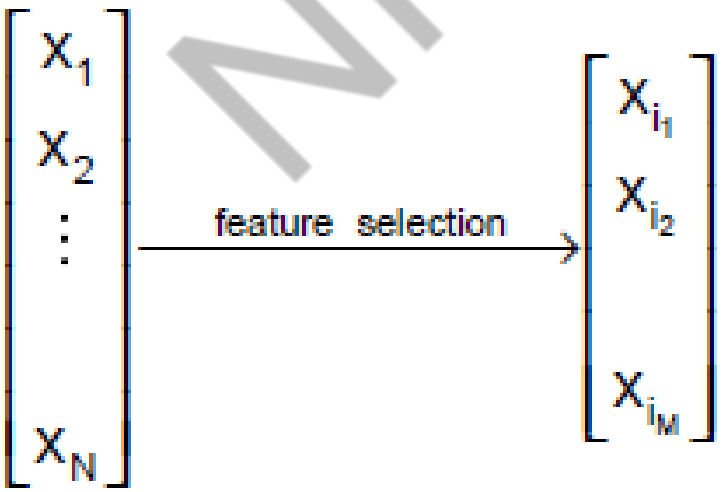
# Feature Selection - Definition

- Given a set of features $F = \{x_1, \ldots, x_n\}$

  the Feature Selection problem is

  to find a subset $F' \subseteq F$ that maximizes the learners ability to classify patterns.

- Formally $F'$ should maximize some scoring function



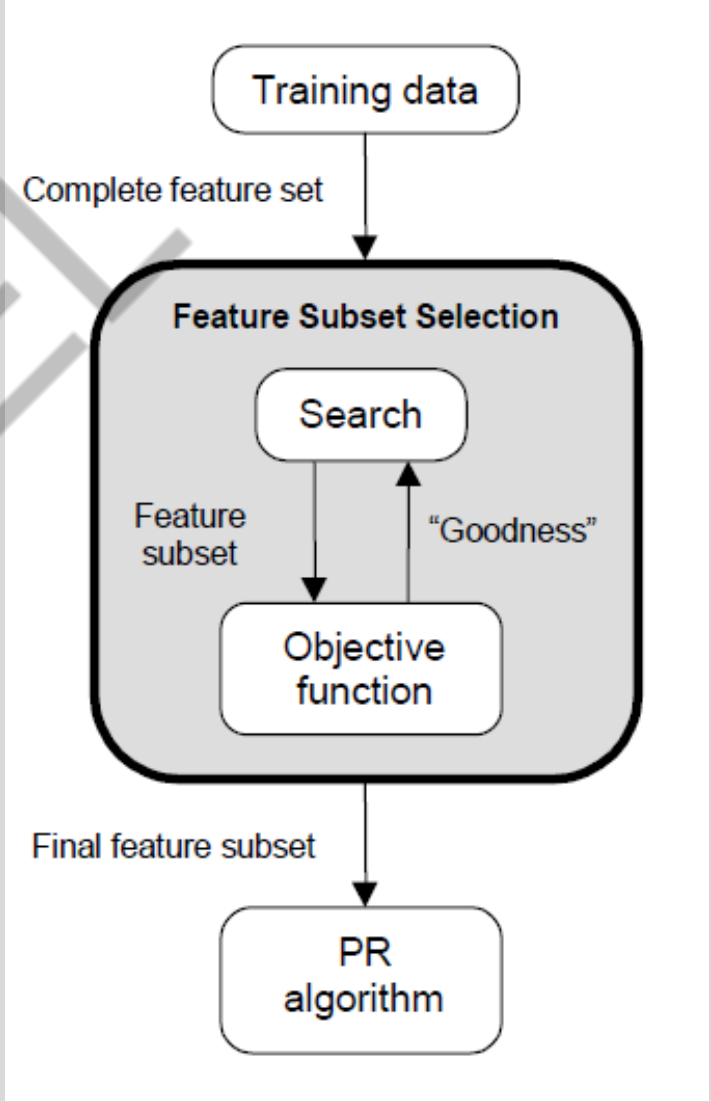
# Subset selection

- $d$ initial features
- There are $2^d$ possible subsets
- Criteria to decide which subset is the best:
  - classifier based on these m features has the lowest probability of error of all such classifiers
- Can't go over all $2^d$ possibilities
- Need some heuristics

# Feature Selection Steps

Feature selection is an **optimization** problem.

- Step 1: Search the space of possible feature subsets.
- Step 2: Pick the subset that is optimal or near-optimal with respect to some objective function.



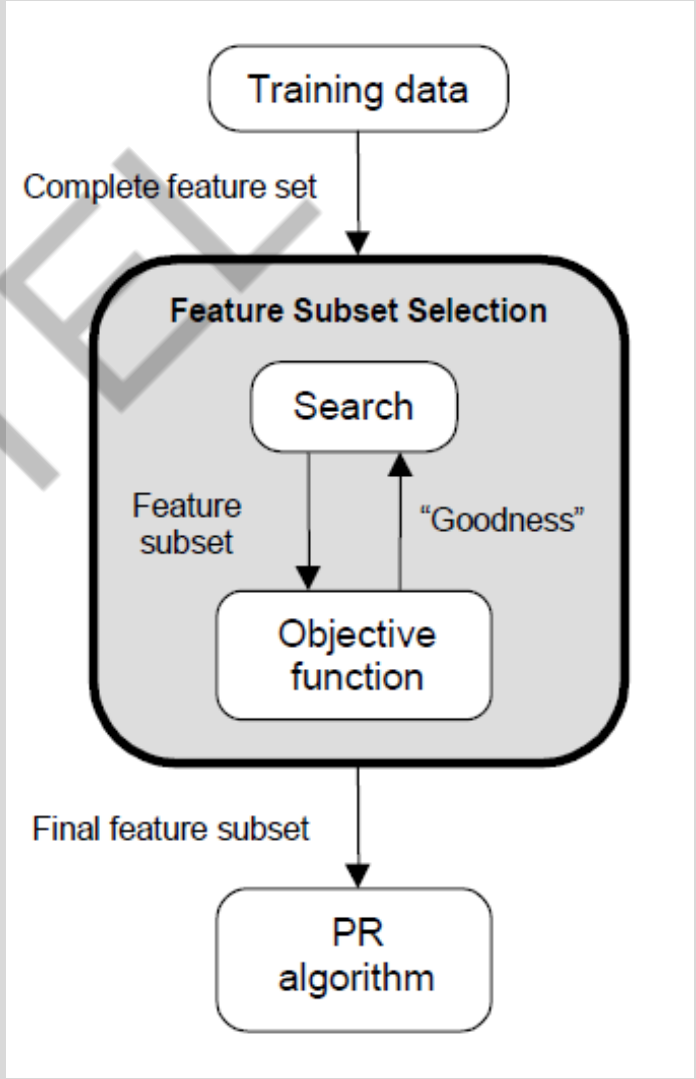
# Feature Selection Steps (cont'd)

Search strategies

- Optimum
- Heuristic
- Randomized

Evaluation strategies

- Filter methods
- Wrapper methods

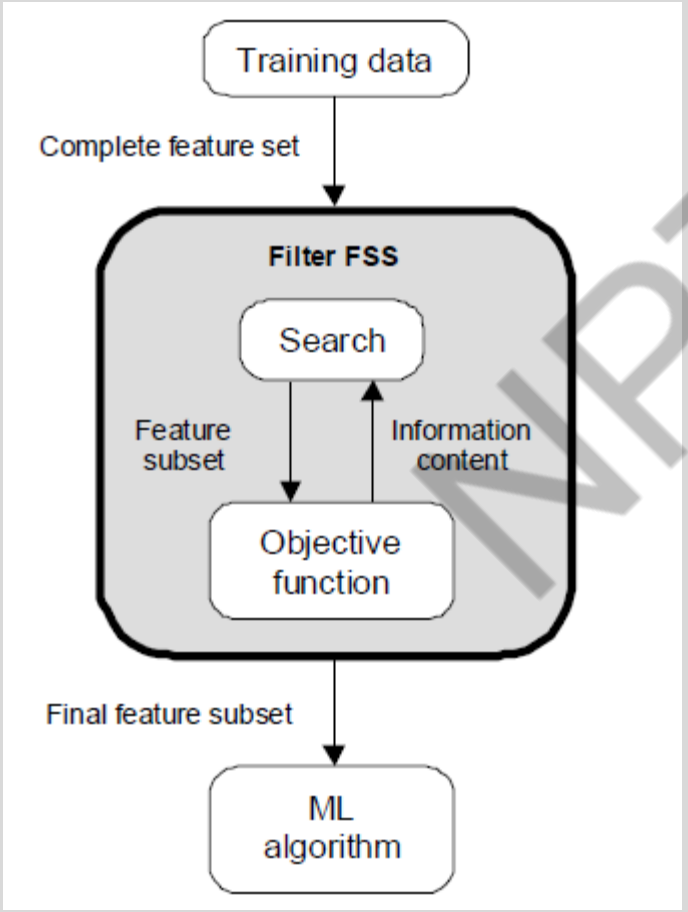


# Evaluating feature subset

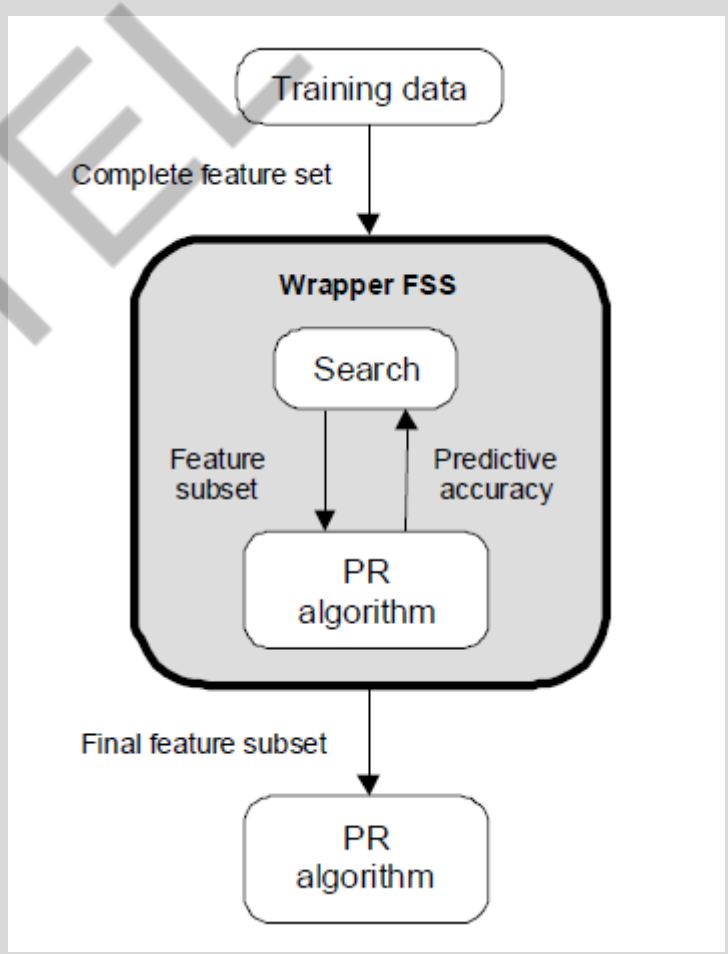
- Supervised (wrapper method)
  - Train using selected subset
  - Estimate error on validation dataset
- Unsupervised (filter method)
  - Look at input only
  - Select the subset that has the most information

# Evaluation Strategies

## Filter Methods




## Wrapper Methods



# Subset selection

- Select uncorrelated features
- Forward search
  - Start from empty set of features
  - Try each of remaining features
  - Estimate classification/regression error for adding specific feature
  - Select feature that gives maximum improvement in validation error
  - Stop when no significant improvement
- Backward search
  - Start with original set of size *d*
  - Drop features with smallest impact on error

# Feature selection

Univariate (looks at each feature independently of others)

- Pearson correlation coefficient
- F-score
- Chi-square
- Signal to noise ratio
- mutual information
- Etc.

Univariate methods measure some type of correlation between two random variables

- the label ($y_i$) and a fixed feature ($x_{ij}$ for fixed j)

- Rank features by importance
- Ranking cut-off is determined by user

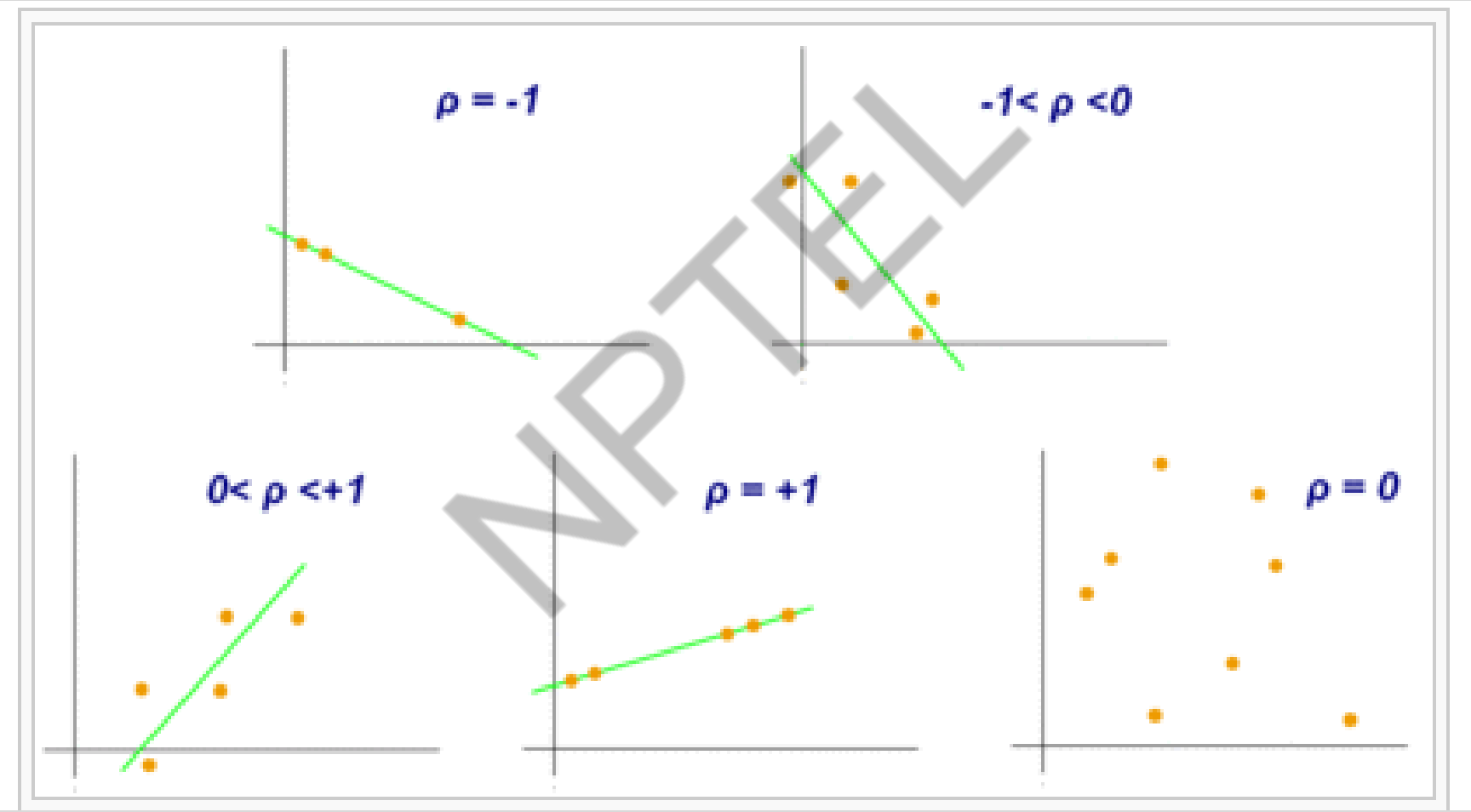
# Pearson correlation coefficient

- Measures the correlation between two variables
- Formula for Pearson correlation =

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}}$$

- The correlation r is between $+1$ and $-1$.
  - $+1$ means perfect positive correlation
  - $-1$ in the other direction

# Pearson correlation coefficient






From Wikipedia

# Signal to noise ratio

- Difference in means divided by difference in standard deviation between the two classes

$$S2N(X,Y) = (\mu_X - \mu_Y)/(\sigma_X - \sigma_Y)$$

- Large values indicate a strong correlation

# Multivariate feature selection

- Multivariate (considers all features simultaneously)
- Consider the vector w for any linear classifier.
- Classification of a point x is given by $w^T x + w_0$.
- Small entries of w will have little effect on the dot product and therefore those features are less relevant.
- For example if w = (10, .01, -9) then features 0 and 2 are contributing more to the dot product than feature 1.
  - A ranking of features given by this w is 0, 2, 1.

# Multivariate feature selection

- The w can be obtained by any of linear classifiers
- A variant of this approach is called recursive feature elimination:
  - Compute w on all features
  - Remove feature with smallest $w_i$
  - Recompute w on reduced data
  - If stopping criterion not met then go to step 2

# Feature Extraction

# Foundations of Machine Learning

## Module 3: Instance Based Learning and Feature Reduction

## Part C: Feature Extraction

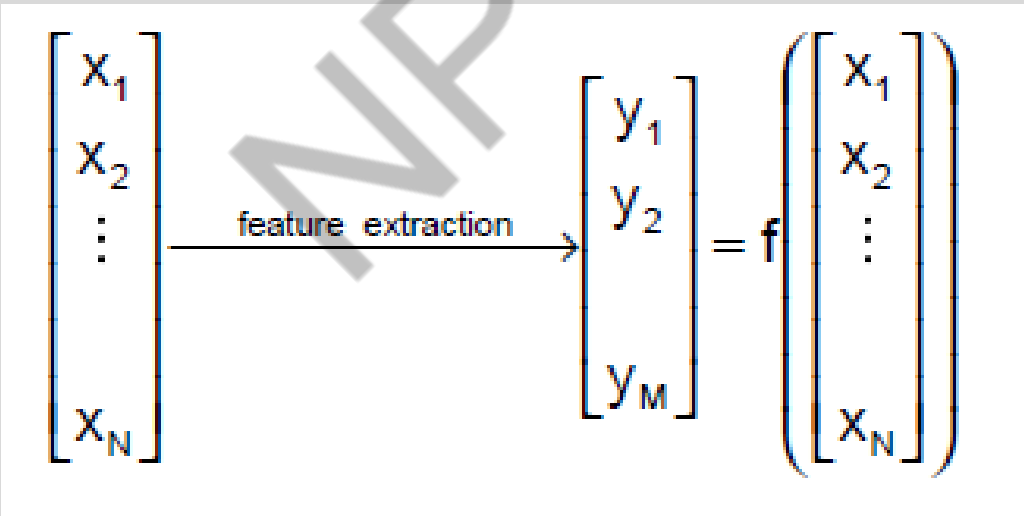
Sudeshna Sarkar
IIT Kharagpur

# Feature extraction - definition

- Given a set of features $F = \{x_1, \ldots, x_N\}$

  the Feature Extraction("Construction") problem is

  is to map $F$ to some feature set $F''$ that maximizes the learner's ability to classify patterns



# Feature Extraction

- Find a projection matrix w from N-dimensional to M-dimensional vectors that keeps error low

$$\boldsymbol{z} = w^T \boldsymbol{x}$$

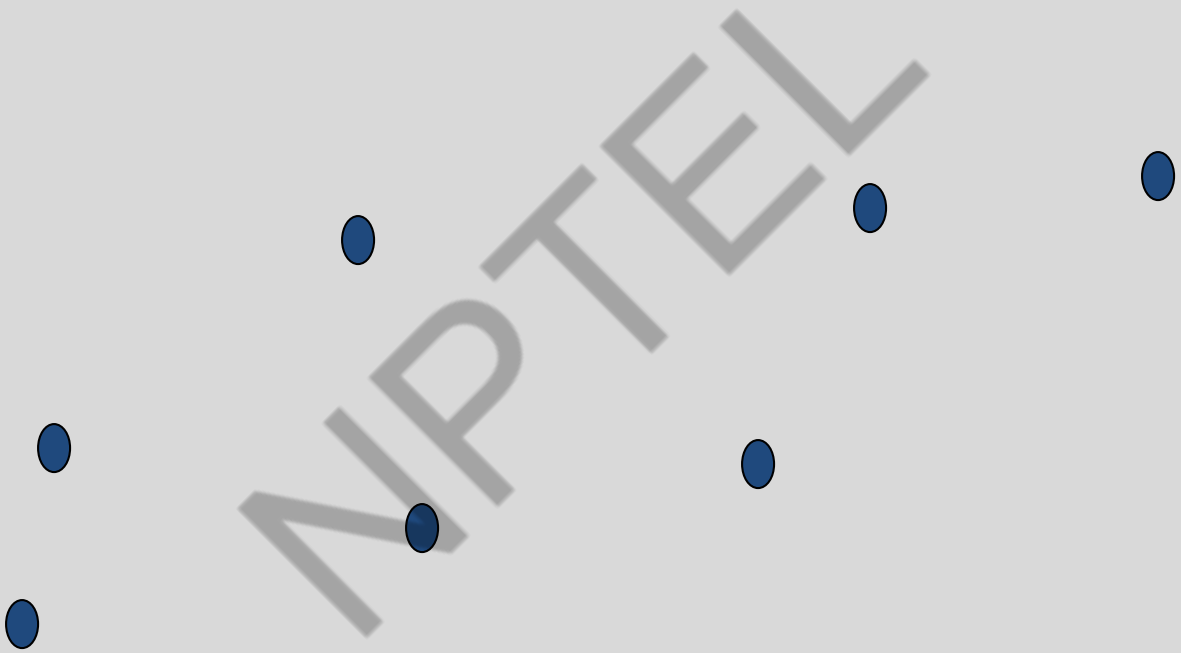
# PCA

- Assume that N features are linear combination of $\mathrm{M} < N$ vectors

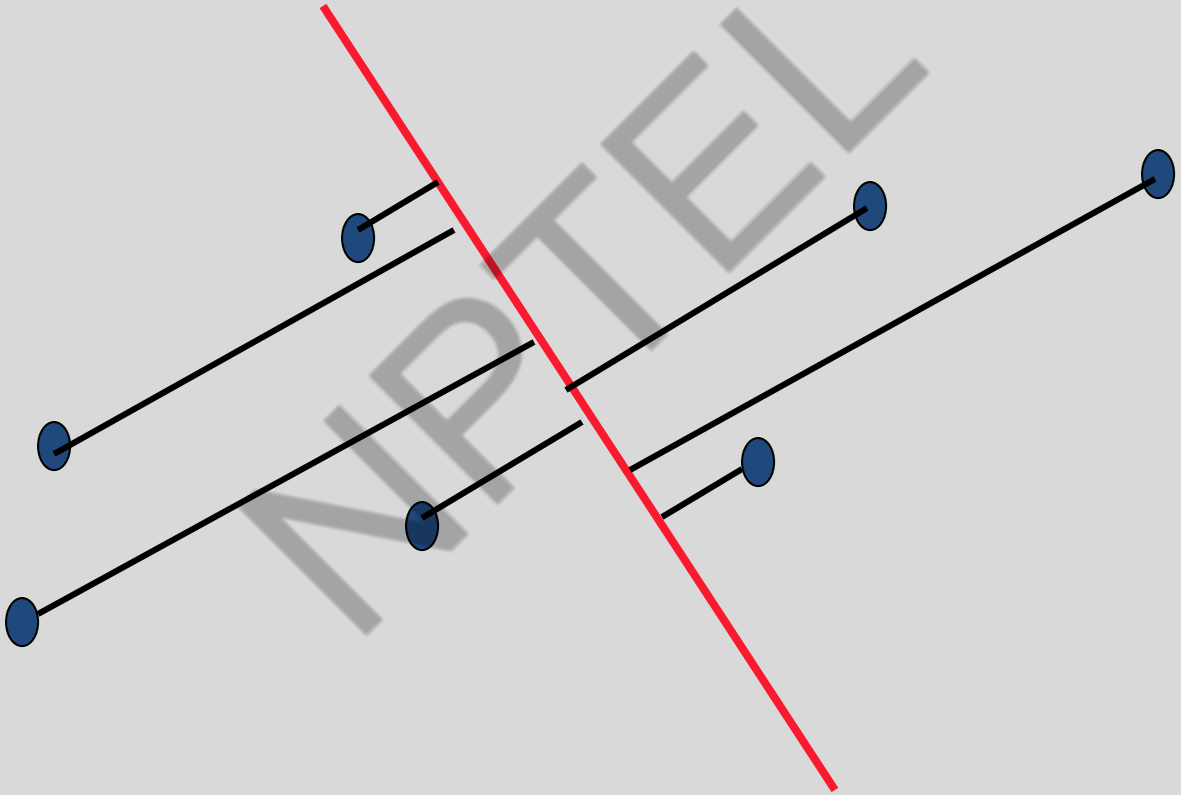
$$z_i = w_{i1}x_{i1} + \cdots + w_{id}x_{iN}$$

- What we expect from such basis
  - Uncorrelated or otherwise can be reduced further
  - Have large variance (e.g. $w_{i1}$ have large variation) or otherwise bear no information

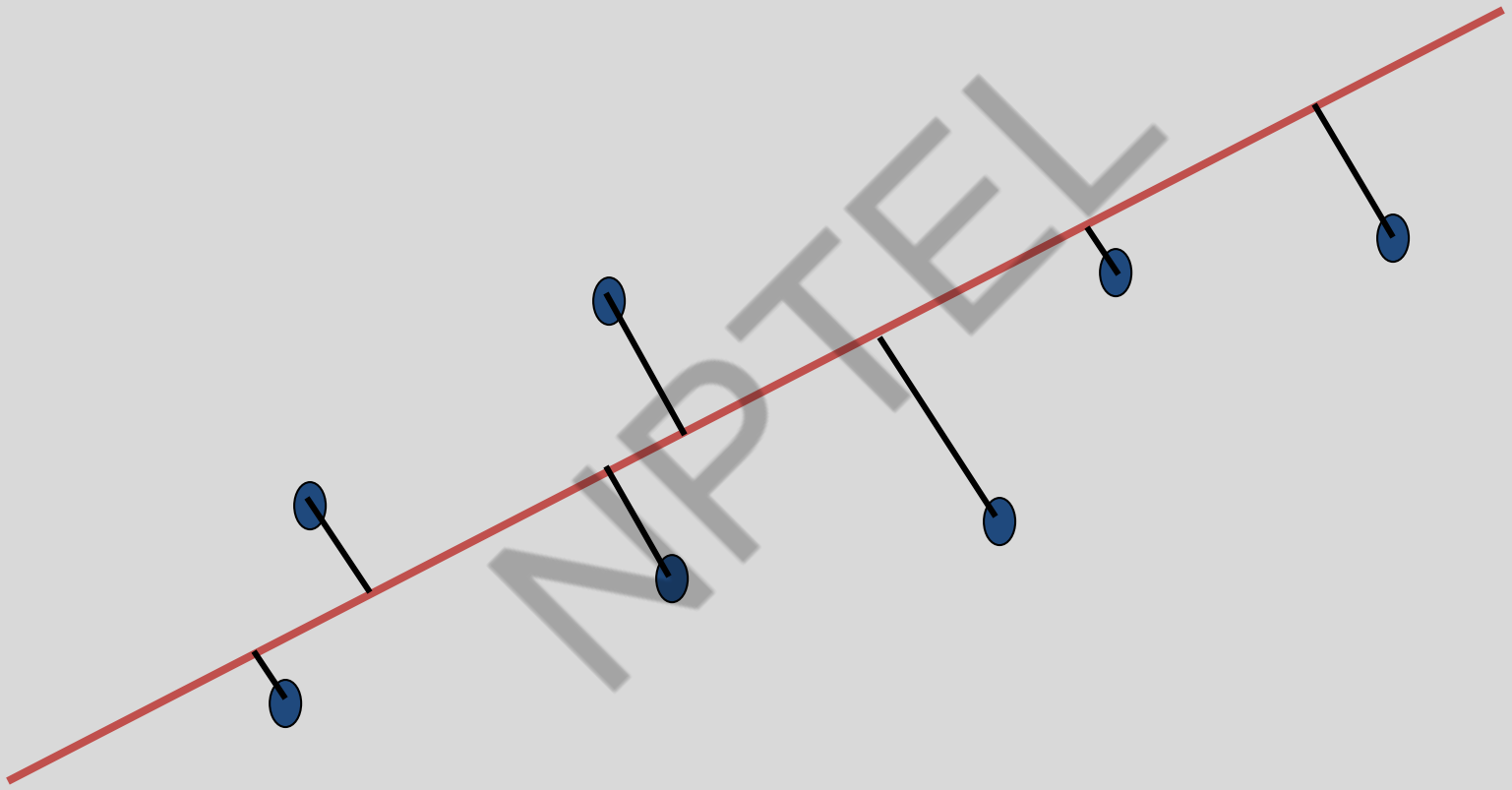
# Geometric picture of principal components (PCs)



# Geometric picture of principal components (PCs)



# Geometric picture of principal components (PCs)



# Algebraic definition of PCs

Given a sample of *p* observations on a vector of *N* variables

$$\{x_1, x_2, \cdots, x_p\} \in \Re^N$$

define the first principal component of the sample by the linear transformation

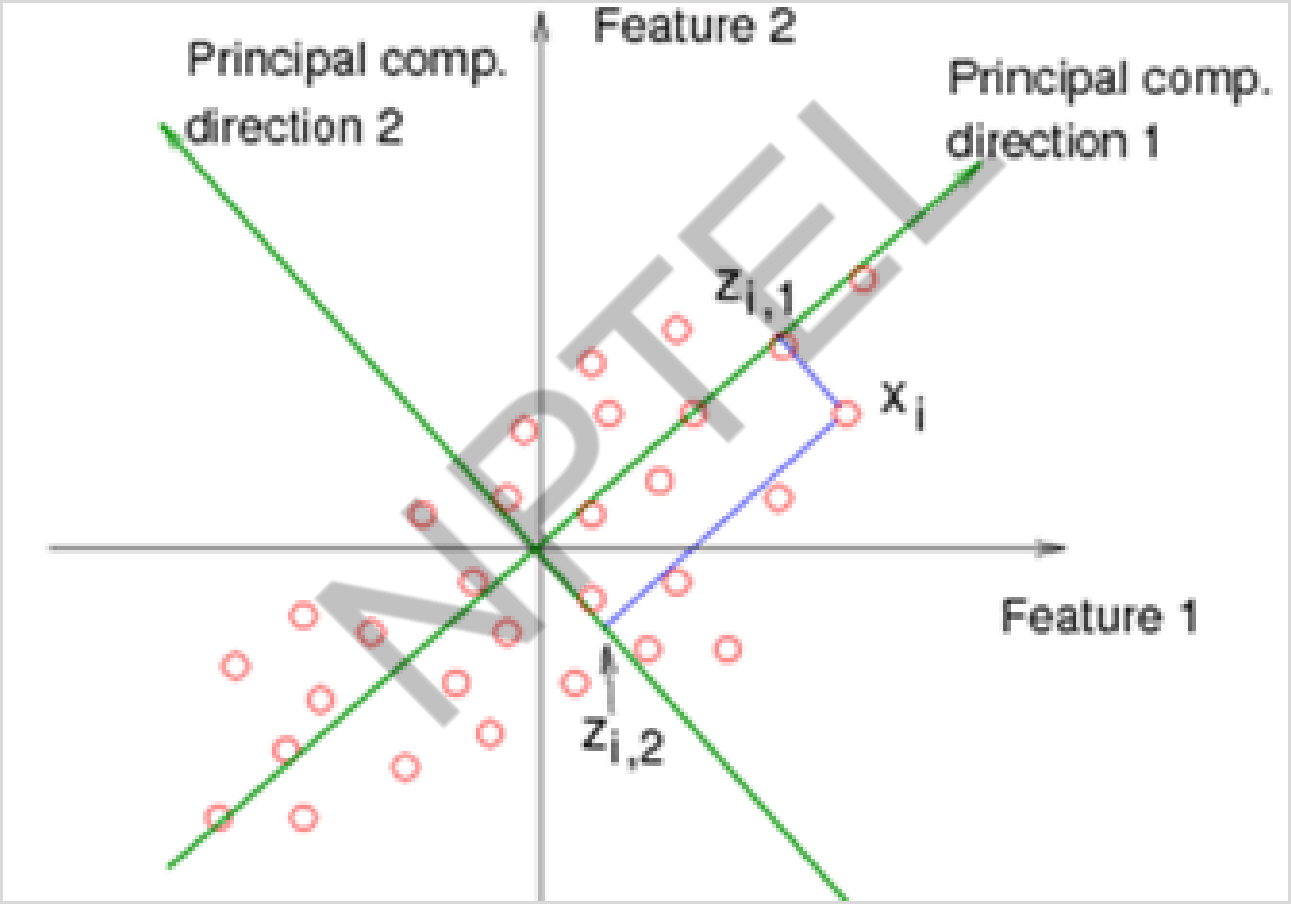
$$z_1 = a_1^T x_j = \sum_{i=1}^{N} a_{i1} x_{ij}, \quad j = 1,2,\cdots, p.$$

where the vector $a_1 = (a_{11}, a_{21}, \cdots, a_{N1})$

$$x_j = (x_{1j}, x_{2j}, \cdots, x_{Nj})$$

is chosen such that $\mathrm{var}[z_1]$ is maximum.

# PCA


Feature 2
Principal comp.
direction 2
Principal comp.
direction 1
$z_{i,1}$
$x_i$
Feature 1
$z_{i,2}$

- Choose directions such that a total variance of data will be maximum
  - Maximize Total Variance
- Choose directions that are orthogonal
  - Minimize correlation
- Choose $M < N$ orthogonal directions which maximize total variance

# PCA

- $N$-dimensional feature space
- $\mathrm{N} \times N$ symmetric covariance matrix estimated from samples $Cov(\boldsymbol{x}) = \Sigma$
- Select $M$ largest eigenvalue of the covariance matrix and associated $M$ eigenvectors
- The first eigenvector will be a direction with largest variance

# PCA for image compression

p=1

p=2

p=4

p=8

p=16

p=32

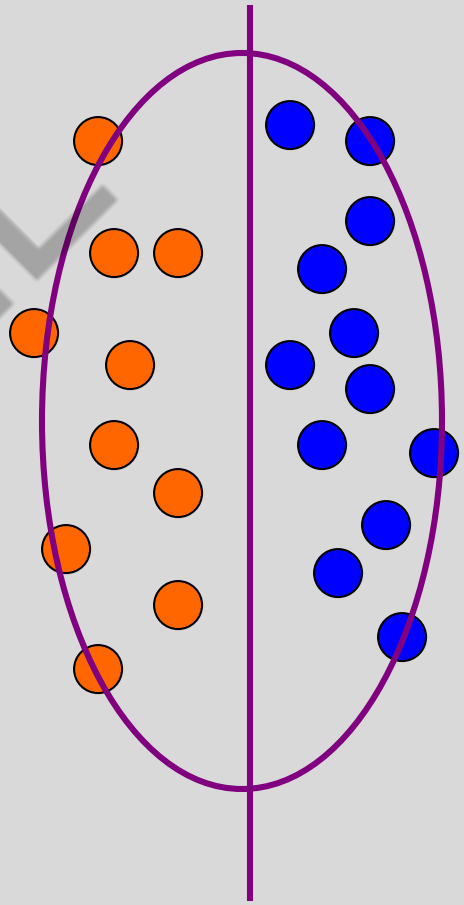
p=64

p=100

Original Image

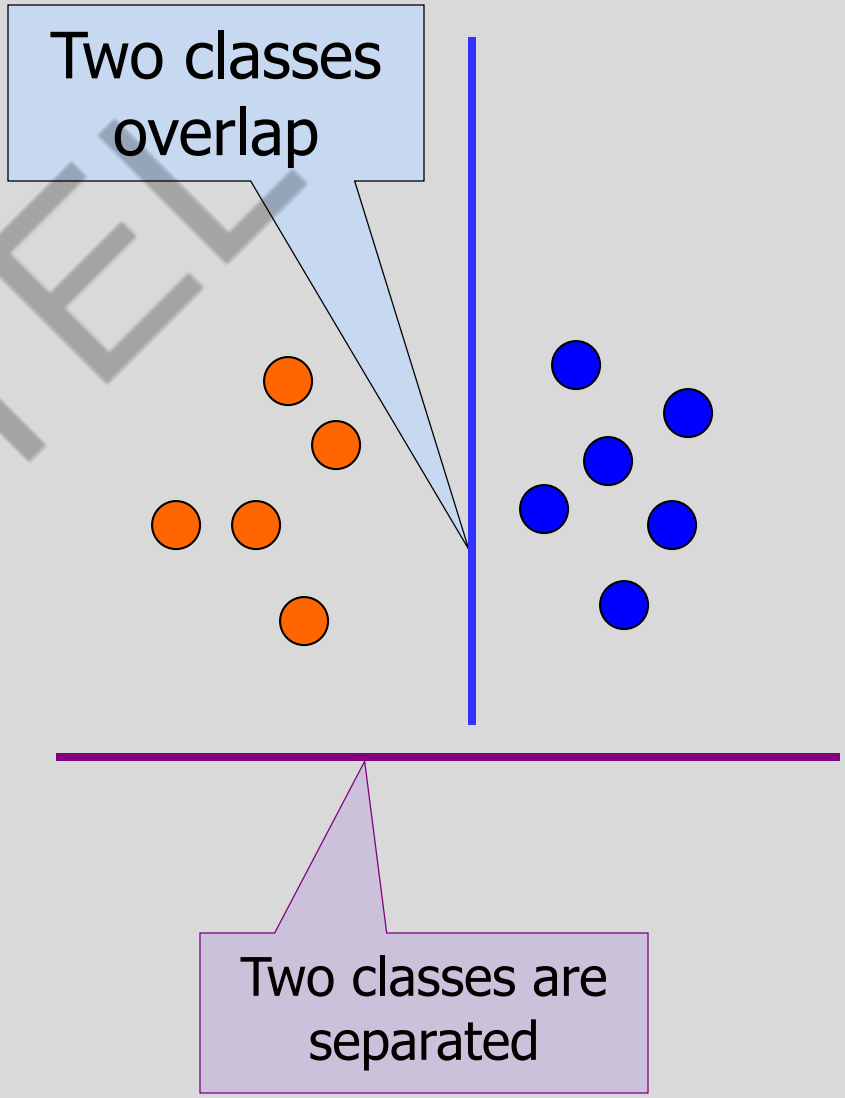
# Is PCA a good criterion for classification?

- Data variation determines the projection direction
- What's missing?
  - Class information



# What is a good projection?

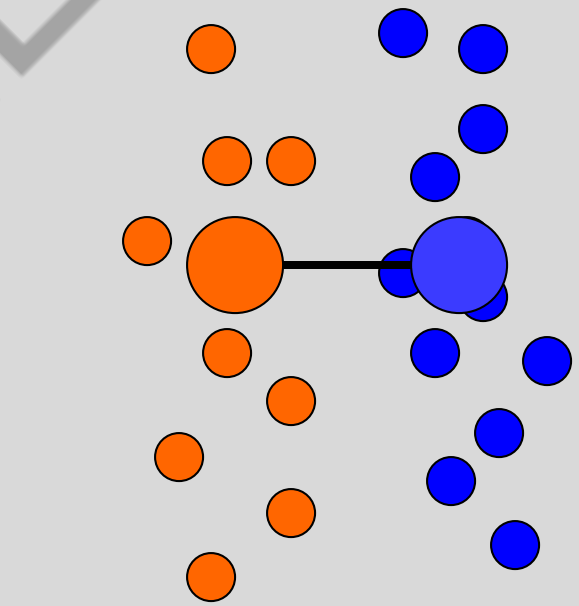
- Similarly, what is a good criterion?
  - Separating different classes



# What class information may be useful?

- Between-class distance
  - Distance between the centroids of different classes



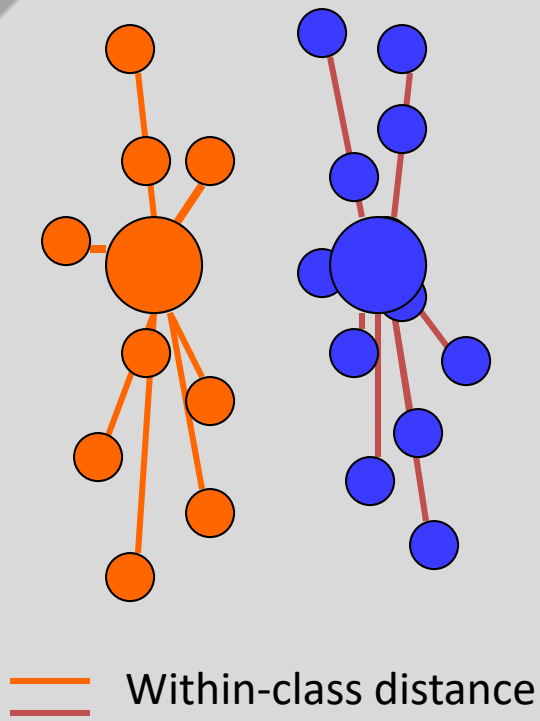


Between-class distance

# What class information may be useful?

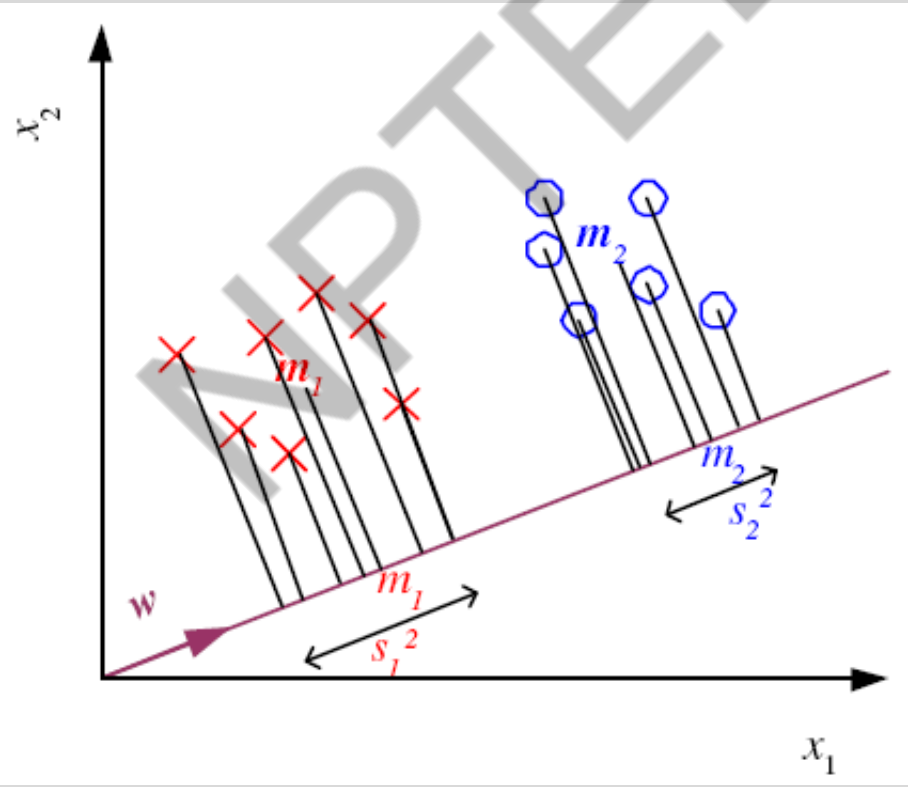
- Between-class distance
  - Distance between the centroids of different classes
- Within-class distance
  - Accumulated distance of an instance to the centroid of its class

- Linear discriminant analysis (LDA) finds most discriminant projection by
  - maximizing between-class distance
  - and minimizing within-class distance



# Linear Discriminant Analysis

- Find a low-dimensional space such that when $\boldsymbol{x}$ is projected, classes are well-separated



# Means and Scatter after projection

$$m_1 = \frac{\sum_t \mathbf{w}^T \mathbf{x}^t r^t}{\sum_t r^t} = \mathbf{w}^T \mathbf{m}_1$$

$$m_2 = \frac{\sum_t \mathbf{w}^T \mathbf{x}^t (1 - r^t)}{\sum_t (1 - r^t)} = \mathbf{w}^T \mathbf{m}_2$$

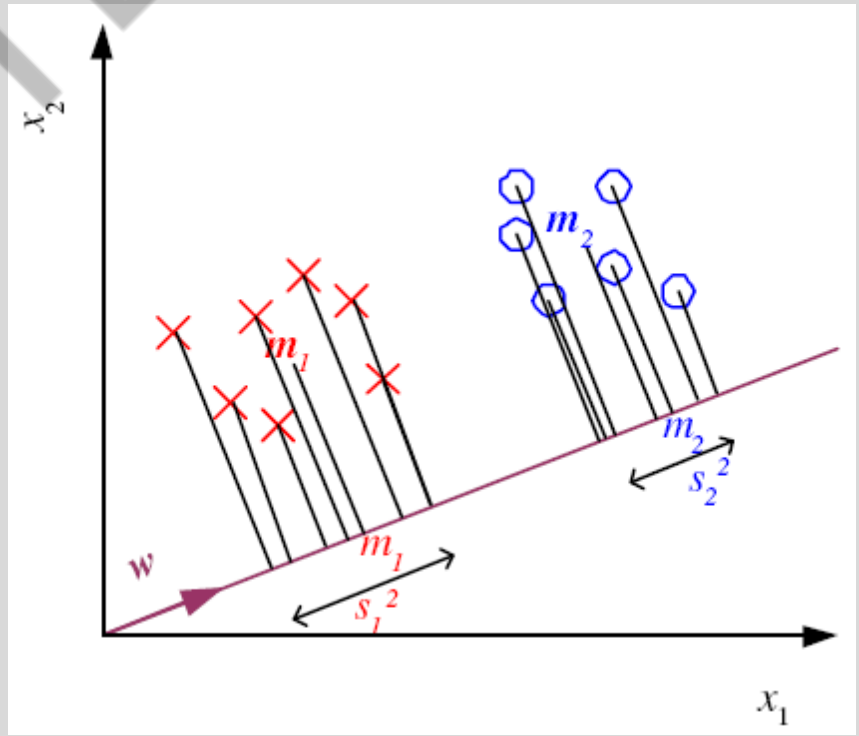
$$s_1^2 = \sum_t (\mathbf{w}^T \mathbf{x}^t - m_1)^2 r^t$$

$$s_2^2 = \sum_t (\mathbf{w}^T \mathbf{x}^t - m_2)^2 (1 - r^t)$$

# Good Projection

- Means are as far away as possible
- Scatter is small as possible
- Fisher Linear Discriminant

$$J(\mathbf{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2}$$



# Multiple Classes

- For $c$ classes, compute $c - 1$ discriminants, project N-dimensional features into $c - 1$ space.

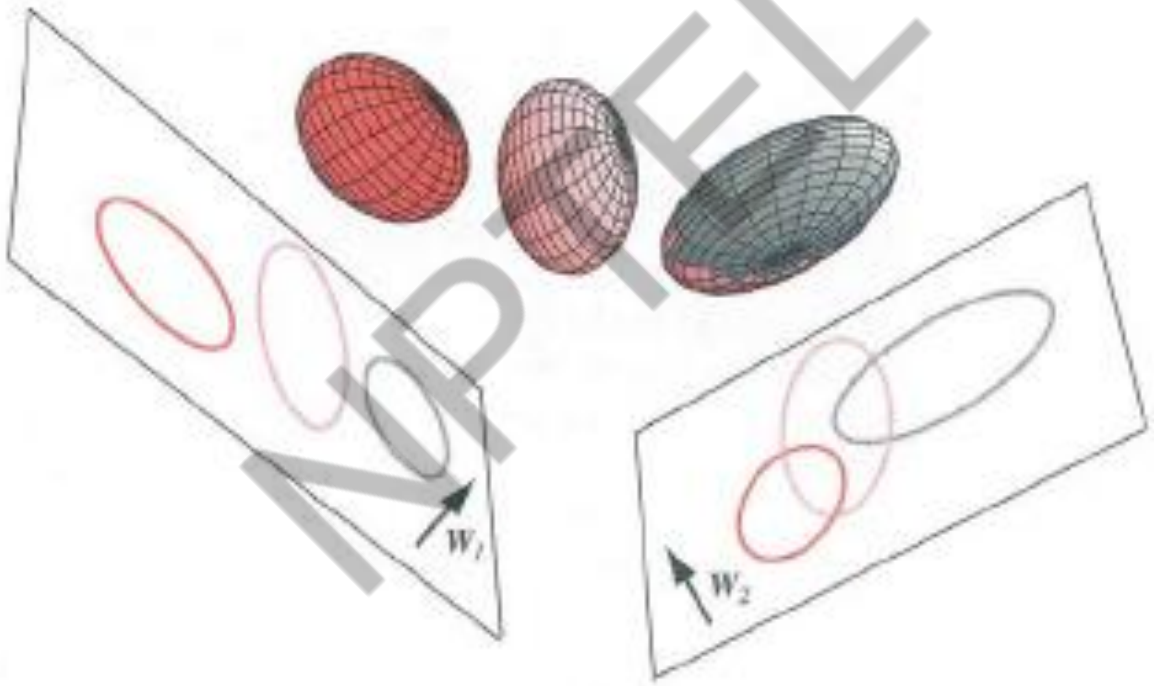



FIGURE 3.6. Three three-dimensional distributions are projected onto two-dimensional subspaces, described by a normal vectors $\mathbf{W}_1$ and $\mathbf{W}_2$. Informally, multiple discriminant methods seek the optimum such subspace, that is, the one with the greatest separation of the projected distributions for a given total within-scatter matrix, here as associated with $\mathbf{W}_1$.

# Foundations of Machine Learning

## Module 3: Instance Based Learning and Feature Reduction

## Part C: Feature Extraction

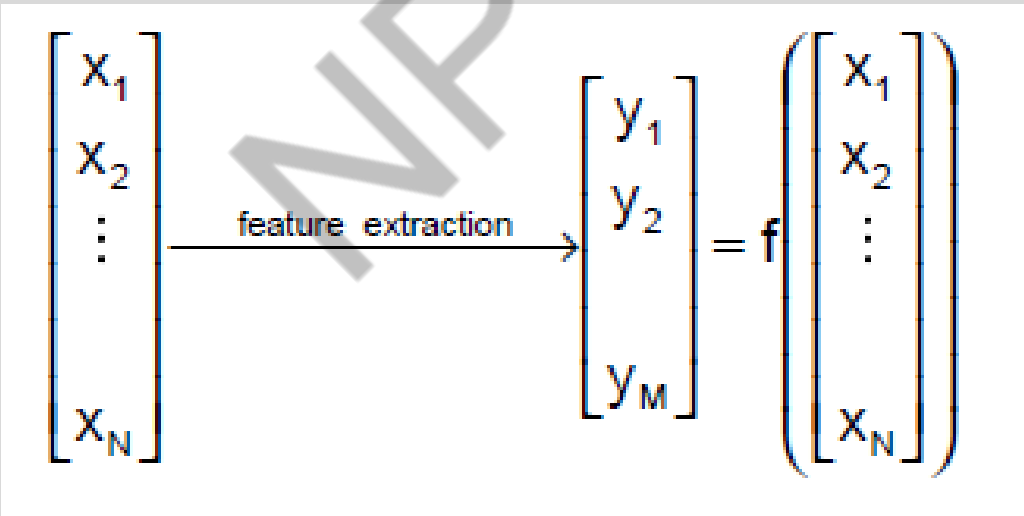
Sudeshna Sarkar
IIT Kharagpur

# Feature extraction - definition

- Given a set of features $F = \{x_1, \ldots, x_N\}$

  the Feature Extraction("Construction") problem is

  is to map $F$ to some feature set $F''$ that maximizes the learner's ability to classify patterns



# Feature Extraction

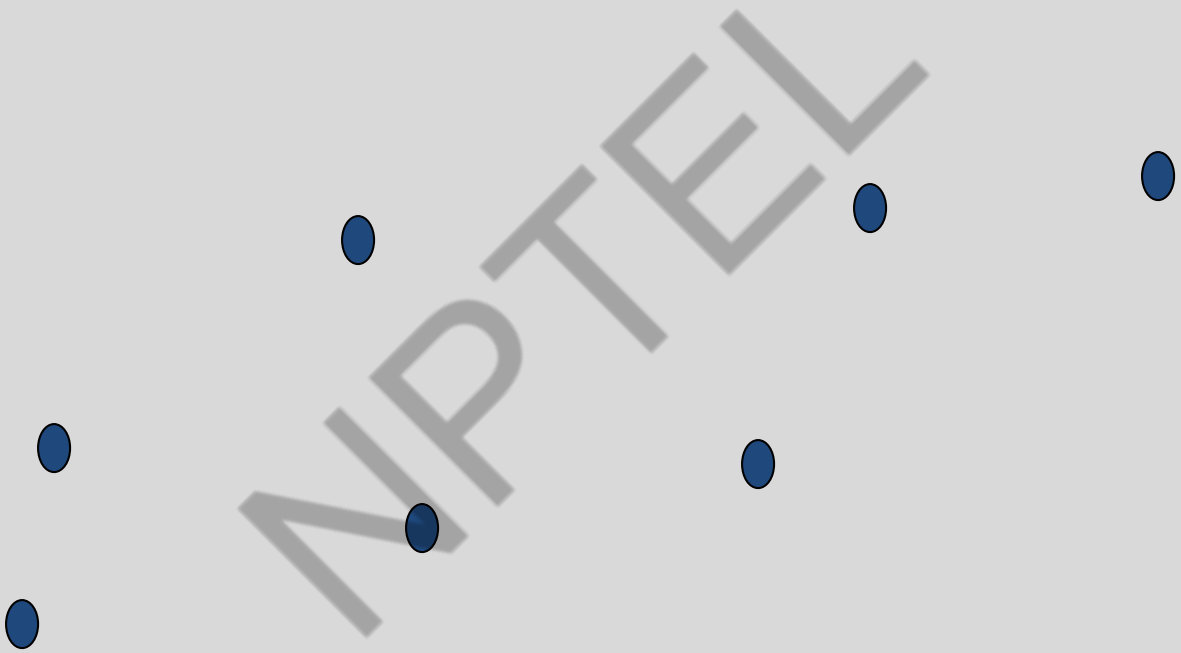
- Find a projection matrix w from N-dimensional to M-dimensional vectors that keeps error low
- Assume that N features are linear combination of $\mathrm{M} < N$ vectors

$$z_i = w_{i1}x_{i1} + \cdots + w_{id}x_{iN}$$

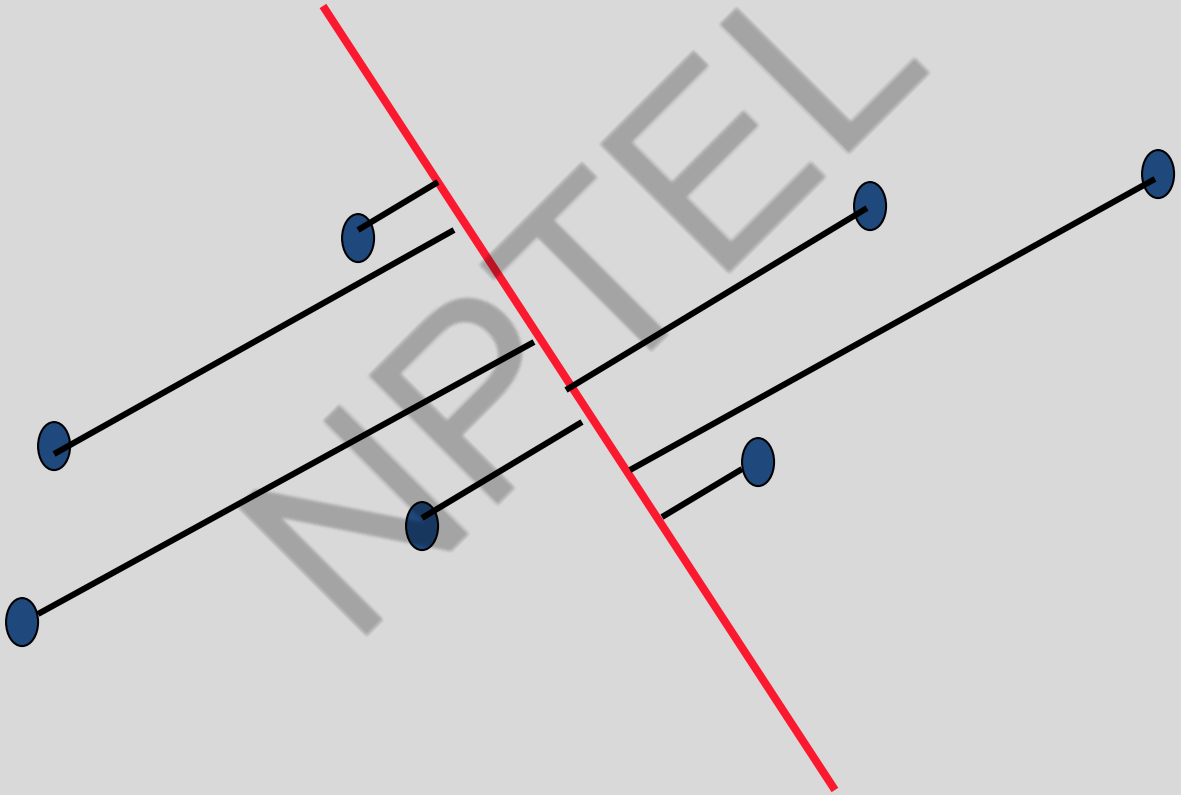
$$\boldsymbol{z} = w^T \boldsymbol{x}$$

- What we expect from such basis
  - Uncorrelated, cannot be reduced further
  - Have large variance or otherwise bear no information

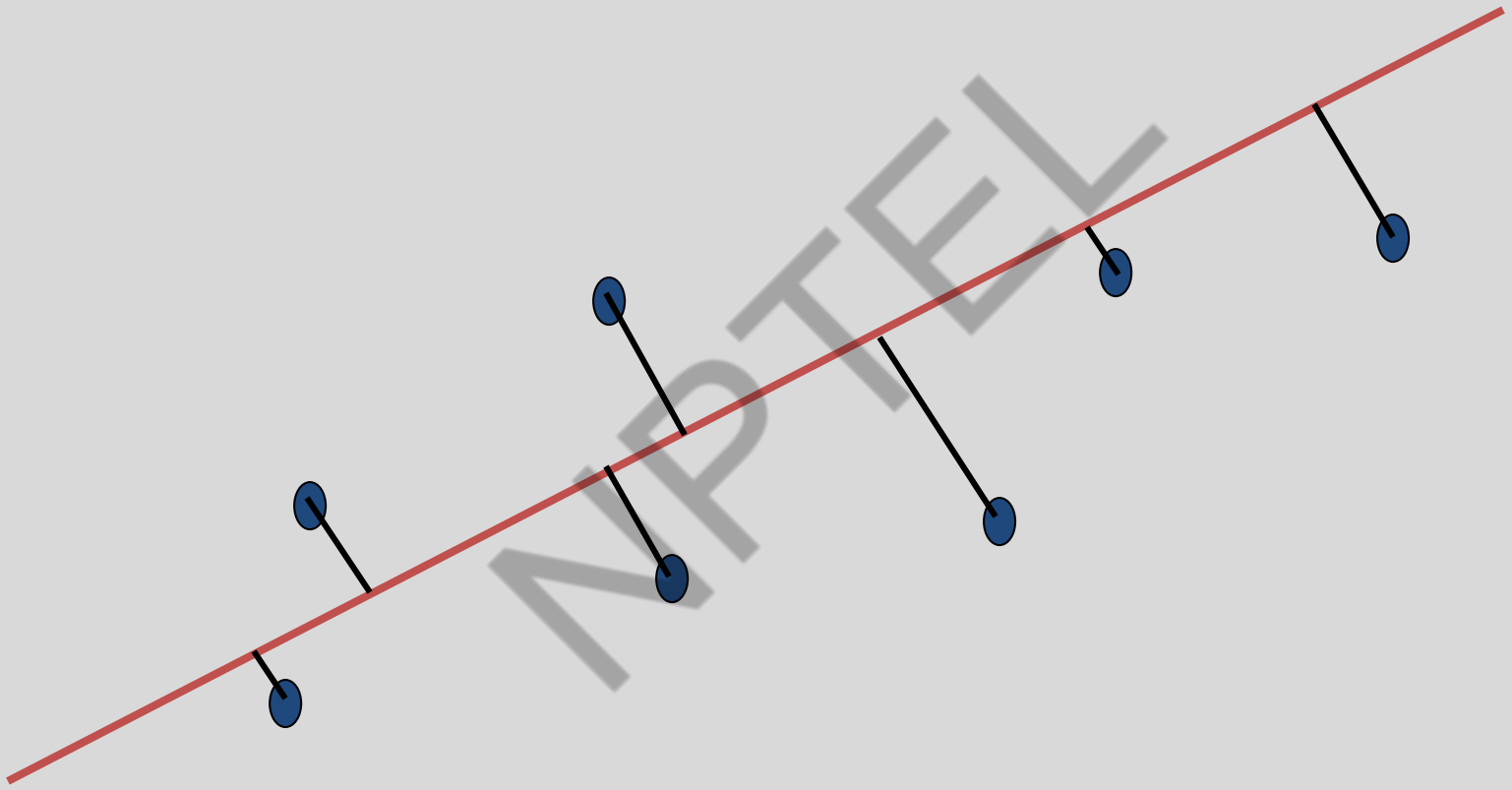
# Geometric picture of principal components (PCs)



# Geometric picture of principal components (PCs)



# Geometric picture of principal components (PCs)



# Algebraic definition of PCs

Given a sample of *p* observations on a vector of *N* variables

$$\{x_1, x_2, \cdots, x_p\} \in \Re^N$$

define the first principal component of the sample by the linear transformation

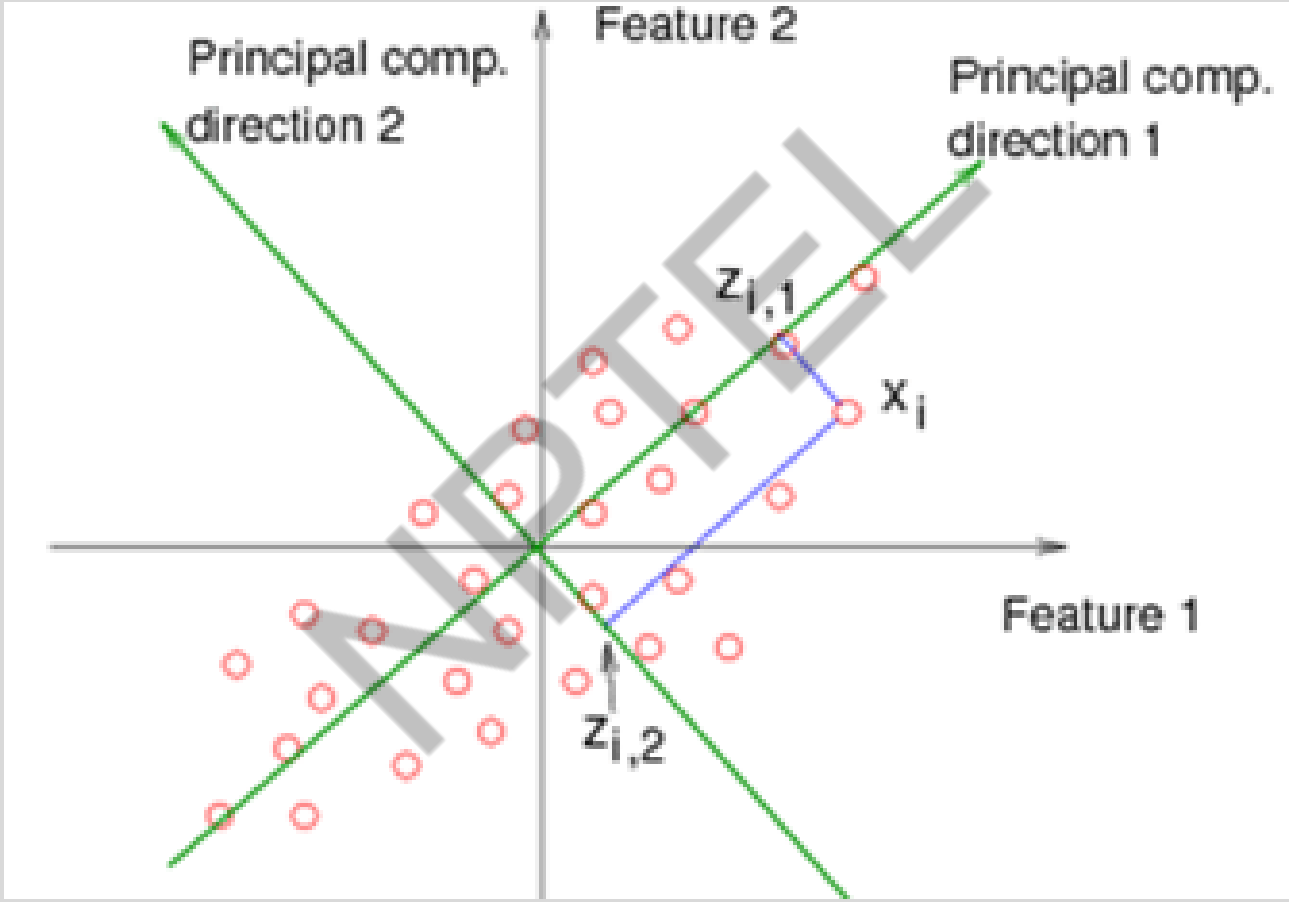
$$z_1 = w_1^T x_j = \sum_{i=1}^{N} w_{i1} x_{ij}, \quad j = 1,2,\cdots, p.$$

where the vector $w_1 = (w_{11}, w_{21}, \cdots, w_{N1})$

$$x_j = (x_{1j}, x_{2j}, \cdots, x_{Nj})$$

is chosen such that $\mathrm{var}[z_1]$ is maximum.

# PCA


Feature 2
Principal comp.
direction 2
Principal comp.
direction 1
$z_{i,1}$
$x_i$
Feature 1
$z_{i,2}$

# PCA

- Choose directions such that a total variance of data will be maximum
  1. Maximize Total Variance
- Choose directions that are orthogonal
  2. Minimize correlation

- Choose $M < N$ orthogonal directions which maximize total variance

# PCA

- $N$-dimensional feature space
- $\mathrm{N} \times N$ symmetric covariance matrix estimated from samples $Cov(\boldsymbol{x}) = \Sigma$
- Select $M$ largest eigenvalue of the covariance matrix and associated $M$ eigenvectors
- The first eigenvector will be a direction with largest variance

# PCA for image compression

p=1

p=2

p=4

p=8

p=16

p=32

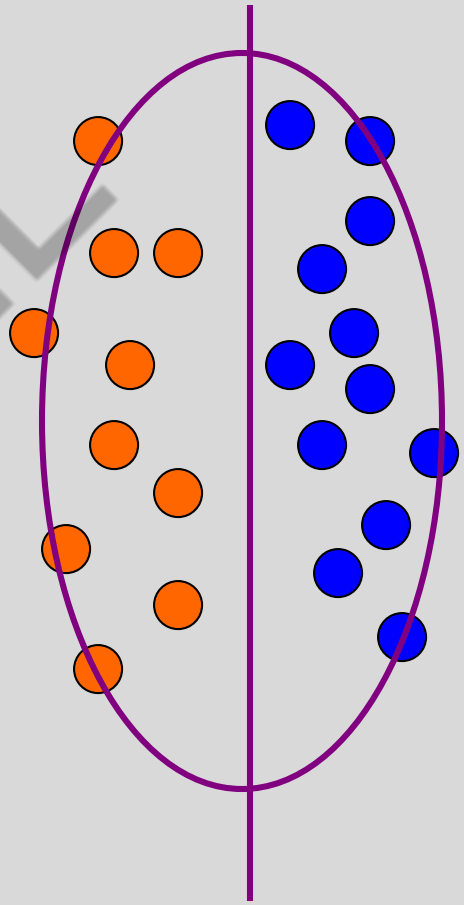
p=64

p=100

Original Image

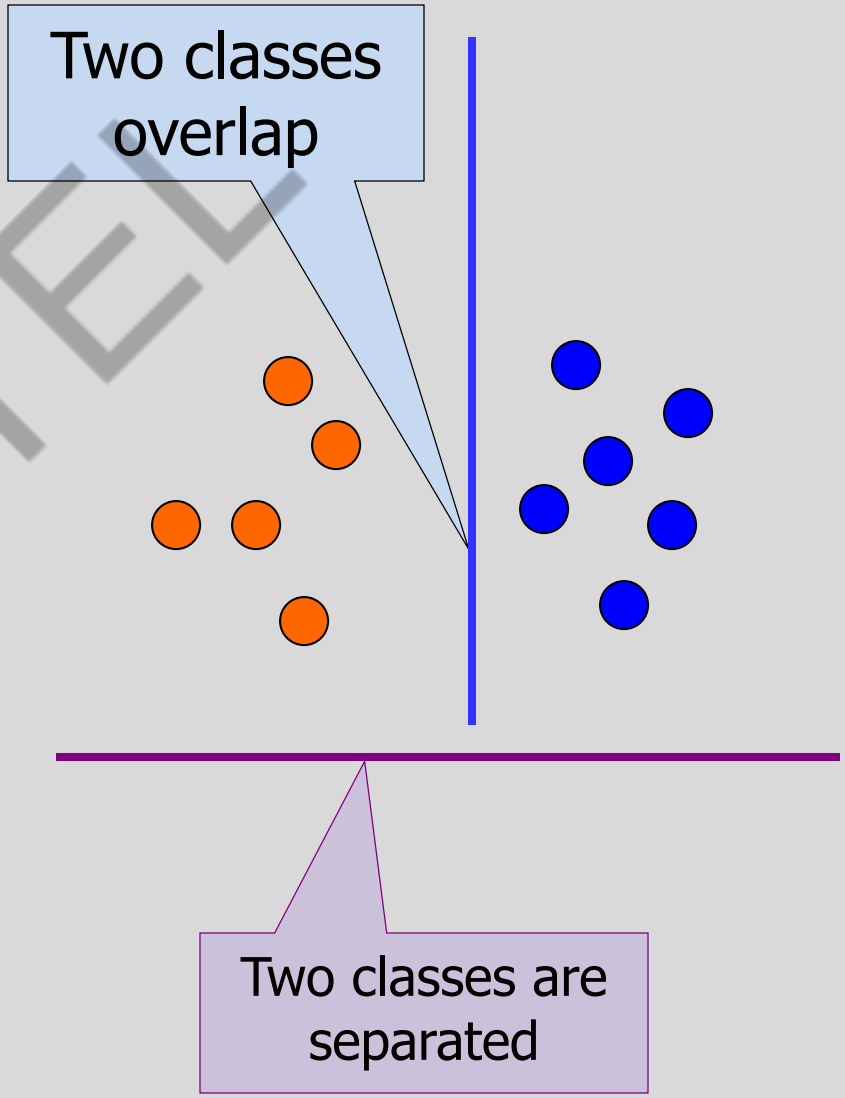
# Is PCA a good criterion for classification?

- Data variation determines the projection direction
- What's missing?
  - Class information



# What is a good projection?

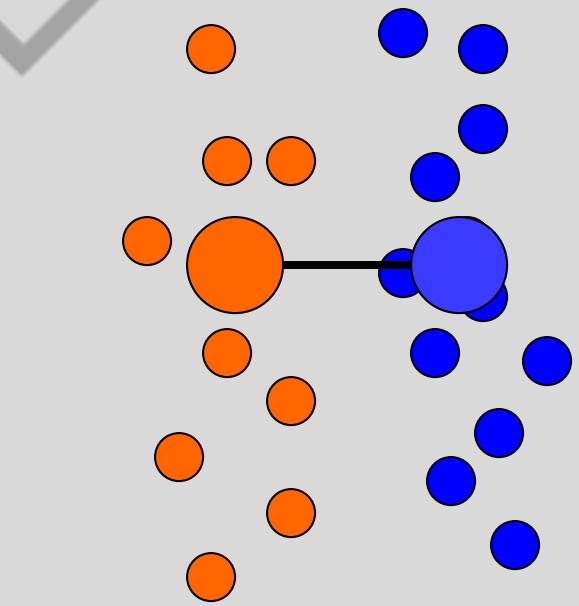
- Similarly, what is a good criterion?
  - Separating different classes



# What class information may be useful?

- Between-class distance
  - Distance between the centroids of different classes



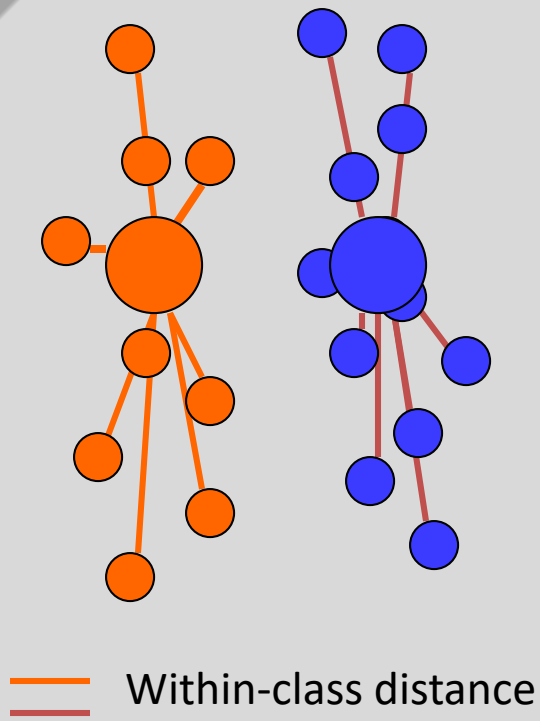


Between-class distance

# What class information may be useful?

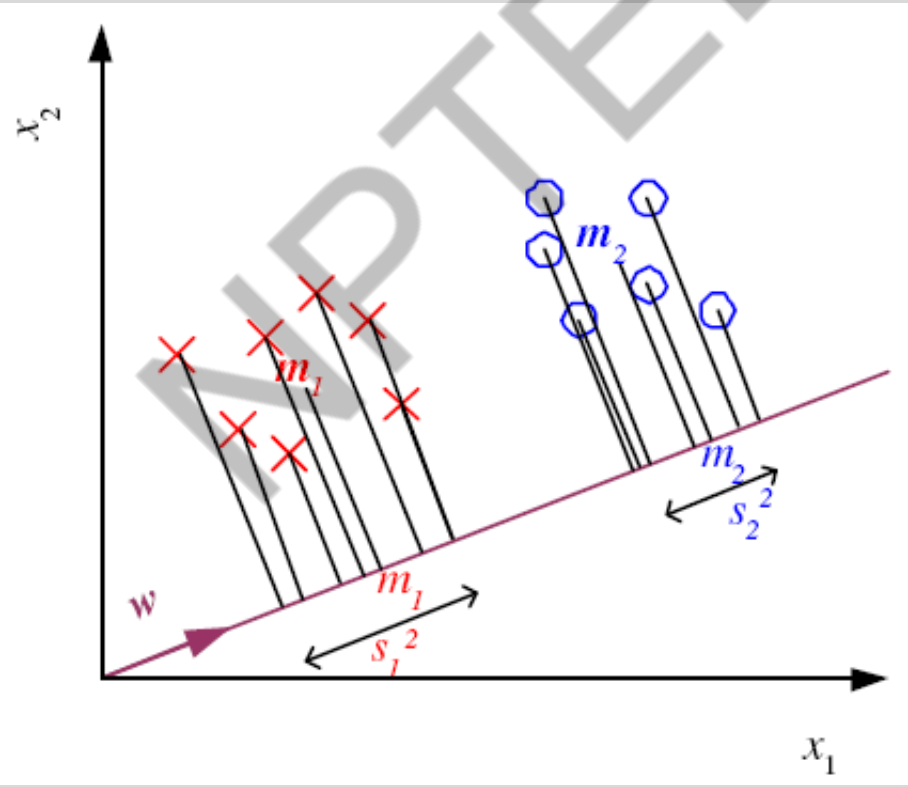
- Between-class distance
  - Distance between the centroids of different classes
- Within-class distance
  - Accumulated distance of an instance to the centroid of its class

- Linear discriminant analysis (LDA) finds most discriminant projection by
  - maximizing between-class distance
  - and minimizing within-class distance



# Linear Discriminant Analysis

- Find a low-dimensional space such that when $\boldsymbol{x}$ is projected, classes are well-separated



# Means and Scatter after projection

$$m_1 = \frac{\sum_t \mathbf{w}^T \mathbf{x}^t r^t}{\sum_t r^t} = \mathbf{w}^T \mathbf{m}_1$$

$$m_2 = \frac{\sum_t \mathbf{w}^T \mathbf{x}^t (1 - r^t)}{\sum_t (1 - r^t)} = \mathbf{w}^T \mathbf{m}_2$$

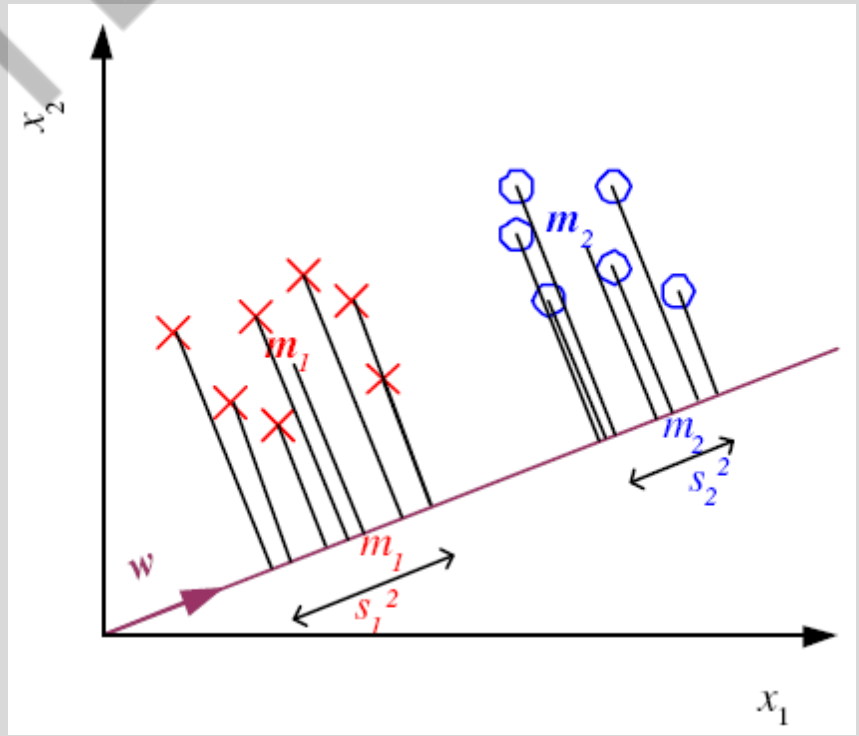
$$s_1^2 = \sum_t (\mathbf{w}^T \mathbf{x}^t - m_1)^2 r^t$$

$$s_2^2 = \sum_t (\mathbf{w}^T \mathbf{x}^t - m_2)^2 (1 - r^t)$$

# Good Projection

- Means are as far away as possible
- Scatter is small as possible
- Fisher Linear Discriminant

$$J(\mathbf{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2}$$



Thank You

# Foundations of Machine Learning

## Module 3: Instance based Learning and Feature Reduction

## Part D: Collaborative Filtering

Sudeshna Sarkar

IIT Kharagpur

# Recommender Systems

- Item Prediction: predict a ranked list of items that a user is likely to buy or use. predict the rating score that a user is likely to give to an item that s/he has not seen or used before. E.g.,
  - rating on an unseen movie. In this case, the utility of item *s* to user *u* is the rating given to *s* by *u*.

- Rating Prediction: Predict whether someone will like a movie, book, webpage, etc.

# The Recommendation Problem

- We have a set of users $U$ and a set of items $S$ to be recommended to the users.
- Let $p$ be an utility function that measures the usefulness of item $s$ ($\in S$) to user $u$ ($\in U$), i.e.,
  - $p: U \times S \rightarrow R$, where $R$ is a totally ordered set (e.g., non-negative integers or real numbers in a range)
- Objective
  - Learn $p$ based on the past data
  - Use $p$ to predict the utility value of each item $s$ ($\in S$) to each user $u$ ($\in U$)

# Recommender Systems

- Content based :
  - recommend items similar to the ones the user preferred in the past
- Collaborative filtering:
  - Look at what similar users liked
  - Similar users = Similar likes and dislikes

# Collaborative Filtering

- Present each user with a vector of ratings
- Two types:
  - Yes / No
  - Explicit Ratings
- Predict Rating by User-based Nearest Neighbour

# Collaborative Filtering for Rating Prediction

- User-based Nearest Neighbour
  - Neighbour = similar users
  - Generate a prediction for an item i by analyzing ratings for i from users in u's neighbourhood

# Neighborhood formation phase

- Let the record (or profile) of the target user be **u** (represented as a vector), and the record of another user be **v** (**v** $\in$ *T*).
- The similarity between the target user, **u**, and a neighbor, **v**, can be calculated using the **Pearson's correlation coefficient**:

$$sim(\mathbf{u},\mathbf{v}) = \frac{\sum_{i \in C}(r_{\mathbf{u},i} - \bar{r}_{\mathbf{u}})(r_{\mathbf{v},i} - \bar{r}_{\mathbf{v}})}{\sqrt{\sum_{i \in C}(r_{\mathbf{u},i} - \bar{r}_{\mathbf{u}})^2}\sqrt{\sum_{i \in C}(r_{\mathbf{v},i} - \bar{r}_{\mathbf{v}})^2}},$$

# Recommendation Phase

- Use the following formula to compute the rating prediction of item *i* for target user **u**
-

$$p(\mathbf{u}, i) = \bar{r}_{\mathbf{u}} + \frac{\sum_{\mathbf{v} \in V} sim(\mathbf{u}, \mathbf{v}) \times (r_{\mathbf{v},i} - \bar{r}_{\mathbf{v}})}{\sum_{\mathbf{v} \in V} |sim(\mathbf{u}, \mathbf{v})|}$$

where *V* is the set of *k* similar users, $r_{\mathbf{v},i}$ is the rating of user **v** given to item *i*,

# Issue with the user-based *k*NN CF

- The problem with the user-based formulation of collaborative filtering is the lack of scalability:
  - it requires the real-time comparison of the target user to all user records in order to generate predictions.
- A variation of this approach that remedies this problem is called **item-based CF**.

# Item-based CF

- The item-based approach works by comparing items based on their pattern of ratings across users. The similarity of items *i* and *j* is computed as follows:

$$sim(i,j) = \frac{\sum_{\mathbf{u} \in U} (r_{\mathbf{u},i} - \bar{r}_{\mathbf{u}})(r_{\mathbf{u},j} - \bar{r}_{\mathbf{u}})}{\sqrt{\sum_{\mathbf{u} \in U} (r_{\mathbf{u},i} - \bar{r}_{\mathbf{u}})^2} \sqrt{\sum_{\mathbf{u} \in U} (r_{\mathbf{u},j} - \bar{r}_{\mathbf{u}})^2}}$$

# Recommendation phase

- After computing the similarity between items we select a set of *k* most similar items to the target item and generate a predicted value of user **u**'s rating

$$p(\mathbf{u}, i) = \frac{\sum_{j \in J} r_{\mathbf{u},j} \times sim(i, j)}{\sum_{j \in J} sim(i, j)}$$

where *J* is the set of *k* similar items

Thank You