

# What is KNN Algorithm?

---

KNN - K Nearest Neighbors, is one of the simplest **Supervised** Machine Learning algorithm mostly used for

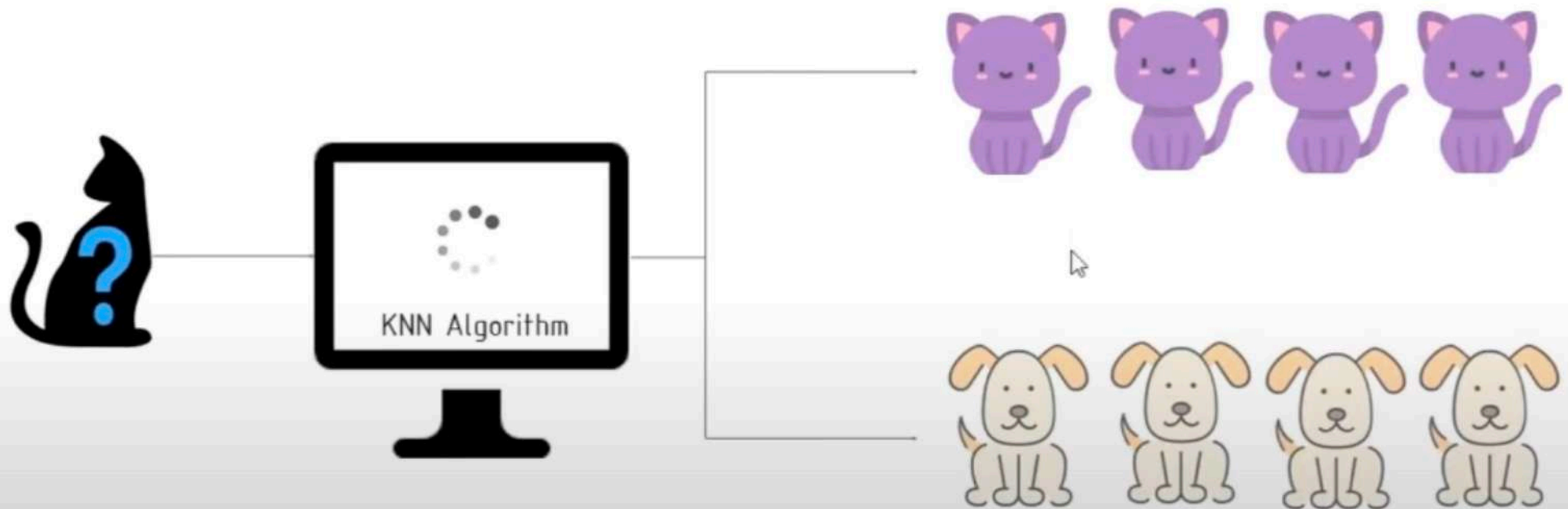
## Classification



It classifies a data point based on how its neighbors are classified

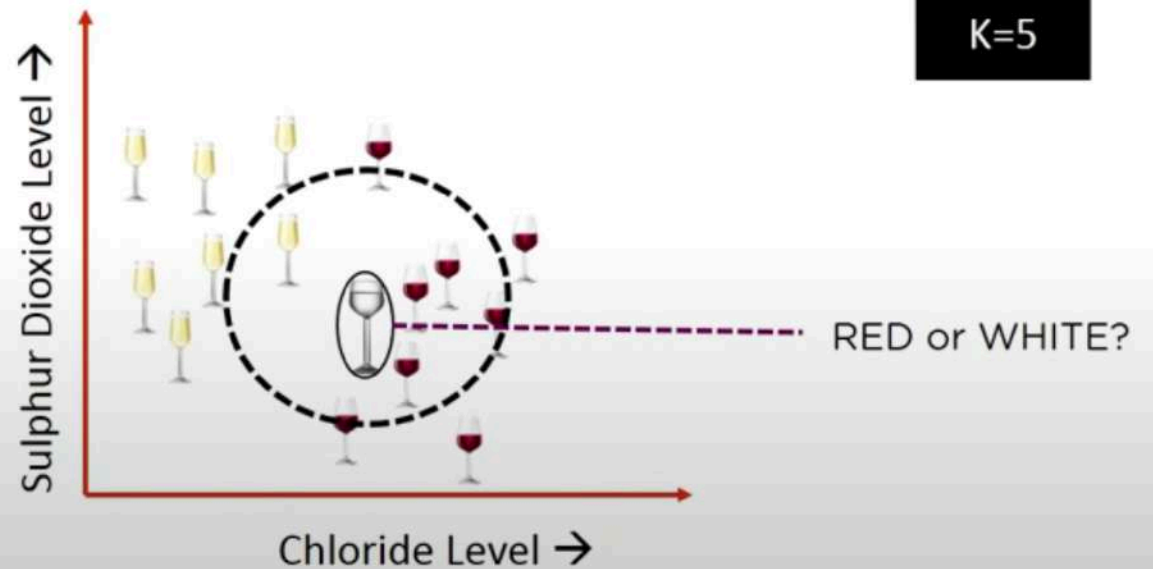
# What Is KNN Algorithm?

*K Nearest Neighbour is a Supervised Learning algorithm that classifies a new data point into the target class, depending on the features of it's neighbouring data points.*



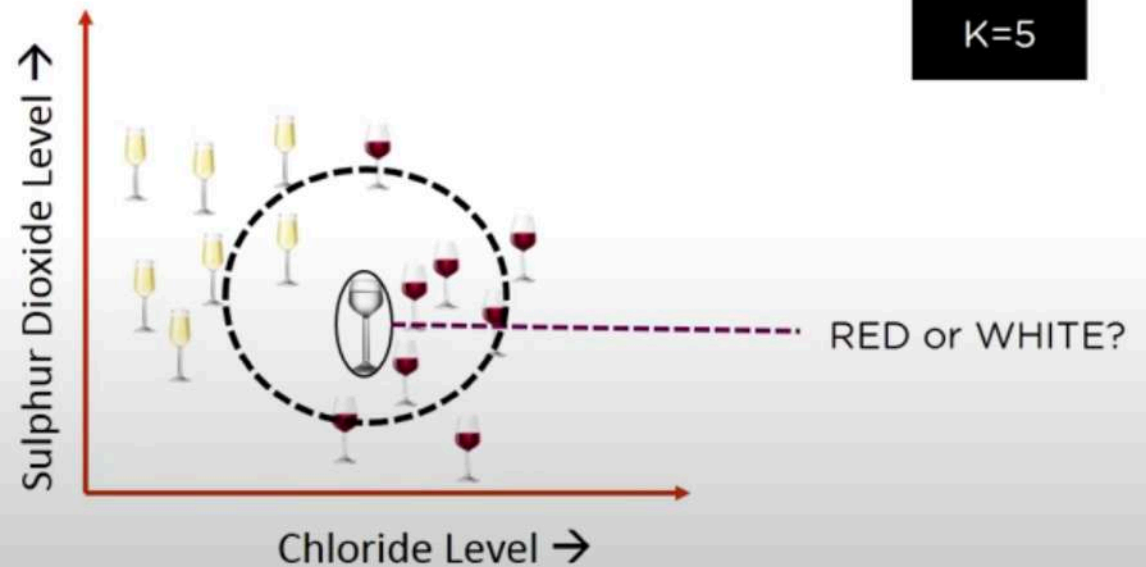
# What is KNN Algorithm?

$k$  in KNN is a parameter that refers to the number of nearest neighbors to include in the majority voting process



# What is KNN Algorithm?

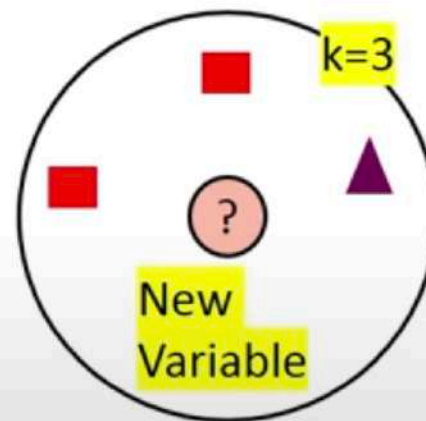
A data point is classified by majority votes from its 5 nearest neighbors



# How do we choose the factor 'k'?

---

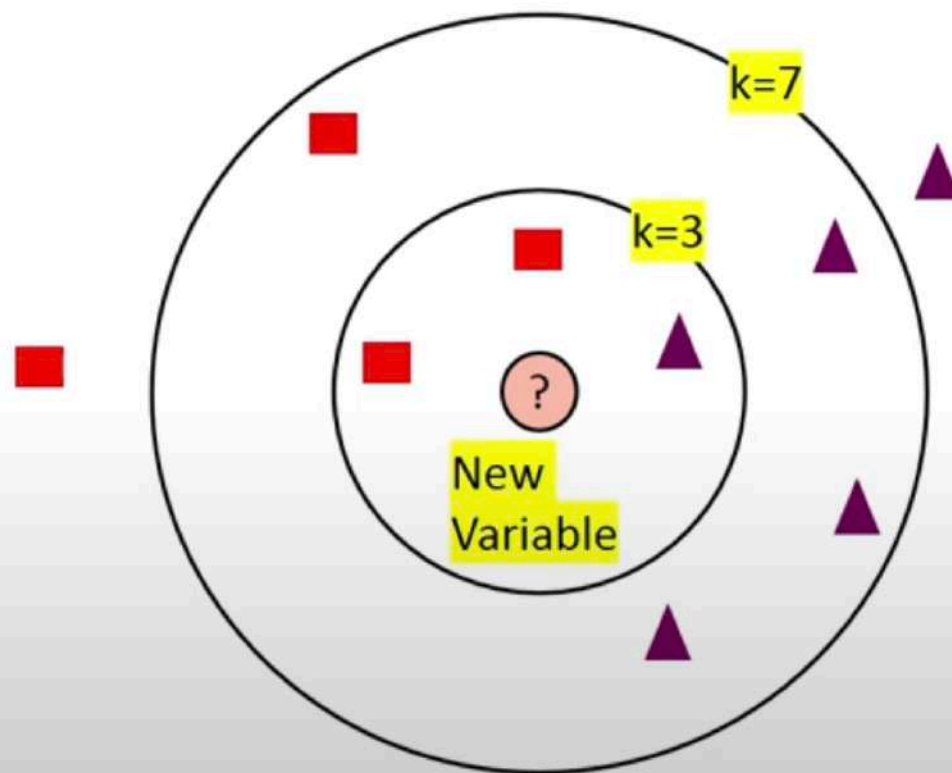
KNN Algorithm is based on feature similarity: Choosing the right value of  $k$  is a process called parameter tuning, and is important for better accuracy



So at  $k=3$ , we can classify '?' as ■

# How do we choose the factor 'k'?

KNN Algorithm is based on feature similarity: Choosing the right value of  $k$  is a process called parameter tuning, and is important for better accuracy



But at  $k=7$ , we classify '?' as



# How do we choose the factor 'k'?

KNN Algorithm is based on feature similarity. Choosing the right value of  $k$  is a process called parameter tuning, and it is a bit tricky.



The class of unknown data point was ■ at  $k=3$  but changed at  $k=7$ , so which  $k$  should we choose?

?

New Variable



## How do we choose the factor 'k'?

---

To choose a value of k:

$\sqrt{n}$ , where n is the total number of data points

Odd value of K is selected to avoid confusion between two classes of data





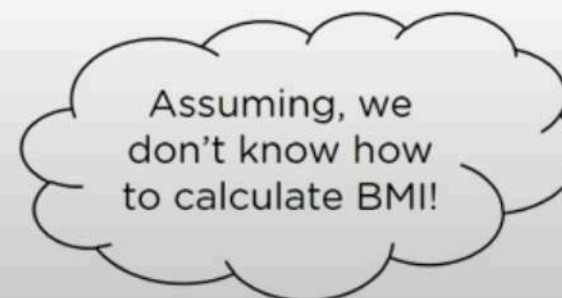
Consider a dataset having two variables: height (cm) & weight (kg) and each point is classified as Normal or Underweight

Weight(x2)	Height(y2)	Class
51	167	Underweight
62	182	Normal
69	176	Normal
64	173	Normal
65	172	Normal
56	174	Underweight
58	169	Normal
57	173	Normal
55	170	Normal

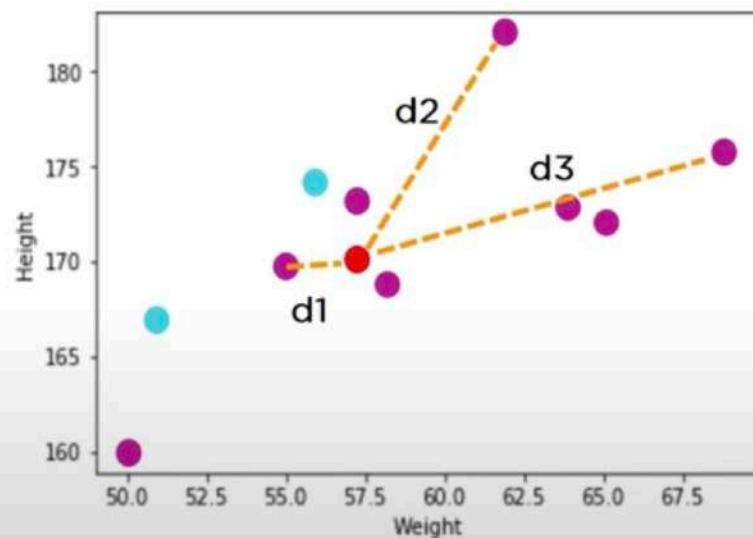


On the basis of the given data we have to classify the below set as Normal or Underweight using KNN

57 kg	170 cm	?
-------	--------	---



Let's calculate it to understand clearly:



● Unknown data point

$$\text{dist}(d1) = \sqrt{(170-167)^2 + (57-51)^2} \approx 6.7$$

$$\text{dist}(d2) = \sqrt{(170-182)^2 + (57-62)^2} \approx 13$$

$$\text{dist}(d3) = \sqrt{(170-176)^2 + (57-69)^2} \approx 13.4$$

Similarly, we will calculate Euclidean distance of unknown data point from all the points in the dataset

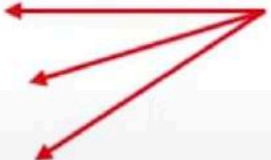
Hence, we have calculated the Euclidean distance of unknown data point from all the points as shown:

Where  $(x_1, y_1) = (57, 170)$  whose class we have to classify

Weight(x2)	Height(y2)	Class	Euclidean Distance
51	167	Underweight	6.7
62	182	Normal	13
69	176	Normal	13.4
64	173	Normal	7.6
65	172	Normal	8.2
56	174	Underweight	4.1
58	169	Normal	1.4
57	173	Normal	3
55	170	Normal	2

Now, let's calculate the nearest neighbor at  $k=3$

Weight(x2)	Height(y2)	Class	Euclidean Distance
51	167	Underweight	6.7
62	182	Normal	13
69	176	Normal	13.4
64	173	Normal	7.6
65	172	Normal	8.2
56	174	Underweight	4.1
58	169	Normal	1.4
57	173	Normal	3
55	170	Normal	2



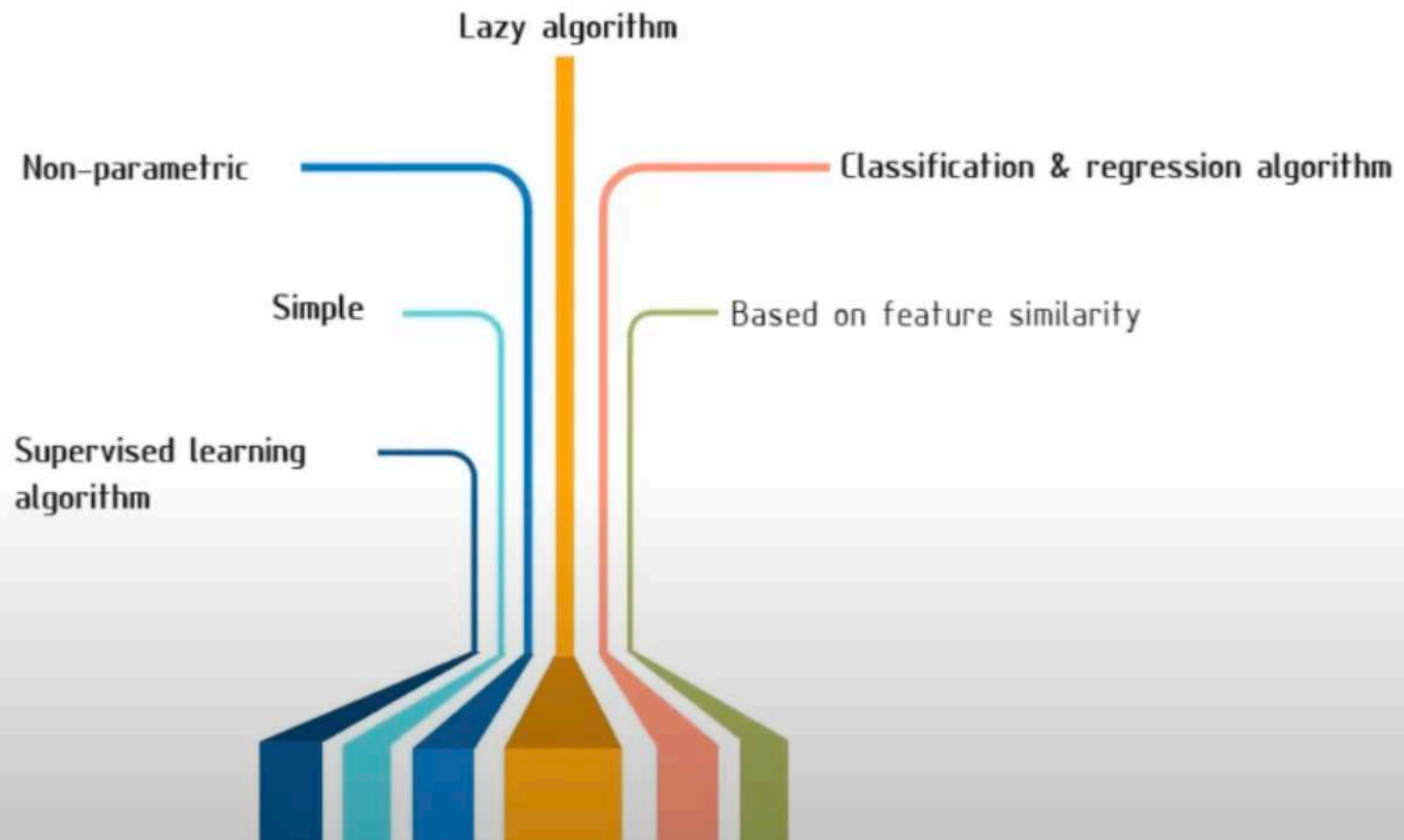
$k = 3$

57 kg

170 cm

?

# Features Of KNN



# Basic k-nearest neighbor classification

- Training method:
  - Save the training examples
- At prediction time:
  - Find the  $k$  training examples  $(x_1, y_1), \dots, (x_k, y_k)$  that are closest to the test example  $x$
  - **Classification**: Predict the most frequent class among those  $y_i$ 's.
  - **Regression**: Predict the average of among the  $y_i$ 's.
- Improvements:
  - *Weighting* examples from the neighborhood
  - Measuring "*closeness*"
  - Finding "close" examples in a large training set *quickly*

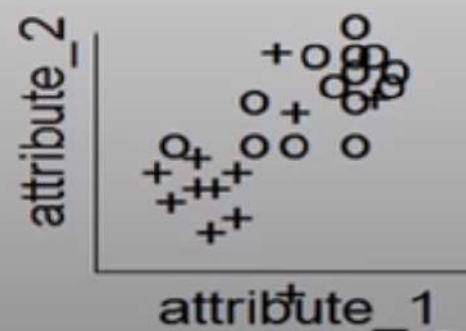


## k-Nearest Neighbor

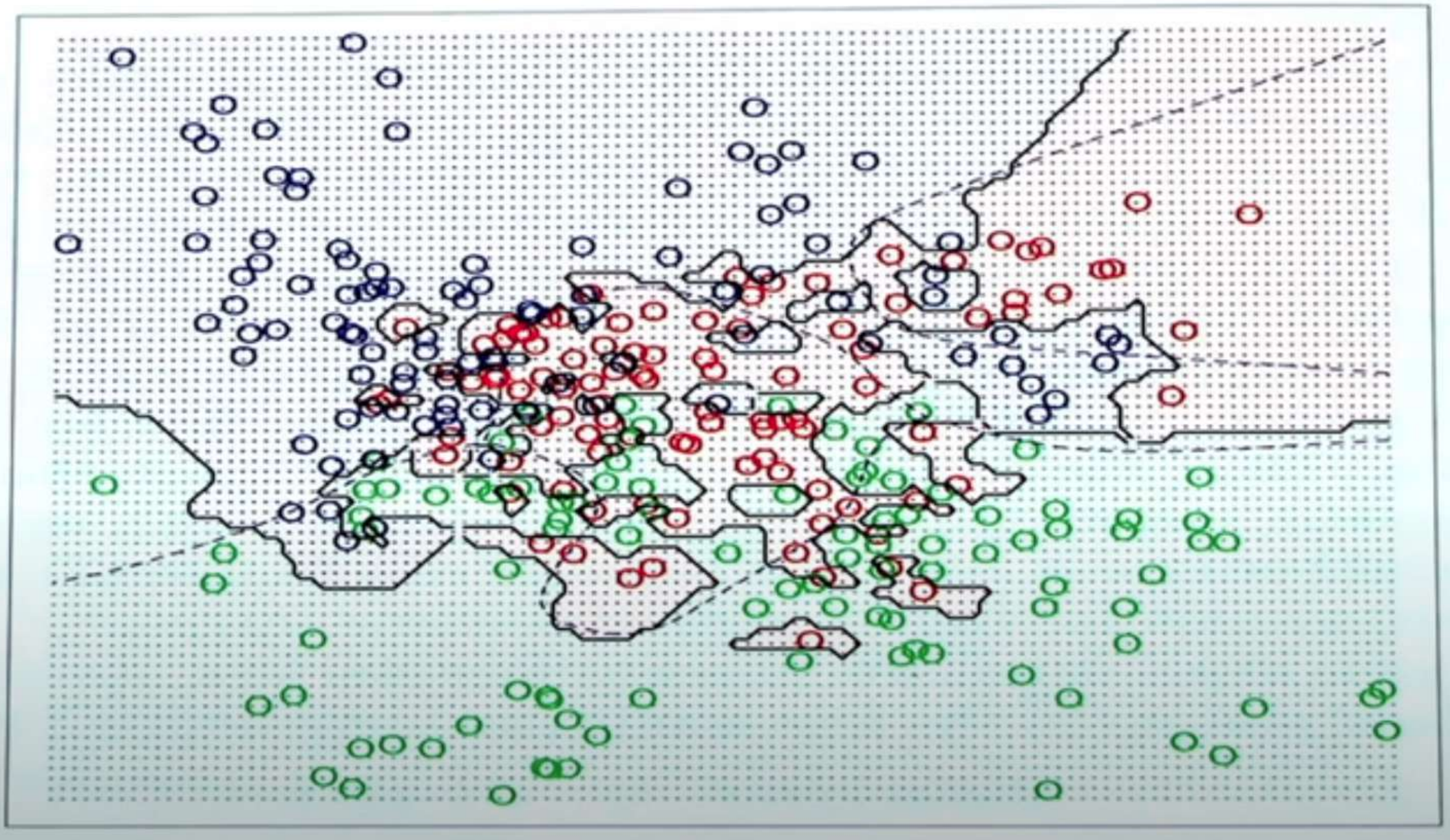
$$Dist(c_1, c_2) = \sqrt{\sum_{i=1}^N (attr_i(c_1) - attr_i(c_2))^2}$$

$prediction_{test} = ?$

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

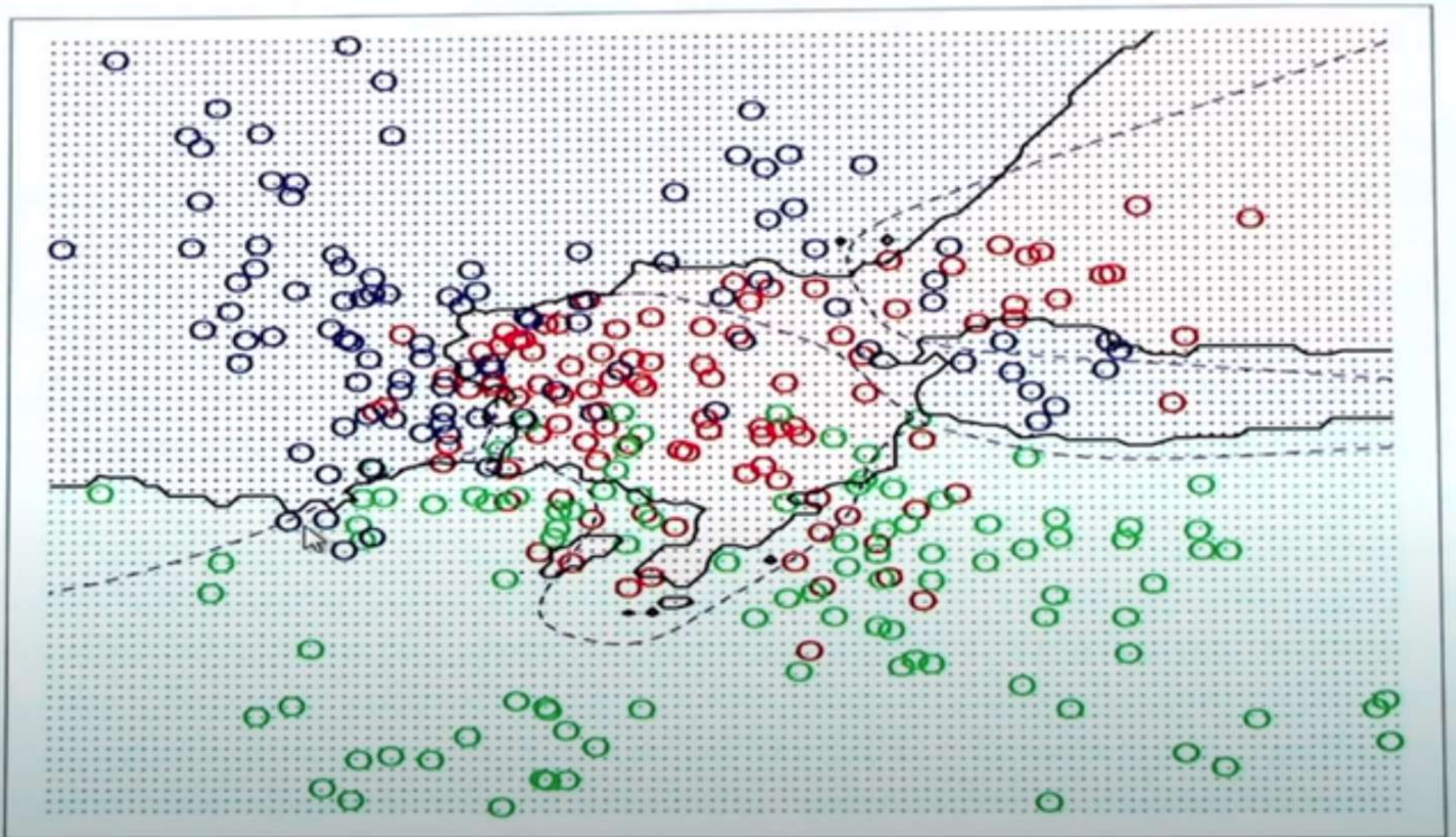


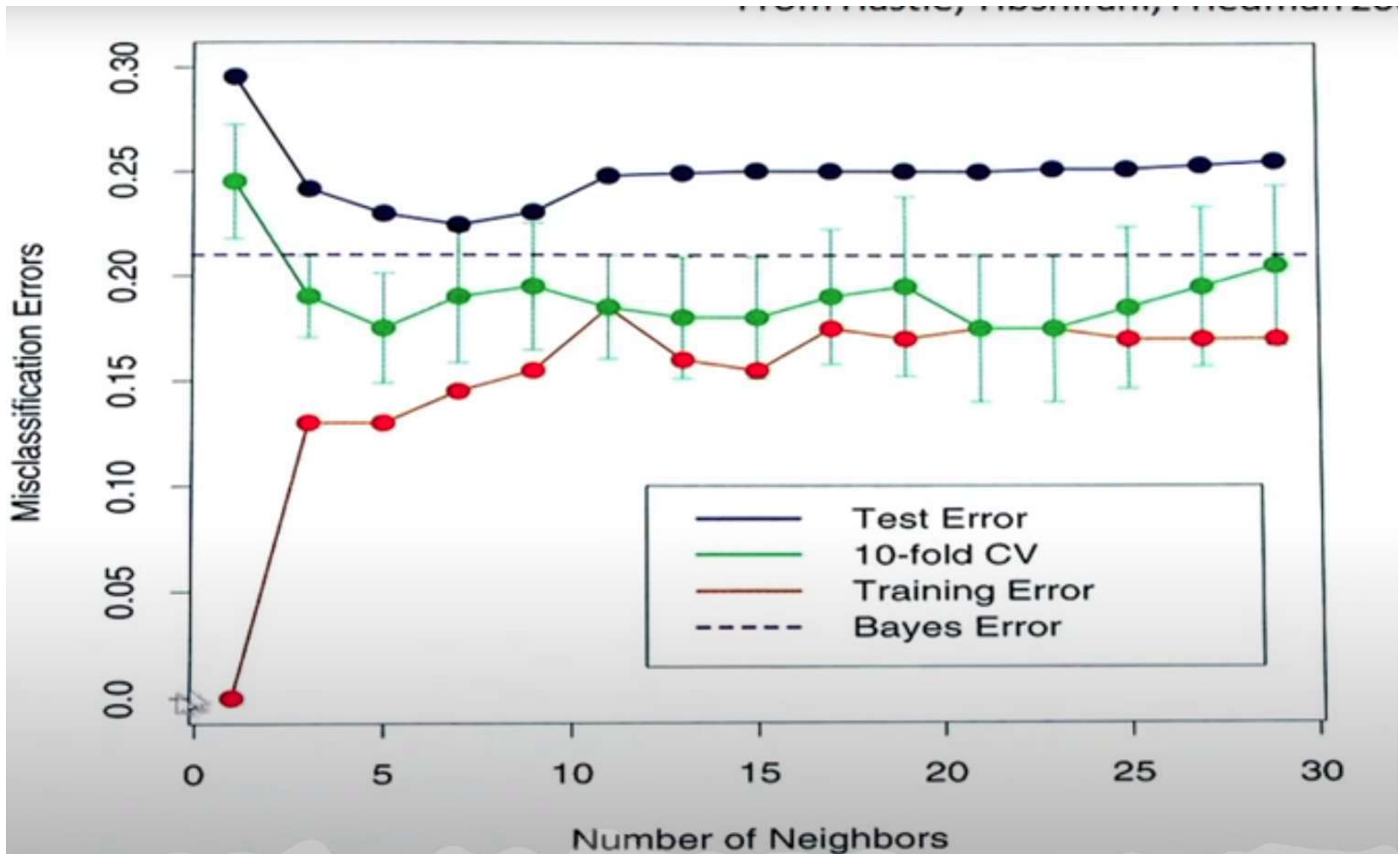
## 1-Nearest Neighbor





## 15-Nearest Neighbors







## Weighted Euclidean Distance

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

- large weights  $\Rightarrow$  attribute is more important
  - small weights  $\Rightarrow$  attribute is less important
  - zero weights  $\Rightarrow$  attribute doesn't matter
- 
- Weights allow kNN to be effective with axis-parallel elliptical classes
  - Where do weights come from?

## Distance-Weighted kNN

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

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

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

# Locally Weighted Averaging

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

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

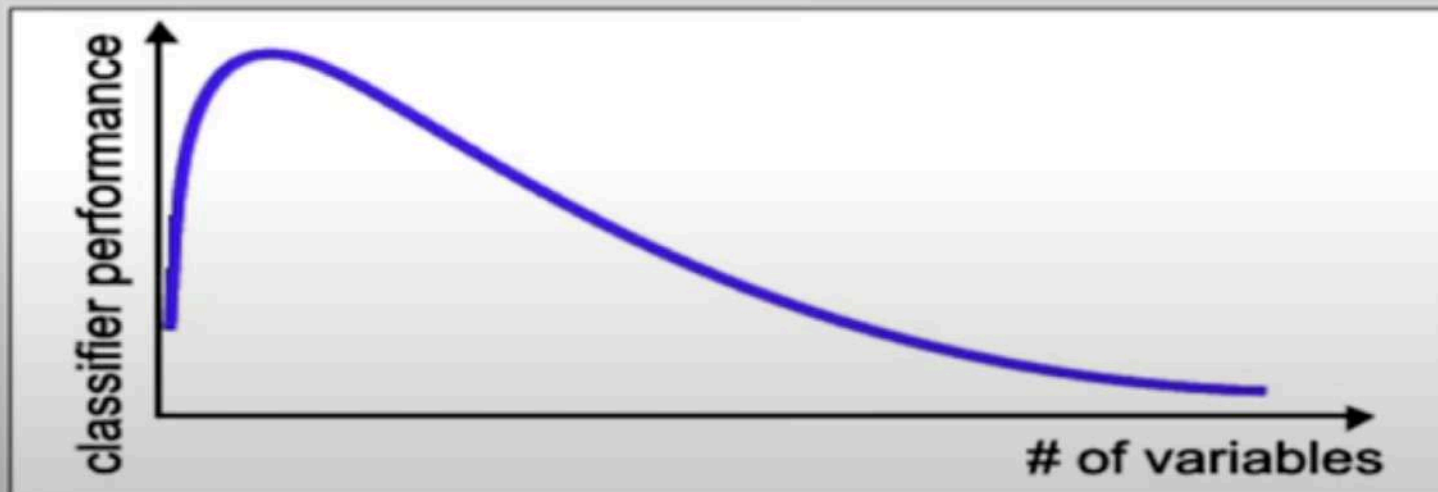
$$w_k = \frac{1}{e^{KernelWidth \cdot Dist(c_k, c_{test})}}$$

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



# Curse of Dimensionality

- number of training examples is fixed  
=> the classifier's performance usually will degrade for a large number of features!



# Feature Selection - Definition

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



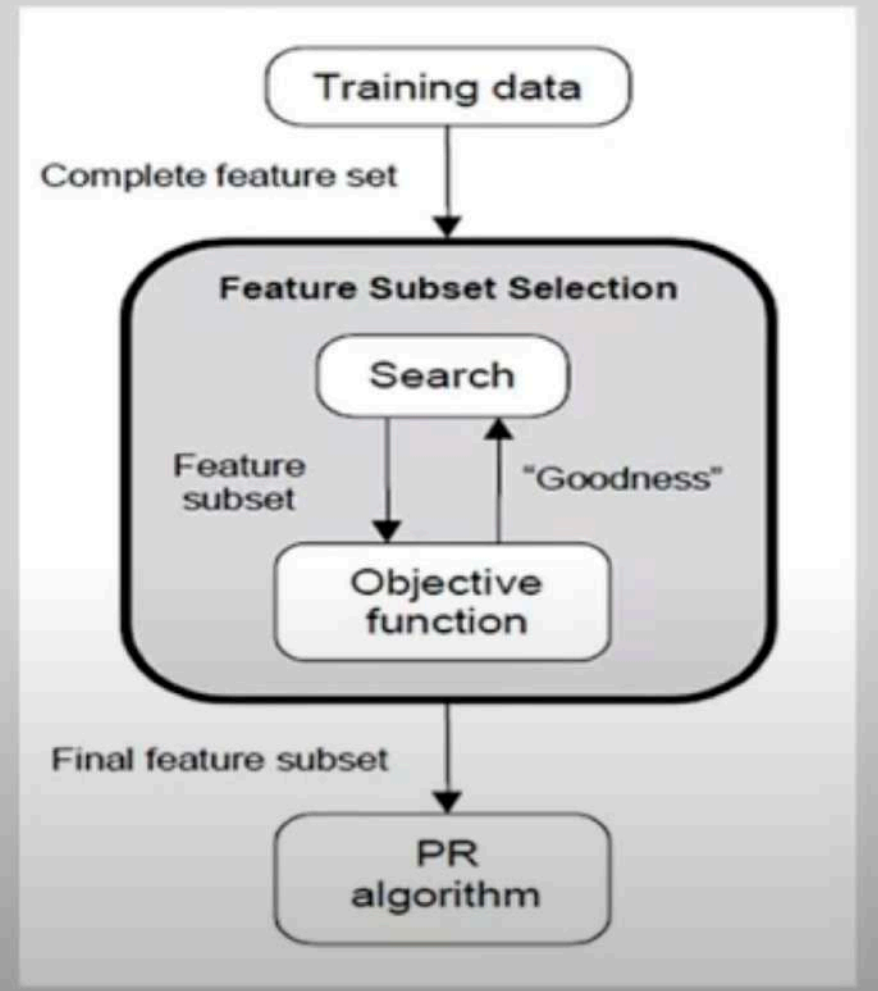
# Subset selection

- $d$  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.



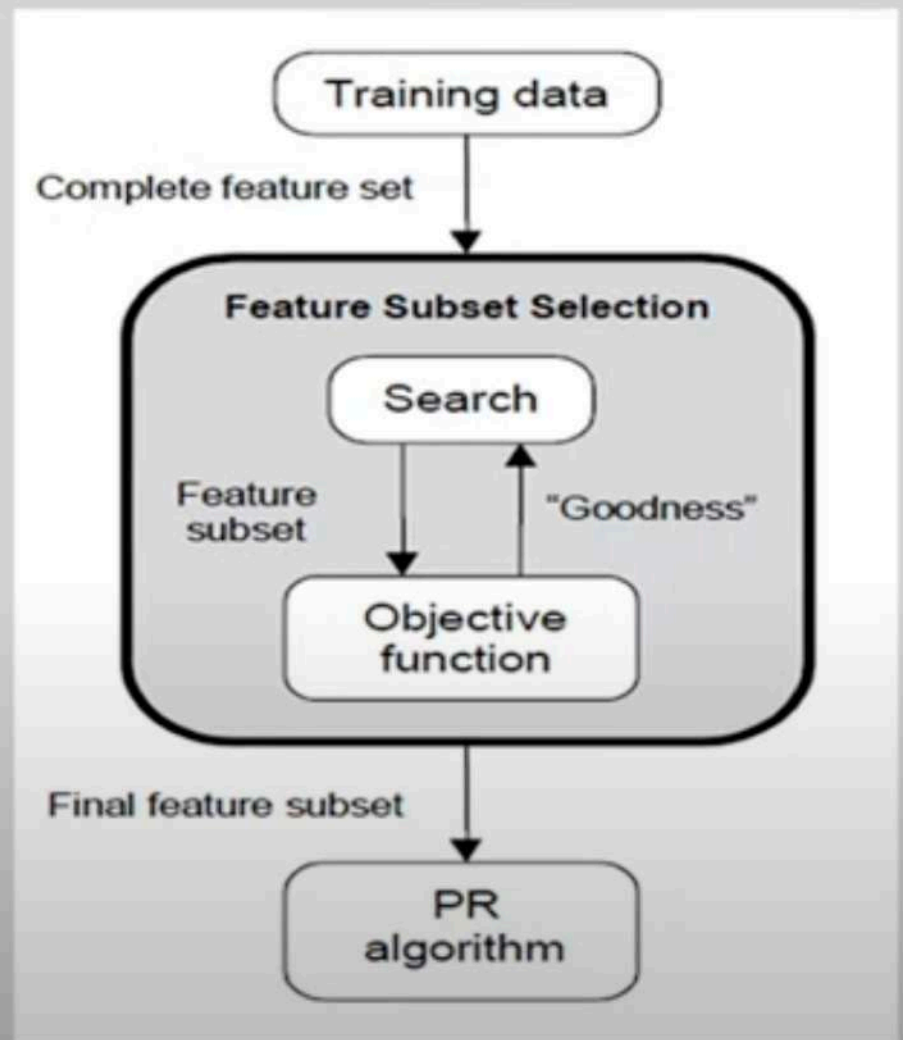
# Feature Selection Steps (cont'd)

## Search strategies

- Optimum
- Heuristic
- Randomized

## Evaluation strategies

- Filter methods
- Wrapper methods



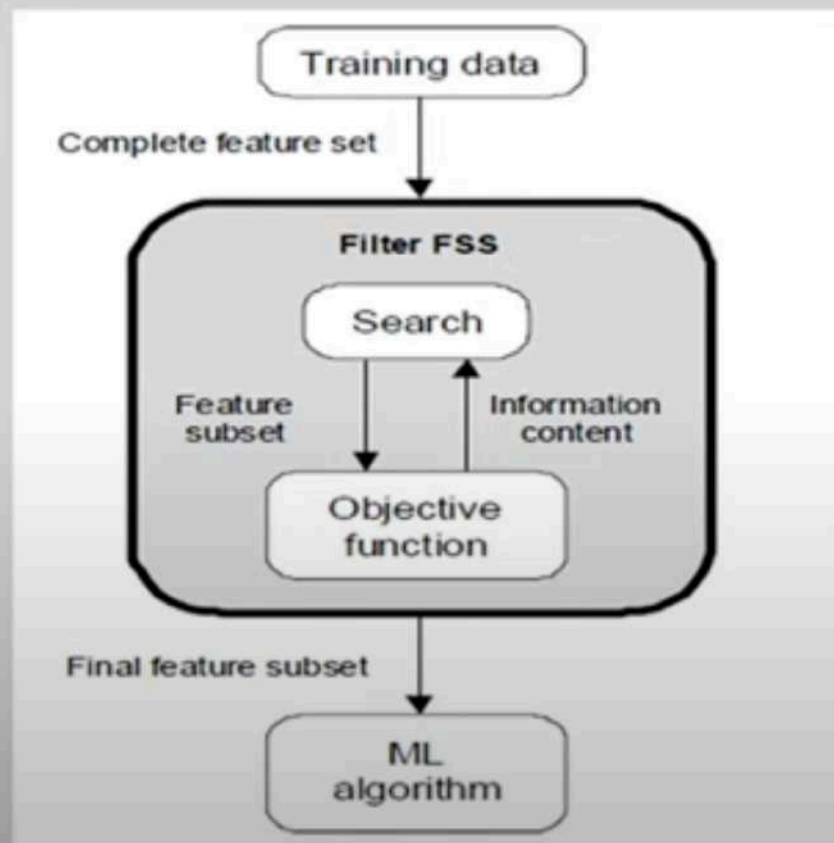
# Evaluating feature subset

- 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

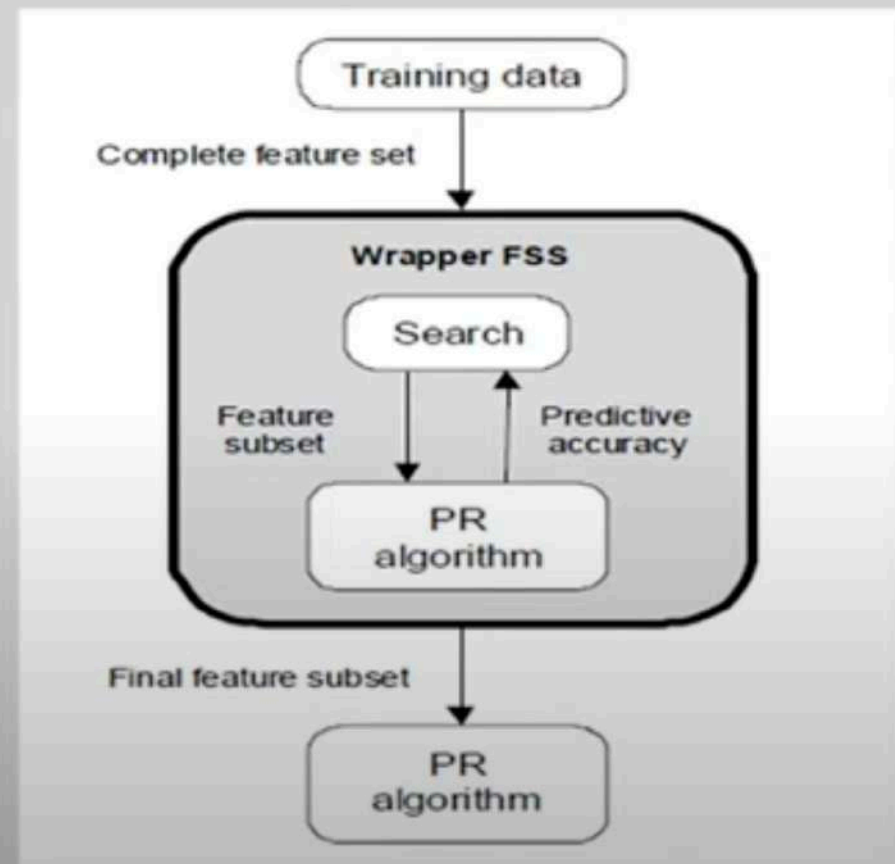


# Evaluation Strategies

## Filter Methods



## Wrapper Methods





# Feature selection

Univariate (looks at each feature independently of others)

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

# 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 - definition

- Given a set of features  $F = \{x_1, \dots, x_N\}$   
the **Feature Extraction("Construction") problem** is  
is to map  $F$  to some feature set  $F''$  that maximizes  
the learner's ability to classify patterns

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \xrightarrow{\text{feature extraction}} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix} = f \left( \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \right)$$

# Feature Extraction

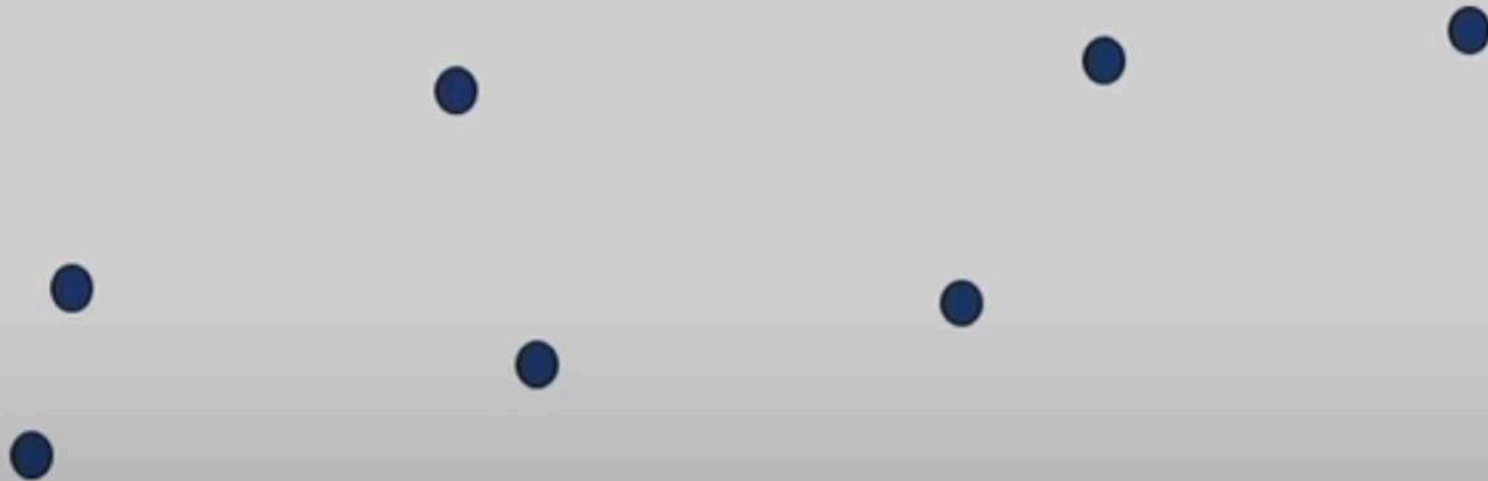
- Find a projection matrix  $w$  from  $N$ -dimensional to  $M$ -dimensional vectors that keeps error low
- Assume that  $N$  features are linear combination of  $M < N$  vectors

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

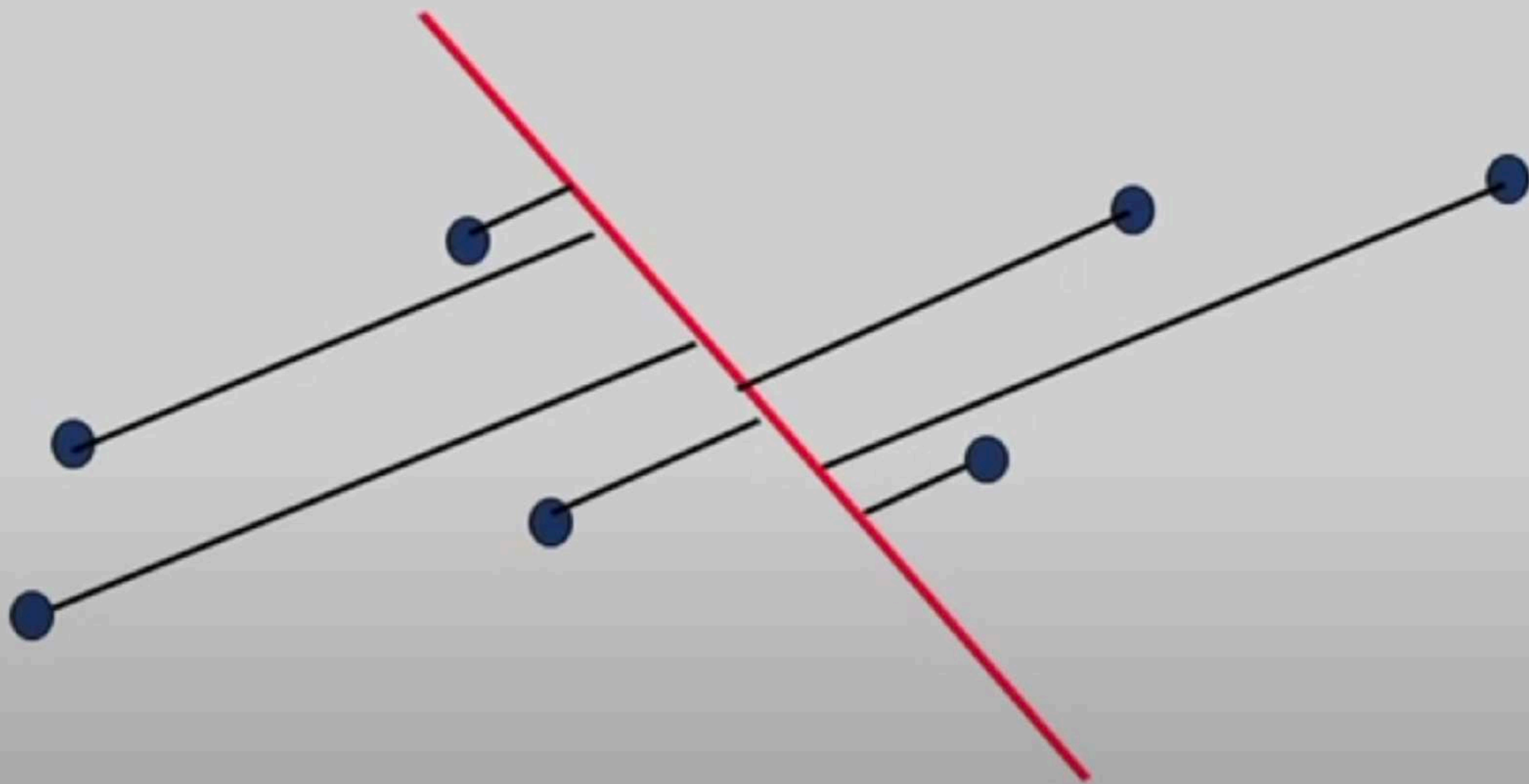
$$\mathbf{z} = \mathbf{w}^T \mathbf{x}$$

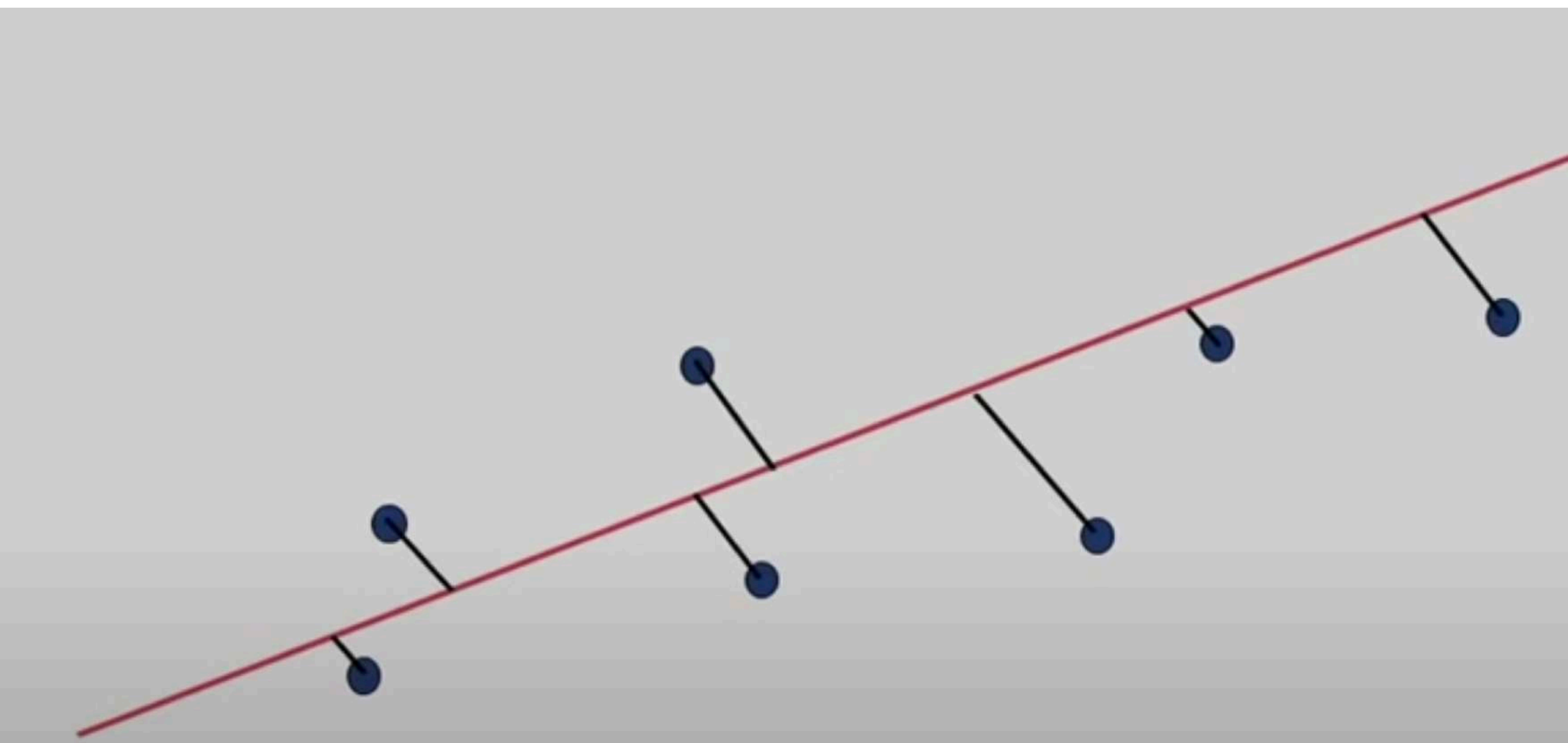
- What we expect from such basis
  - Uncorrelated, cannot be reduced further
  - Have large variance or otherwise bear no information

## Geometric picture of principal components (PCs)









# Algebraic definition of PCs

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

$$\{x_1, x_2, \dots, x_p\} \in \mathfrak{R}^N$$

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

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

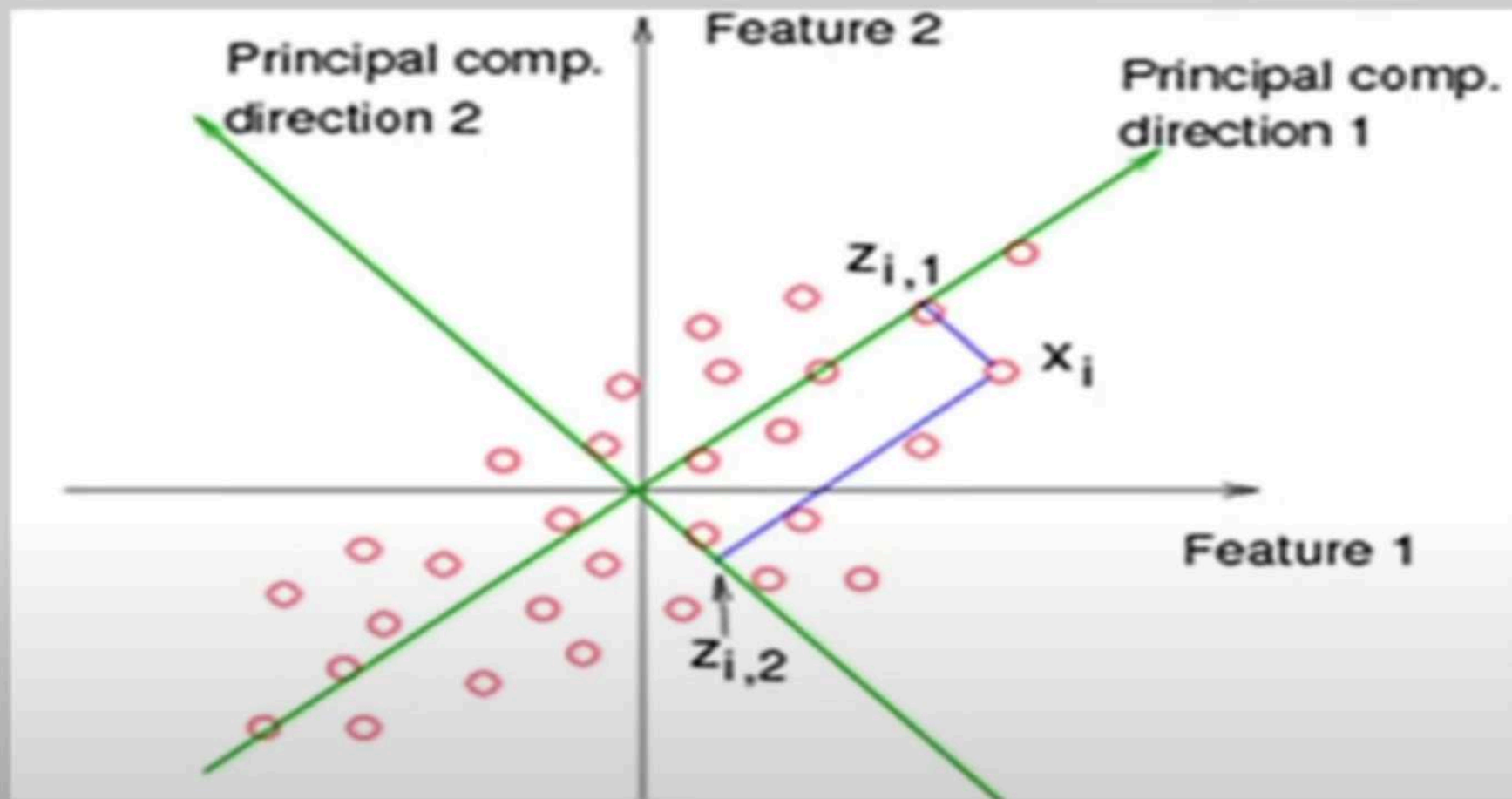
where the vector

$$w_1 = (w_{11}, w_{21}, \dots, w_{N1})$$

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

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

# PCA



# PCA

- $N$ -dimensional feature space
- $N \times N$  symmetric covariance matrix estimated from samples  $Cov(\mathbf{x}) = \Sigma$
- 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**



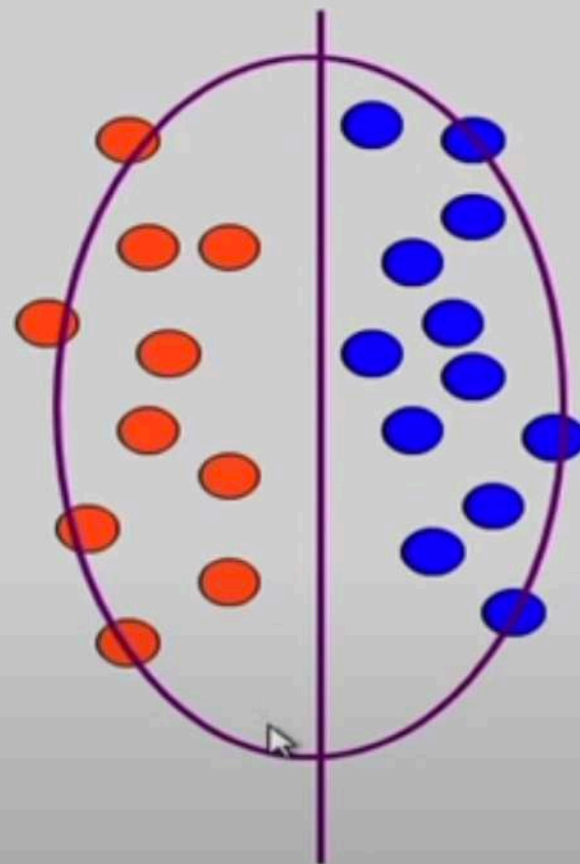
**p=100**



**Original  
Image**

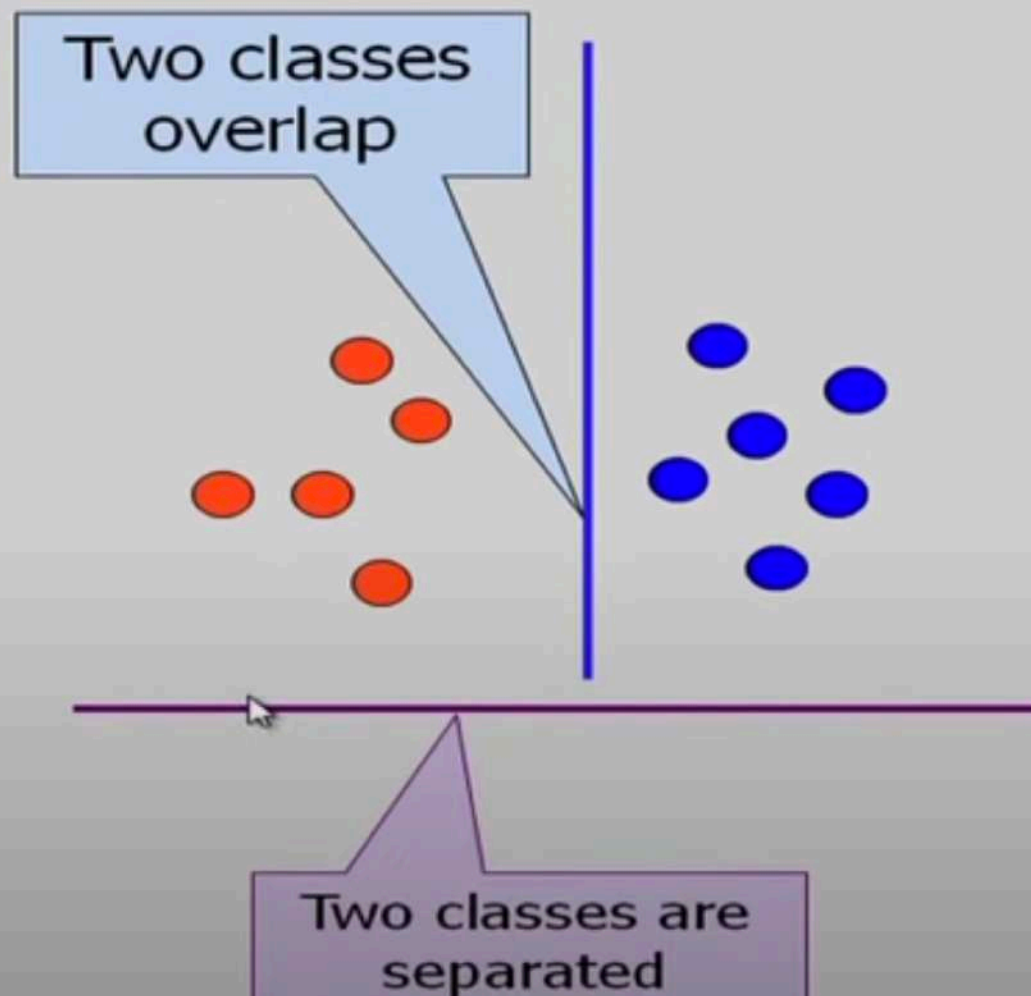
# Is PCA a good criterion for classification?

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



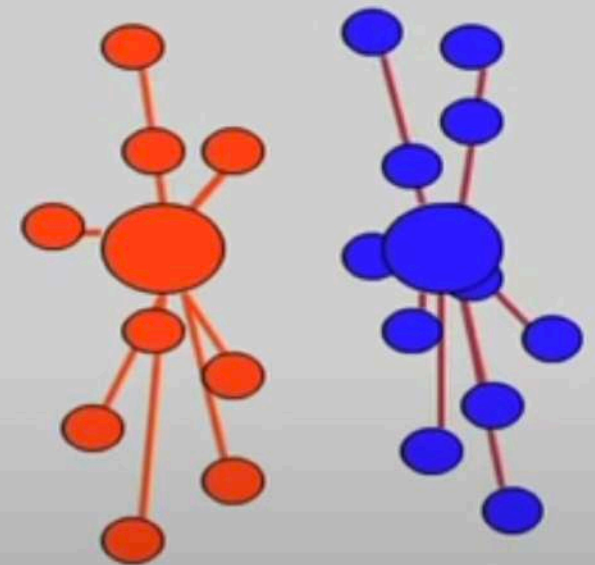
# What is a good projection?

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



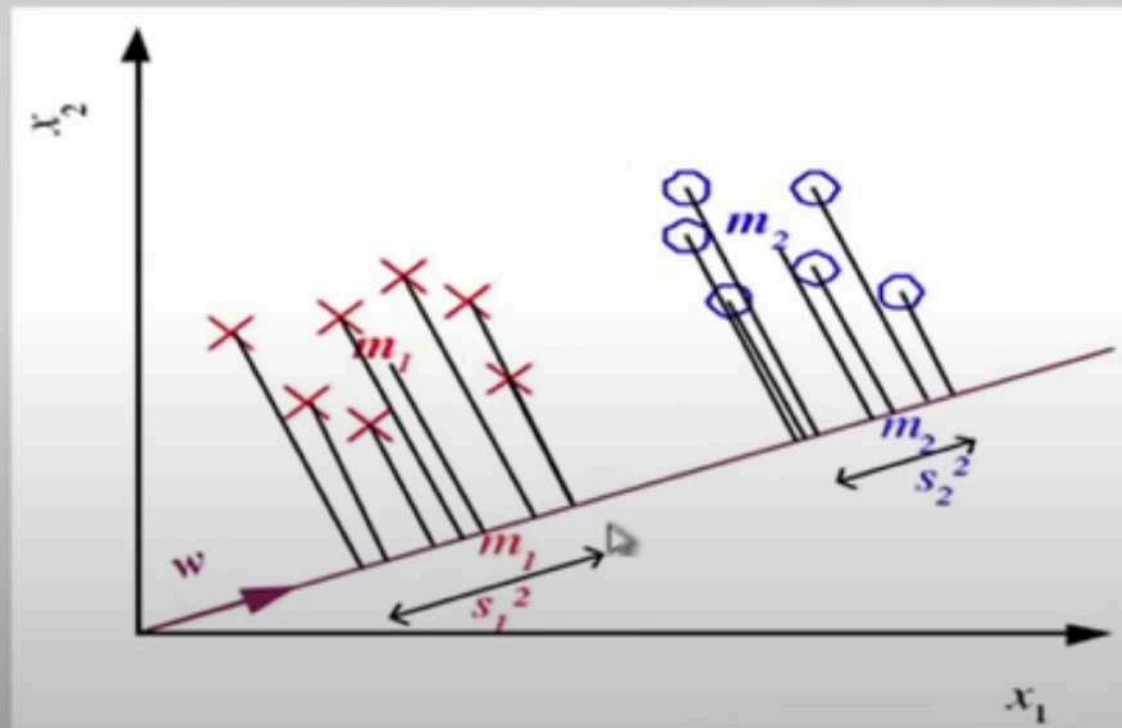
# What class information may be useful?

- Between-class distance
  - Distance between the centroids of different classes
- 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  $x$  is projected, classes are well-separated





## Means and Scatter after projection

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

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

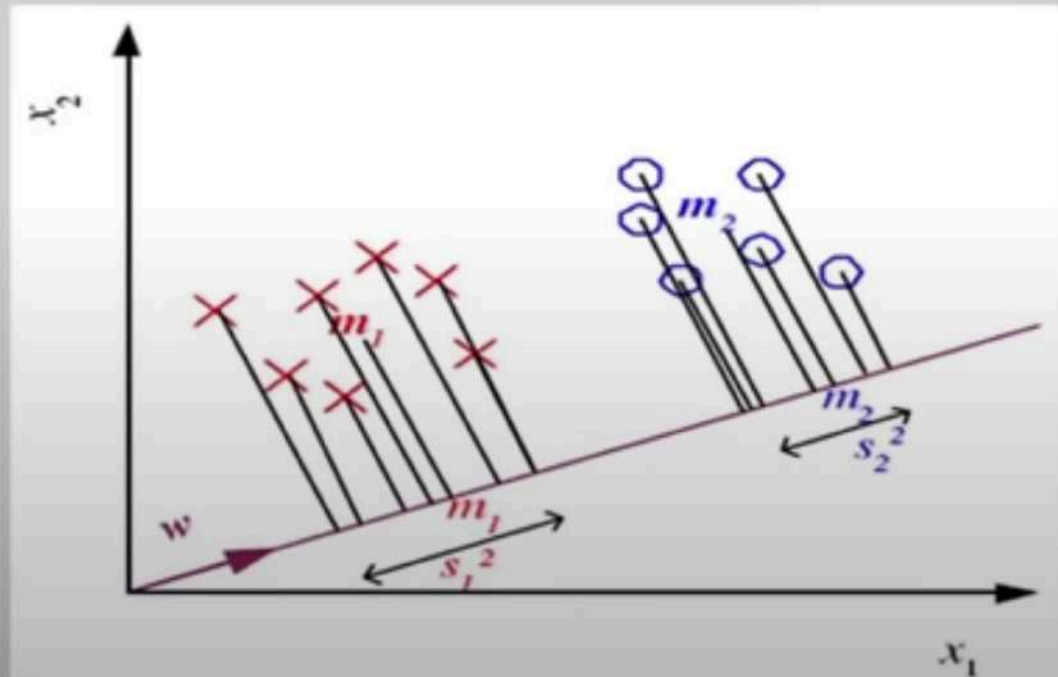
$$s_1^2 = \sum_t (\mathbf{w}^T \mathbf{x}^t - m_1)^2 r^t$$

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

# Good Projection

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

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



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

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

## Recommendation Phase

- Use the following formula to compute the rating prediction of item  $i$  for target user  $\mathbf{u}$

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

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

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

## 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  $\mathbf{u}$ 's rating

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

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