# Computer Vision

Classification

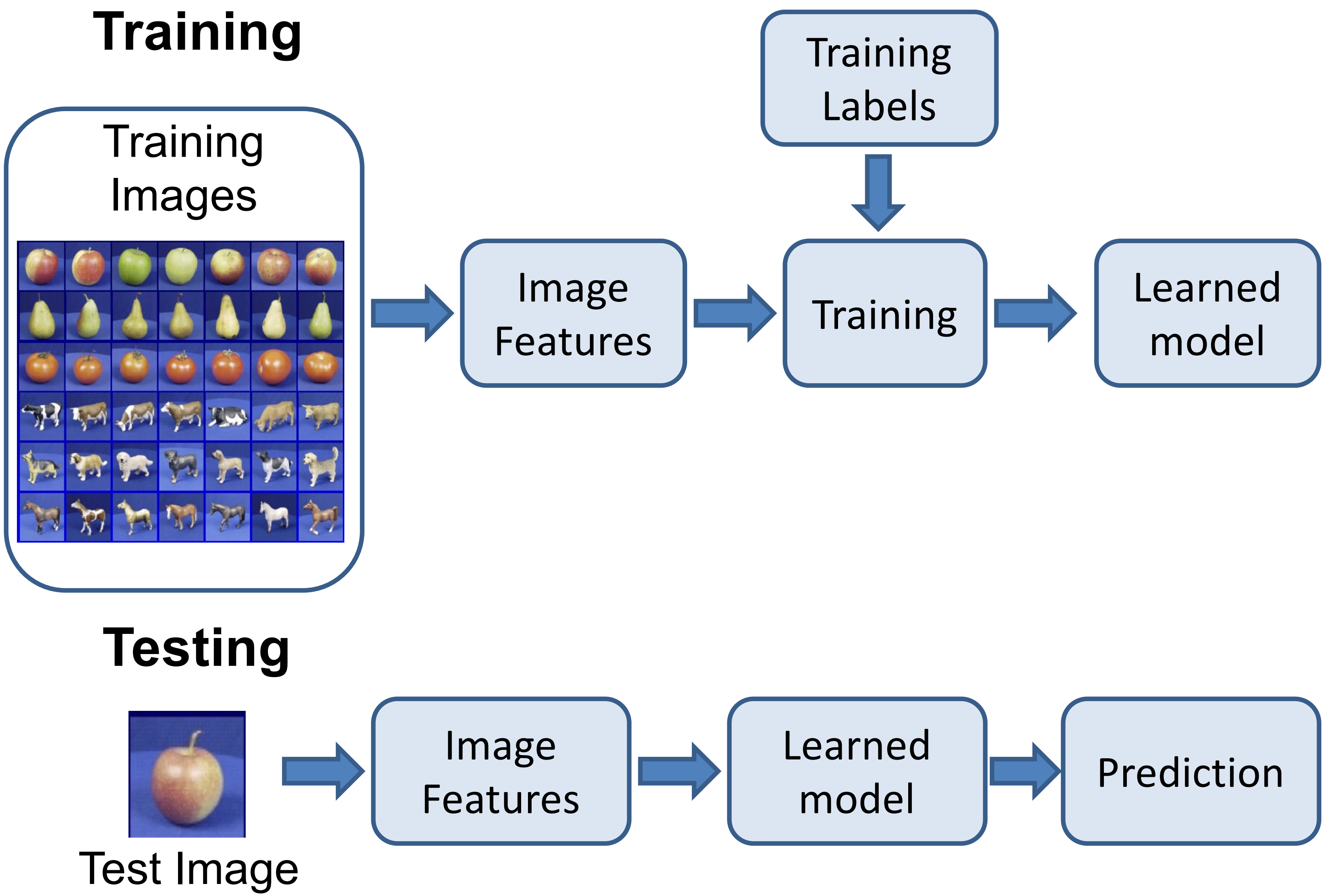
# Classification

$$y = f(\mathbf{x})$$

**output** **prediction function** **feature vector**

- **Training:** given a *training set* of labeled examples $\{(\mathbf{x}_1,y_1), \ldots, (\mathbf{x}_N,y_N)\}$, estimate the prediction function f by minimizing the prediction error on the training set
- **Testing:** apply f to a never before seen *test example* **x** and output the predicted value $y = f(\mathbf{x})$

# Classification in computer vision



# Algorithms

- Classification
  - **Supervised**, **categorical** labels
  - Bayesian classifier, KNN, SVM, Decision Tree, Neural Network, etc.
- Clustering
  - **Unsupervised**, **categorical** labels
  - Mixture models, K-means clustering, Hierarchical clustering, etc.
- Regression
  - **Supervised or Unsupervised**, **real**-valued labels

# Features

- Raw pixels
  - Use directly the color values captured by the sensor
- Low level features
  - These features are very objective features
    - Color, texture, shape, motion
- Middle level features
  - Features resulting from a decision process (related to the existence of some subjective details)
    - Segmentation of certain shapes
    - Identification of certain objects, types of content
- High level features
  - Features with some semantic content information, highly contextual and based on prior knowledge.
    - Person A is talking to person B

# An example*

- **Problem**: sorting incoming fish on a conveyor belt according to species
- Assume that we have only two kinds of fish:
  - Salmon
  - Sea bass

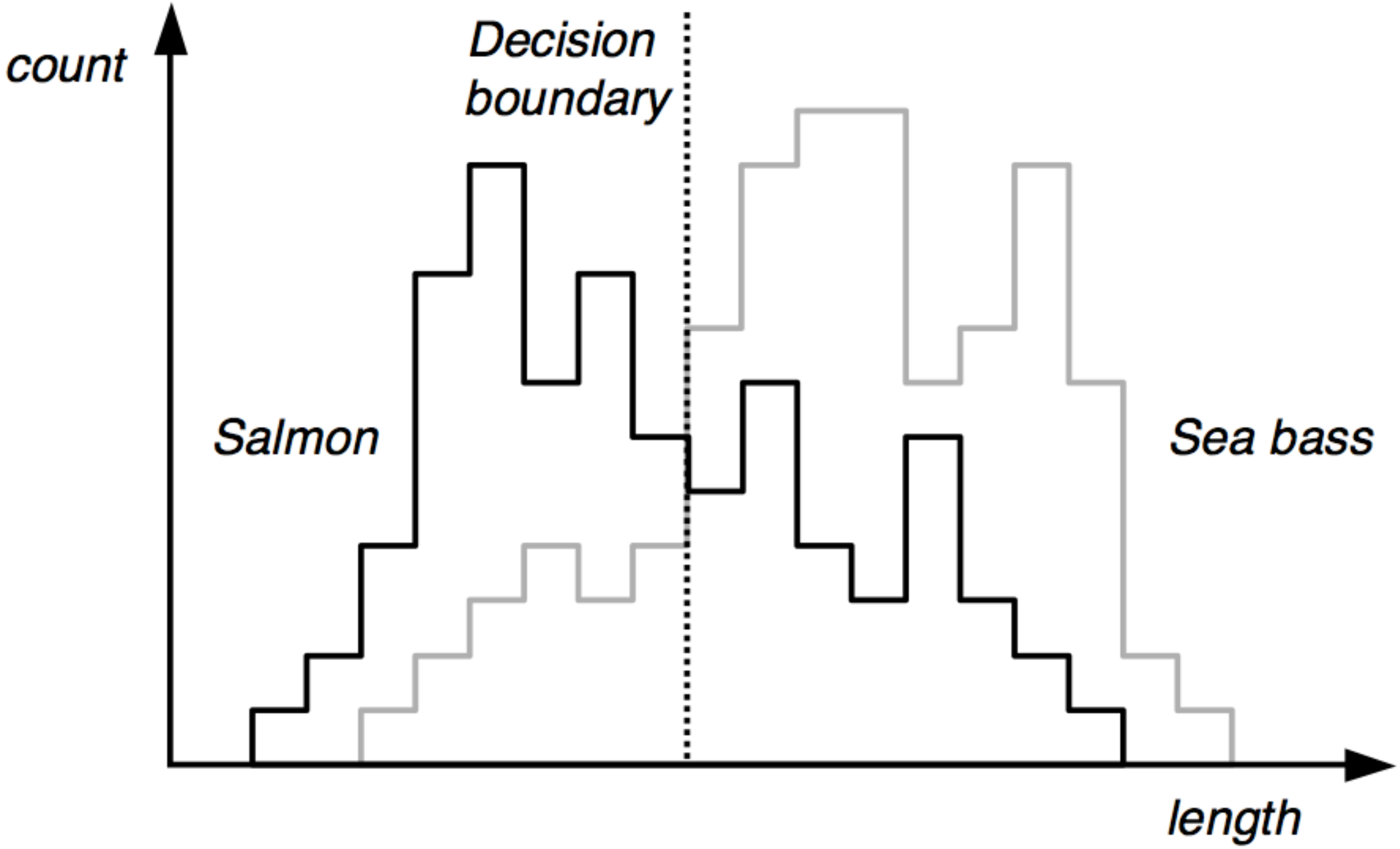
Picture taken with a camera

**Adapted from Duda, Hart and Stork, Pattern Classification, 2nd Ed.*

# An example: decision process

- What kind of information can distinguish one species from the other?
  - Length, width, weight, number and shape of fins, tail shape, etc.
- What can cause problems during sensing?
  - Lighting conditions, position of fish on the conveyor belt, camera noise, etc.
- What are the steps in the process?
  - Capture image -> isolate fish -> take measurements -> make decision

# An example: features

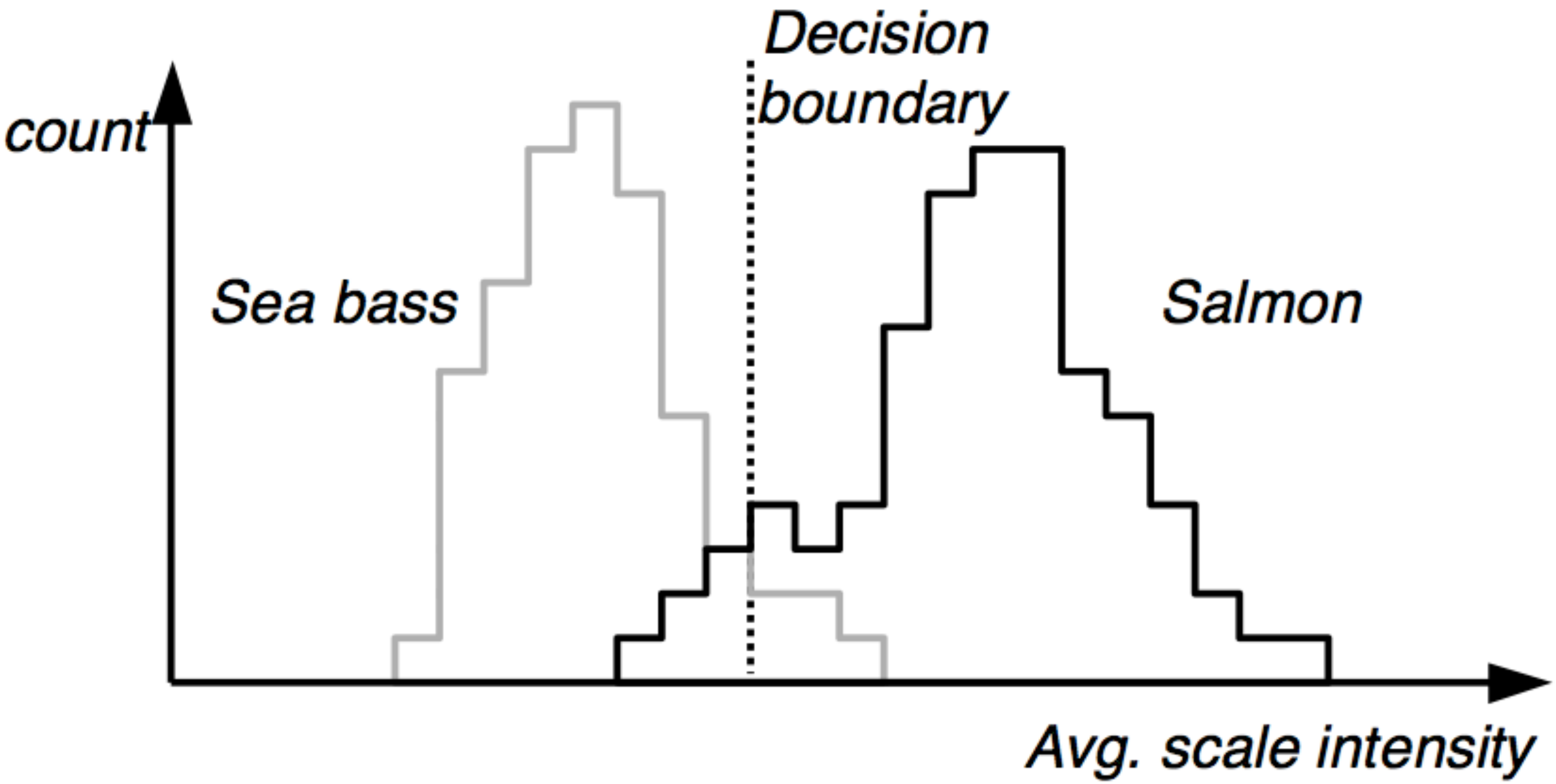



We estimate the system's probability of error and obtain a discouraging result of 40%. Can we improve this result?

# An example: features

- Even though sea bass is longer than salmon on the average, there are many examples of fish where this observation does not hold
- Committed to achieve a higher recognition rate, we try a number of features
  - Width, Area, Position of the eyes w.r.t. mouth...
  - only to find out that these features contain no discriminatory information
- Finally we find a “good” feature: **average intensity of the scales**

# An example: features




**Histogram** of the lightness feature for two types of fish in **training samples**. It looks easier to choose the threshold but we still can not make a perfect decision.

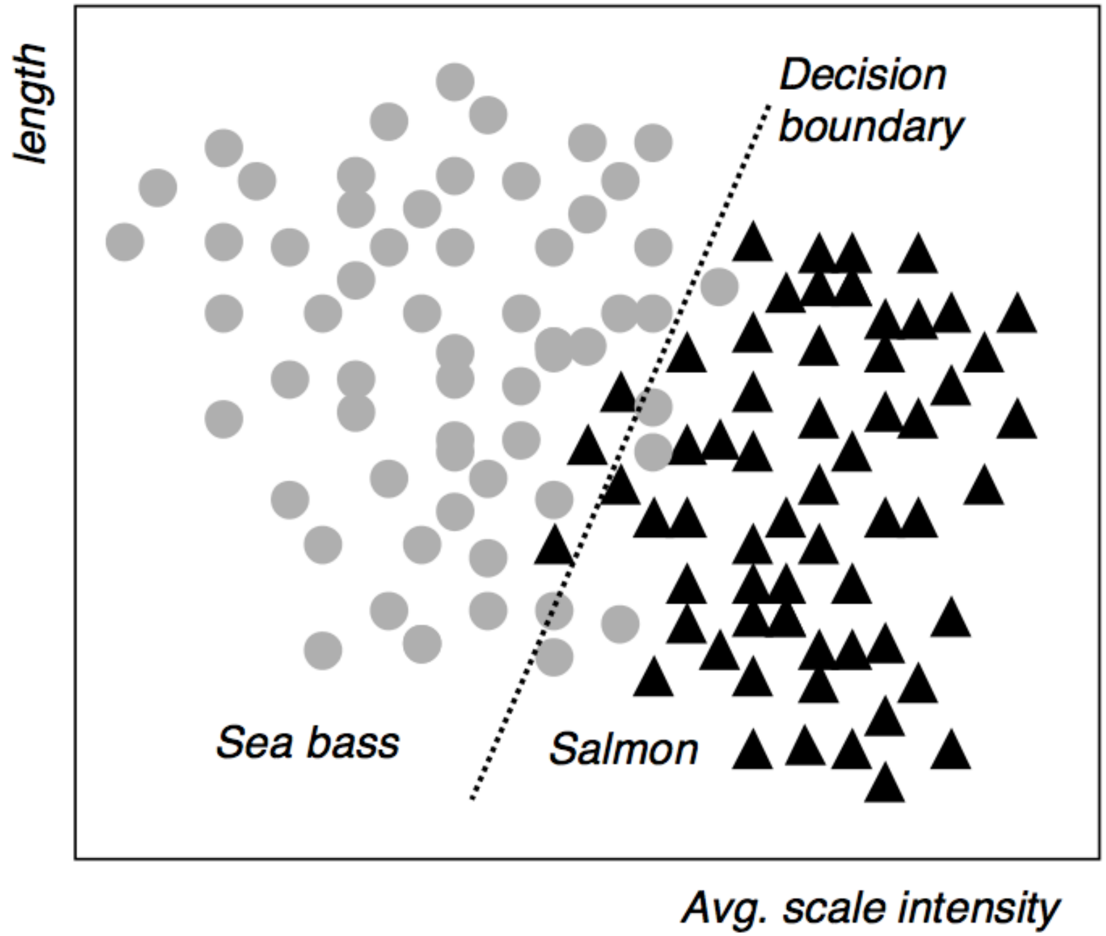
# An example: multiple features

- We can use two features in our decision:
  - lightness: $\boldsymbol{x}_1$
  - length: $\boldsymbol{x}_2$
- Each fish image is now represented as a point (feature vector)

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

in a two-dimensional **feature space**.

# An example: multiple features

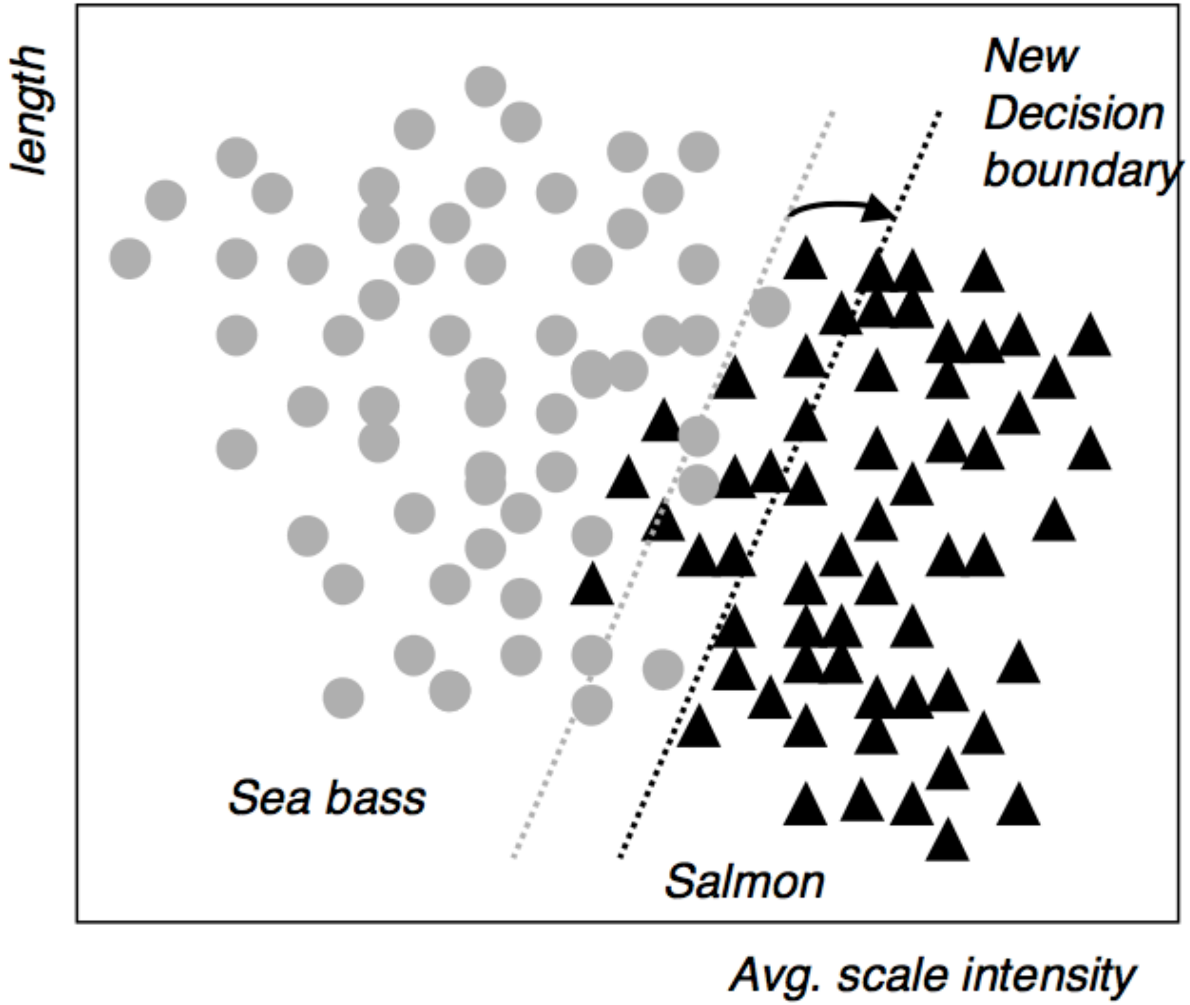



Scatter plot of lightness and length features for training samples. We can compute a **decision boundary** to divide the feature space into two regions with a classification rate of 95.7%.

# An example: cost of error

- We should also consider **costs of different errors** we make in our decisions.
- For example, if the fish packing company knows that:
  - Customers who buy salmon will object vigorously if they see sea bass in their cans.
  - Customers who buy sea bass will not be unhappy if they occasionally see some expensive salmon in their cans.
- How does this knowledge affect our decision?

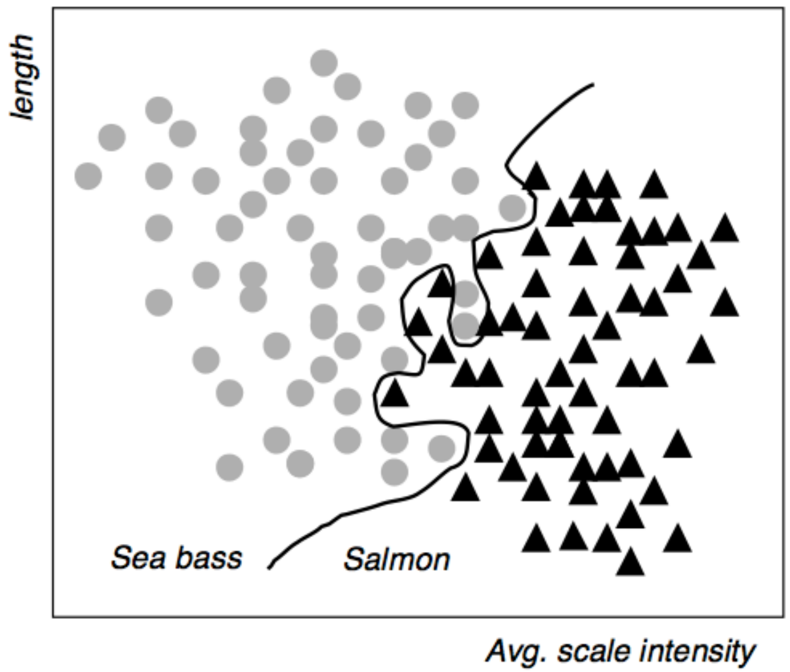
# An example: cost of error




We could intuitively shift the decision boundary to minimize an alternative cost function

# An example: generalization

- **The issue of generalization**
  - The recognition rate of our linear classifier (95.7%) met the design specifications, but we still think we can improve the performance of the system
  - We then design a über-classifier that obtains an impressive classification rate of 99.9975% with the following decision boundary



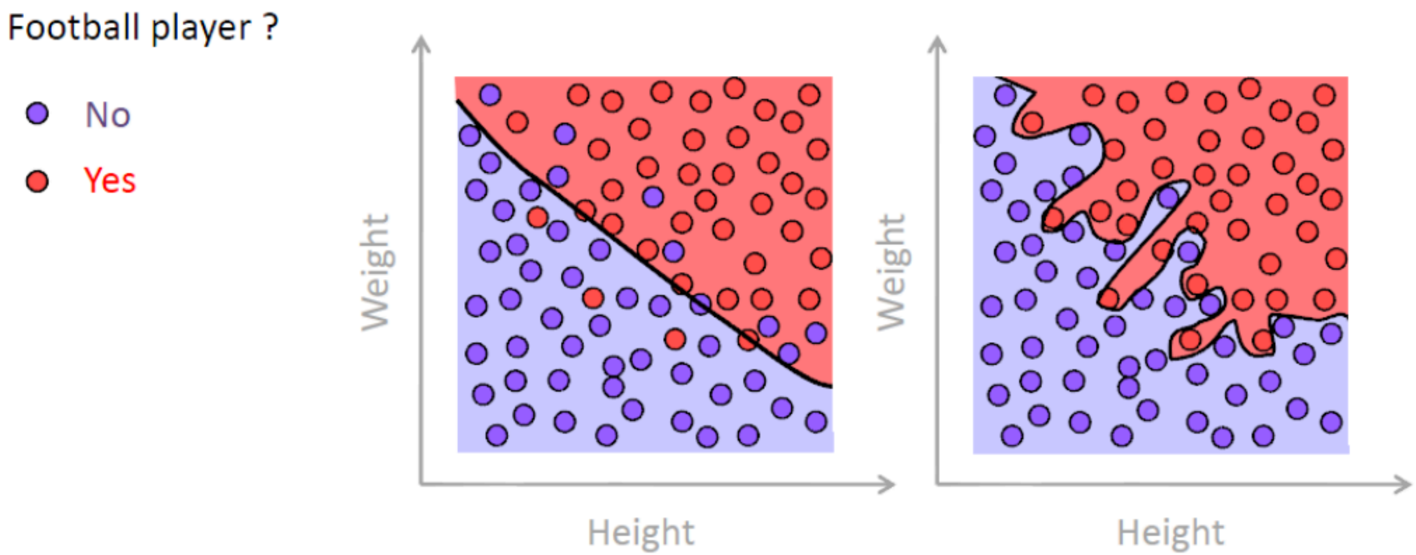
# An example: generalization

- **The issue of generalization**
  - Satisfied with our classifier, we integrate the system and deploy it to the fish processing plant
  - A few days later the plant manager calls to complain that the system is misclassifying an average of 25% of the fish

- **What went wrong?**

# Overfitting

- If we allow very complicated classifiers, we could overfit the training data



# INSTANCE-BASED LEARNING

# k-Nearest neighbour classifier

- Given the training data $D = \{x_1,\ldots,x_n\}$ as a set of n labeled examples, the **nearest neighbour classifier** assigns a test point $x$ the label associated with its closest neighbour (or $k$ neighbours) in $D$.

- Closeness is defined using a **distance function**.

# Distance functions

- A general class of metrics for *d*-dimensional patterns is the **Minkowski metric**, also known as the $L_p$ norm

$$L_p(\mathbf{x},\mathbf{y}) = \left( \sum_{i=1}^{d} \left| x_i - y_i \right|^p \right)^{1/p}$$

- The **Euclidean distance** is the $L_2$ norm

$$L_2(\mathbf{x},\mathbf{y}) = \left( \sum_{i=1}^{d} \left| x_i - y_i \right|^2 \right)^{1/2}$$

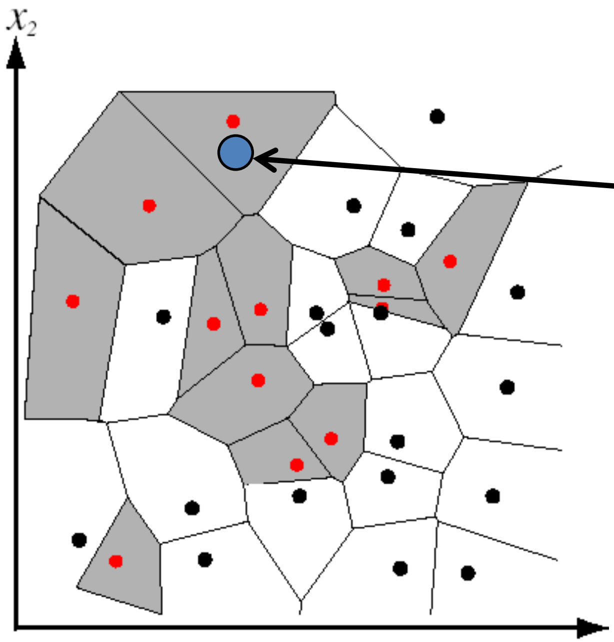
- The **Manhattan** or **city block distance** is the L1 norm

$$L_1(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{d} \left| x_i - y_i \right|$$

# 1-Nearest neighbour classifier

Assign label of nearest training data point to each test data point

Black = negative
Red = positive




from Duda *et al.*

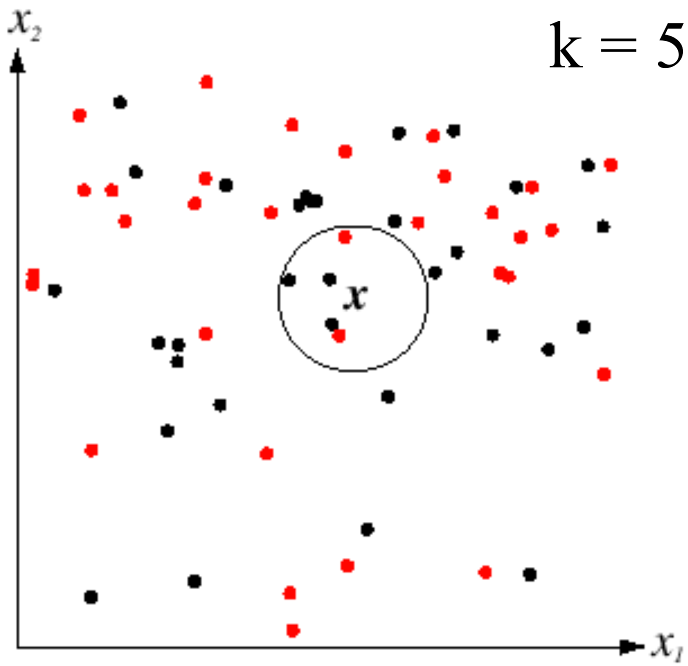
Novel test example

Closest to a positive example from the training set, so classify it as positive.

Voronoi partitioning of feature space
for 2-category 2D data

# k-Nearest neighbour classifier

- For a new point, find the $k$ closest points from training data
- Labels of the $k$ points "vote" to classify




Black = negative
Red = positive

If the query lands here, the 5 NN consist of 3 negatives and 2 positives, so we classify it as negative.

# kNN as a classifier

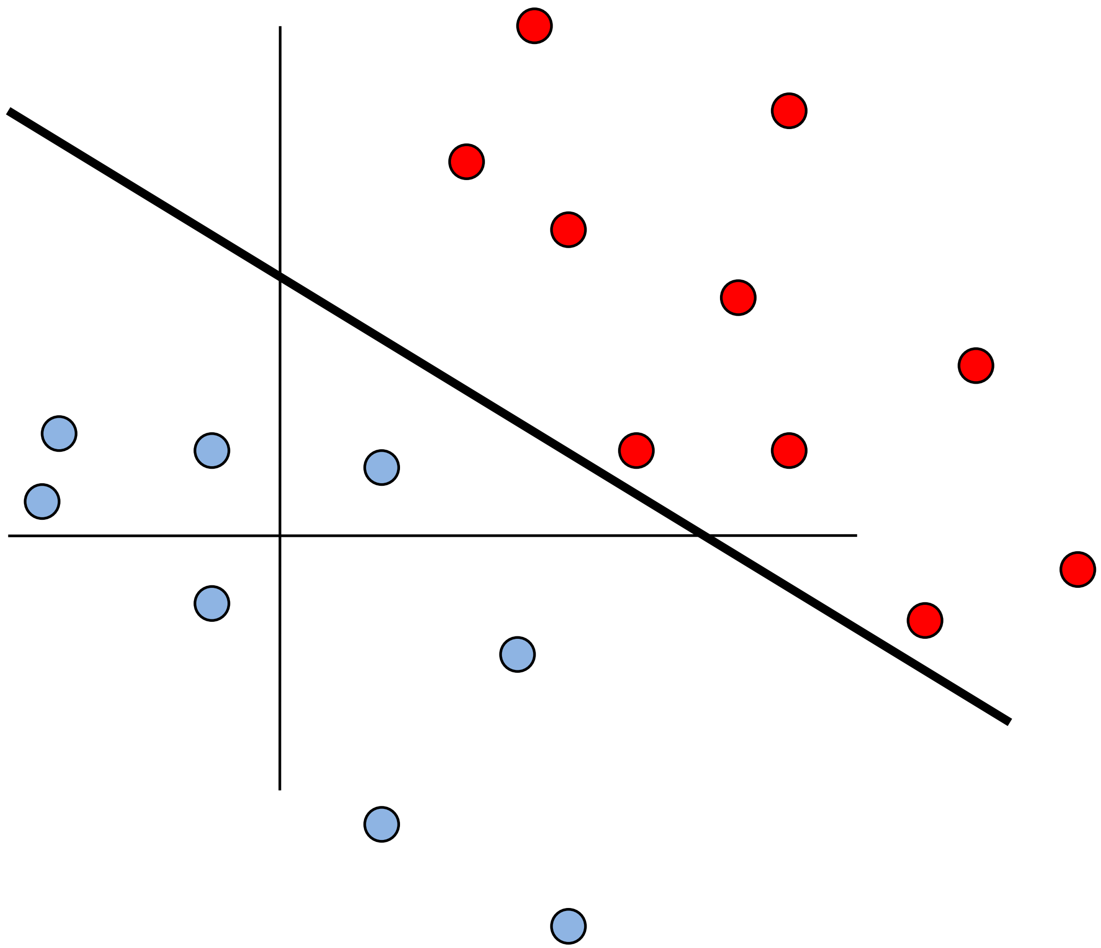
- **Advantages**:
  - Simple to implement
  - Naturally handles multi-class cases
  - Can do well in practice with enough representative data
- **Disadvantages:**
  - Large search problem to find nearest neighbors → Highly susceptible to the **curse of dimensionality**
  - Storage of data
  - Must have a meaningful distance function

# The curse of dimensionality

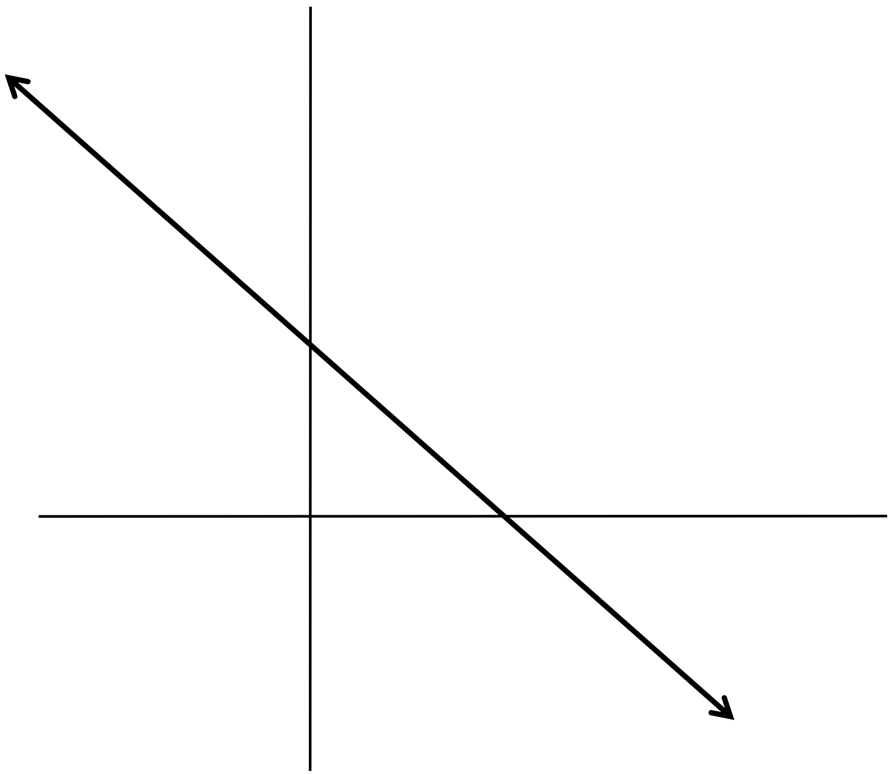
- KNN is easily misled in a high-dimension space – why?
  - Easy problems in low-dim are hard in hi-dim
  - Low-dim intuitions do not apply in hi-dim
- The curse of dimensionality
  - The number of examples needed to accurately train a classifier **grows exponentially** with the dimensionality of the model
  - For a given sample size, there is a maximum number of features above which the classifier's performance **degrades** rather than improves

# SUPPORT VECTOR MACHINES

# Linear classifiers



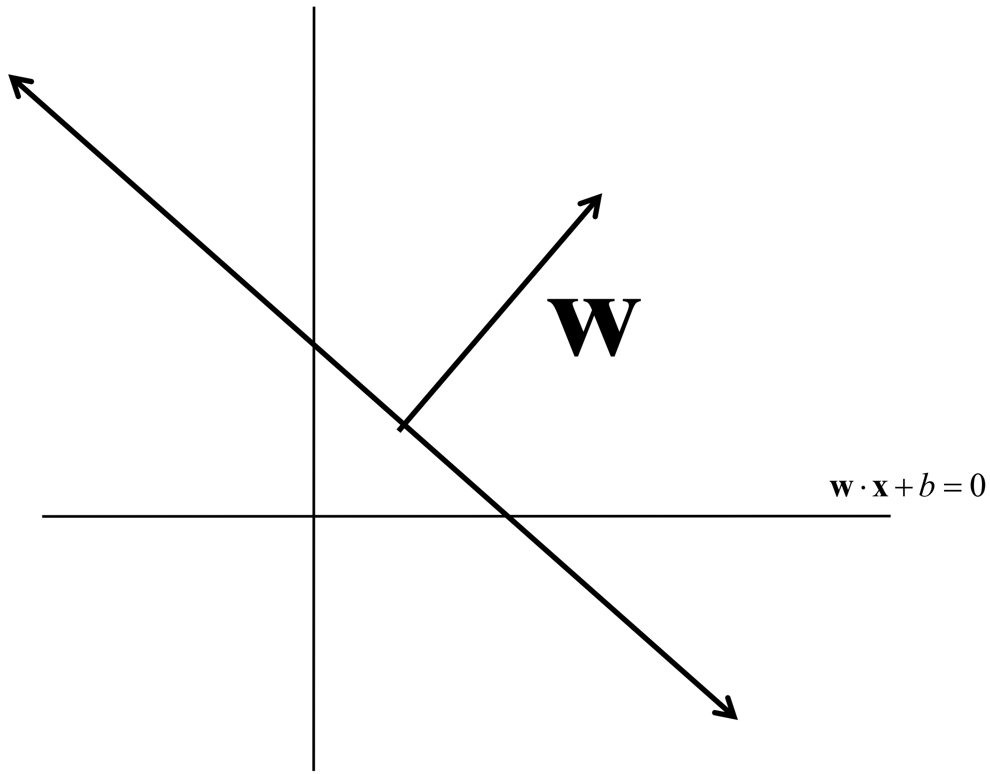
# Linear functions in $R^2$



Let $\mathbf{w} = \begin{bmatrix} a \\ c \end{bmatrix}$ $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

$$ax + cy + b = 0$$

# Linear functions in $R^2$



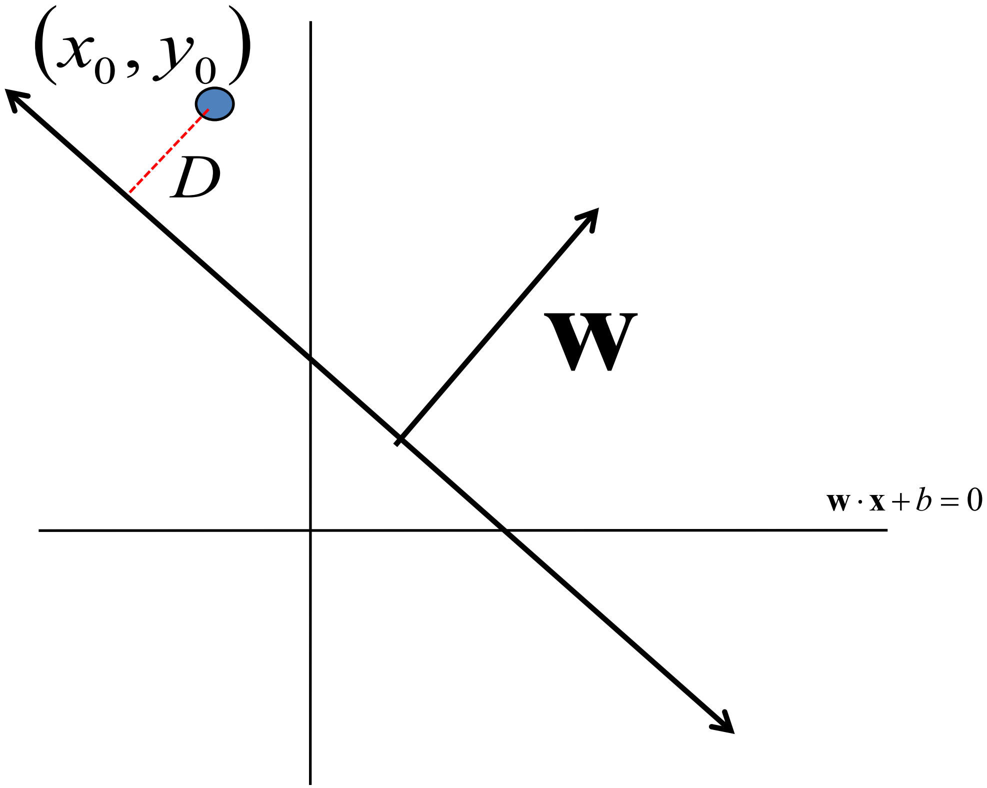

Let $\mathbf{w} = \begin{bmatrix} a \\ c \end{bmatrix}$ $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

$$ax + cy + b = 0$$

$$\Updownarrow$$

$$\mathbf{w}^T \mathbf{x} + b = 0$$

# Linear functions in $R^2$



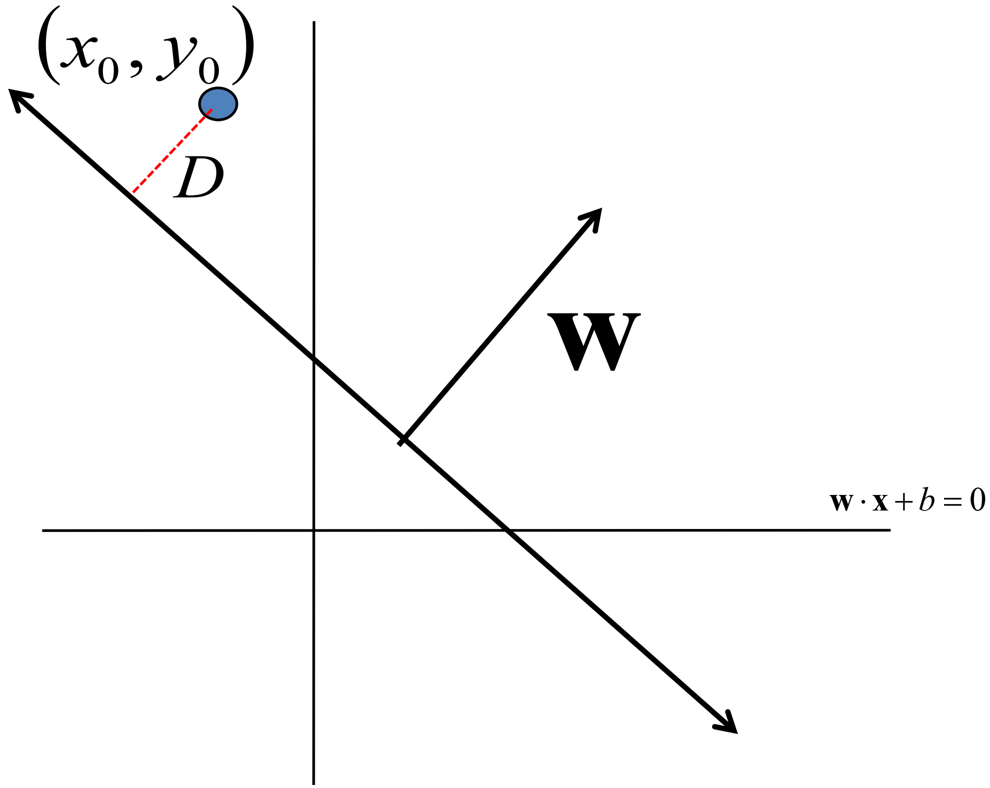

Let $\mathbf{w} = \begin{bmatrix} a \\ c \end{bmatrix}$ $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

$$ax + cy + b = 0$$

$$\Updownarrow$$

$$\mathbf{w}^T \mathbf{x} + b = 0$$

# Linear functions in $R^2$




Let $\mathbf{w} = \begin{bmatrix} a \\ c \end{bmatrix}$ $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

$$ax + cy + b = 0$$

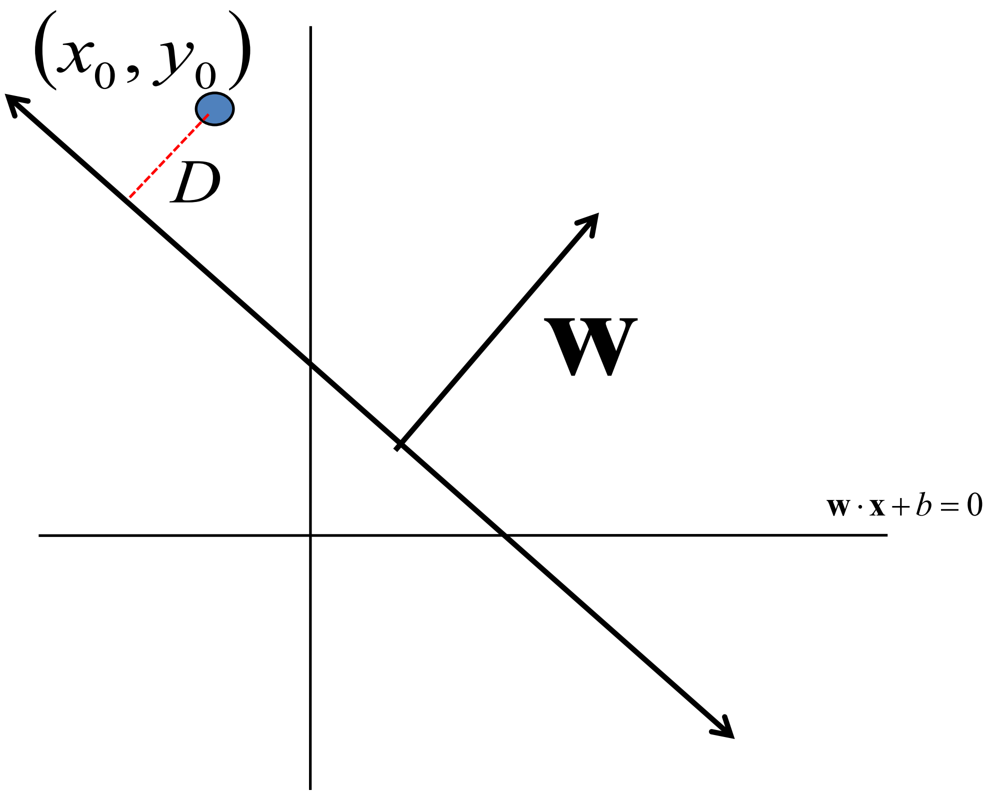
$$\Updownarrow$$

$$\mathbf{w}^T \mathbf{x} + b = 0$$

$$D = \frac{|ax_0 + cy_0 + b|}{\sqrt{a^2 + c^2}}$$

distance from point to line

# Linear functions in $R^2$




Let $\mathbf{w} = \begin{bmatrix} a \\ c \end{bmatrix}$ $\quad \mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

$$ax + cy + b = 0$$

$$\Updownarrow$$

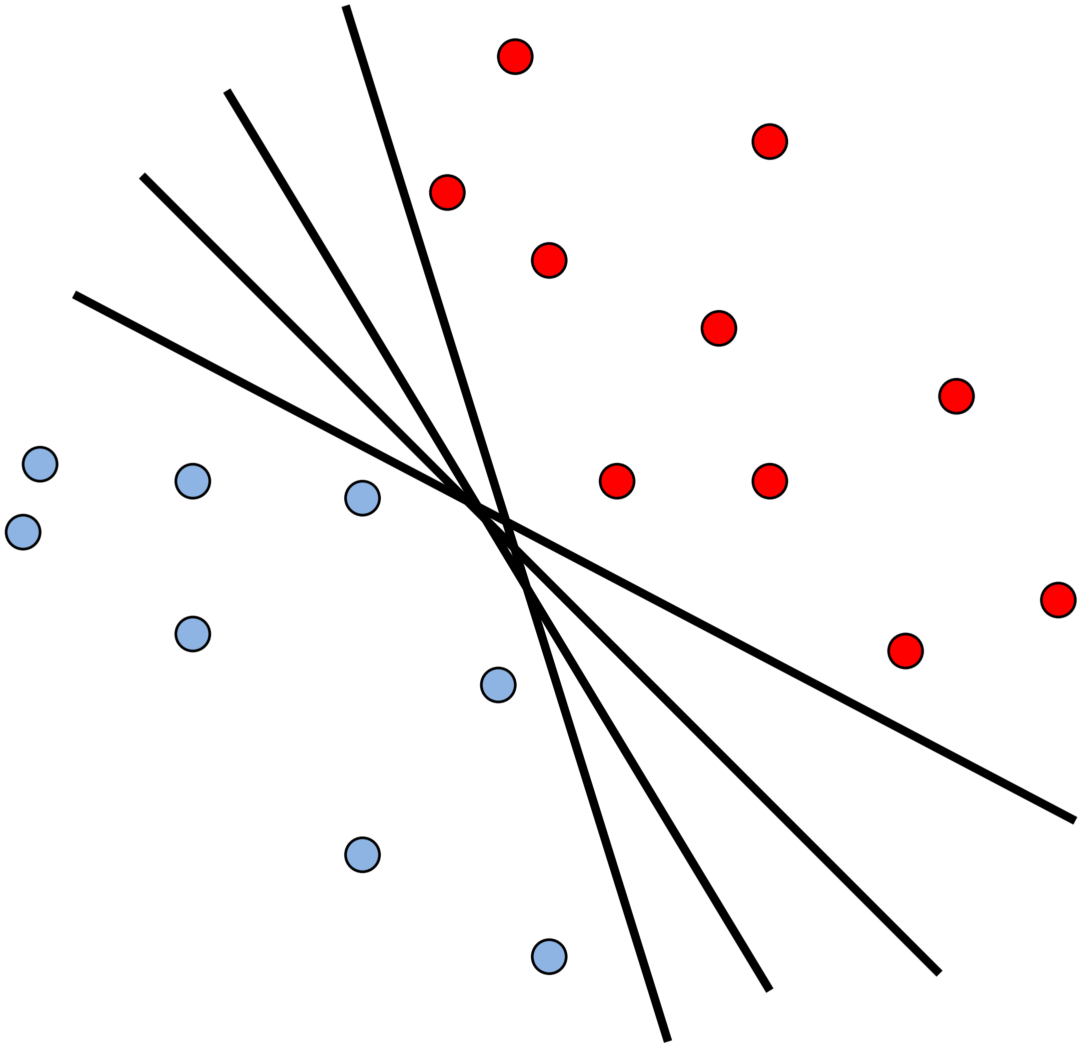
$$\mathbf{w}^T \mathbf{x} + b = 0$$

$$D = \frac{|ax_0 + cy_0 + b|}{\sqrt{a^2 + c^2}} = \frac{\mathbf{w}^{\mathrm{T}}\mathbf{x} + b}{\|\mathbf{w}\|}$$

distance from point to line

# Linear classifiers

Find linear function to separate positive and negative examples

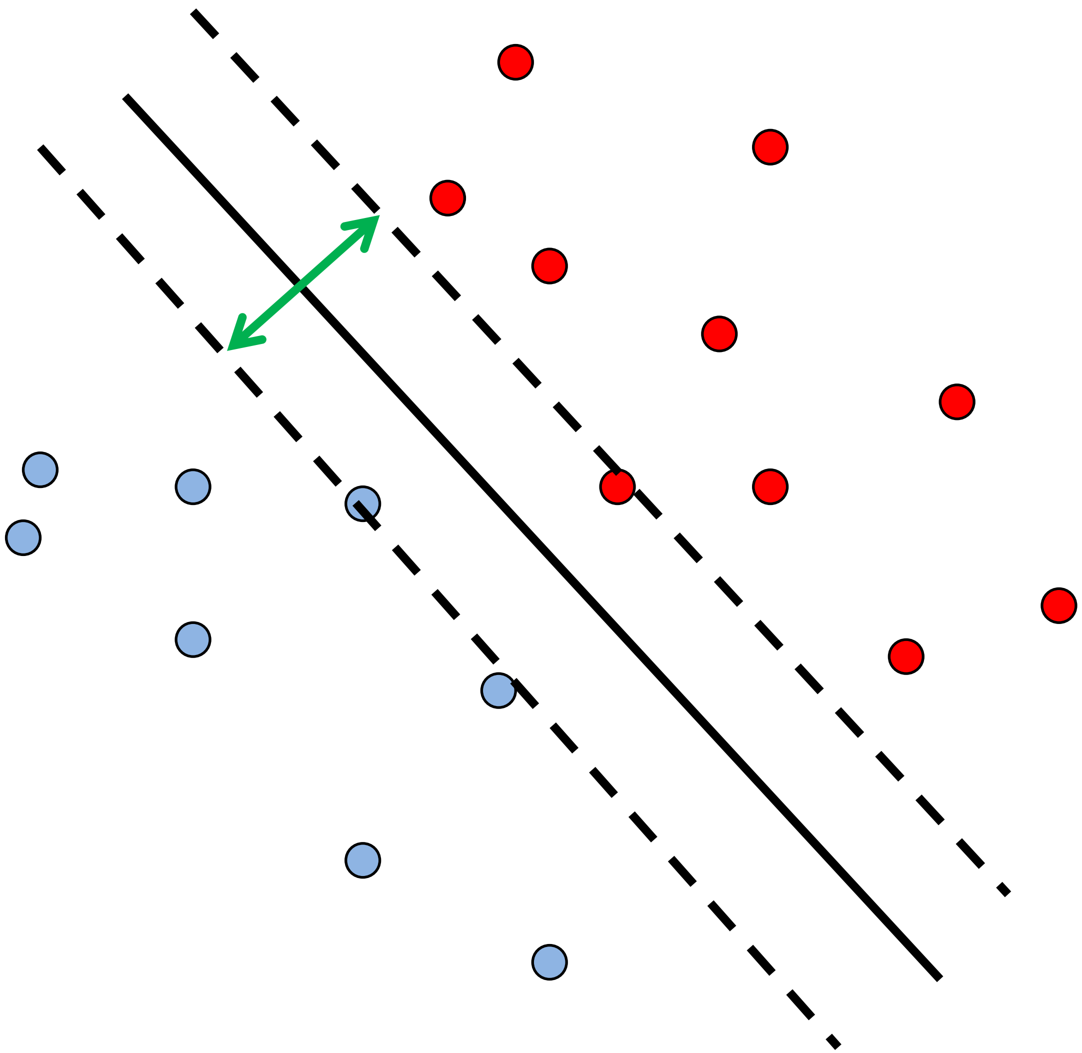


$\mathbf{x}_i$ positive : $\quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 0$

$\mathbf{x}_i$ negative : $\quad \mathbf{x}_i \cdot \mathbf{w} + b < 0$

Which line is best?

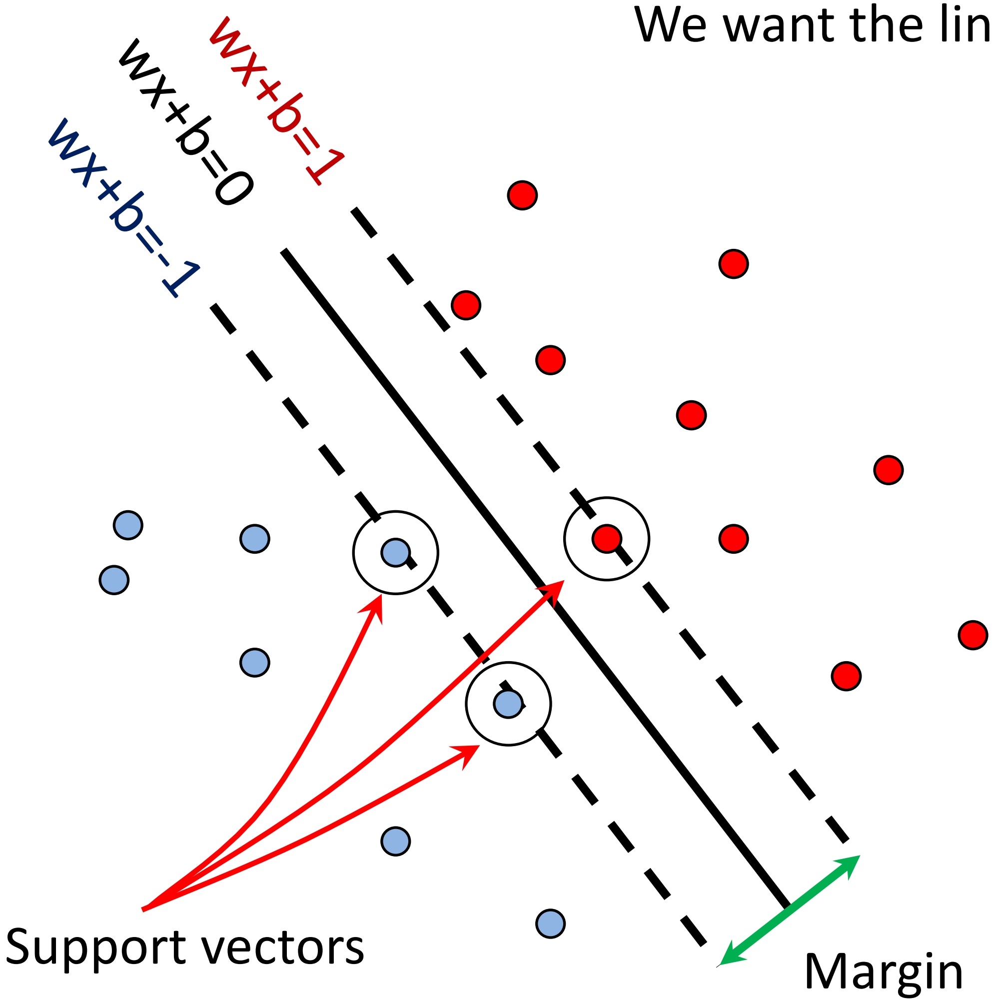
# Support Vector Machines



Classifier based on *optimal separating line (for 2D case)*

Maximize the ***margin*** between the positive and negative training examples

# Support Vector Machines



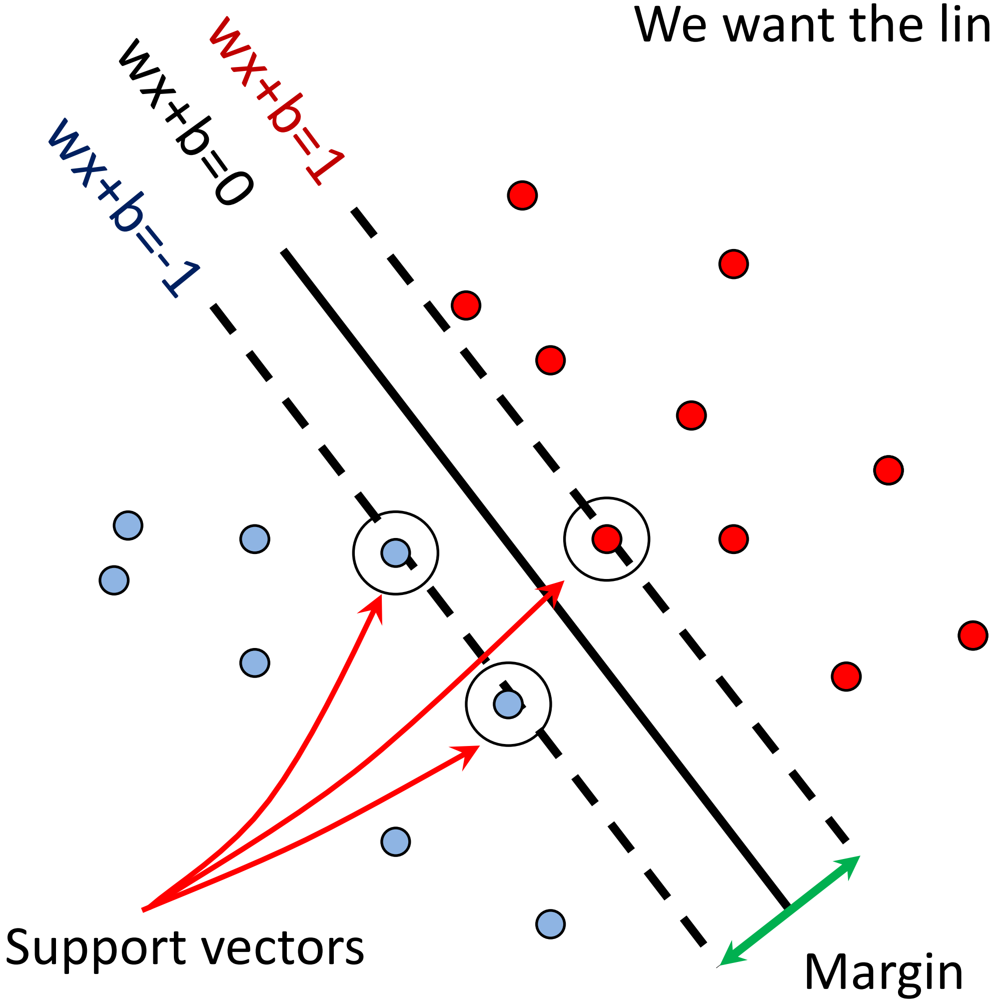

We want the line that maximizes the margin.

$$\mathbf{x}_i \text{ positive } (y_i = 1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

$$\mathbf{x}_i \text{ negative } (y_i = -1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

For support, vectors, $\quad \mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$

# Support Vector Machines




We want the line that maximizes the margin.

$\mathbf{x}_i$ positive $(y_i = 1)$: $\quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$

$\mathbf{x}_i$ negative $(y_i = -1)$: $\quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$

For support, vectors, $\quad \mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$

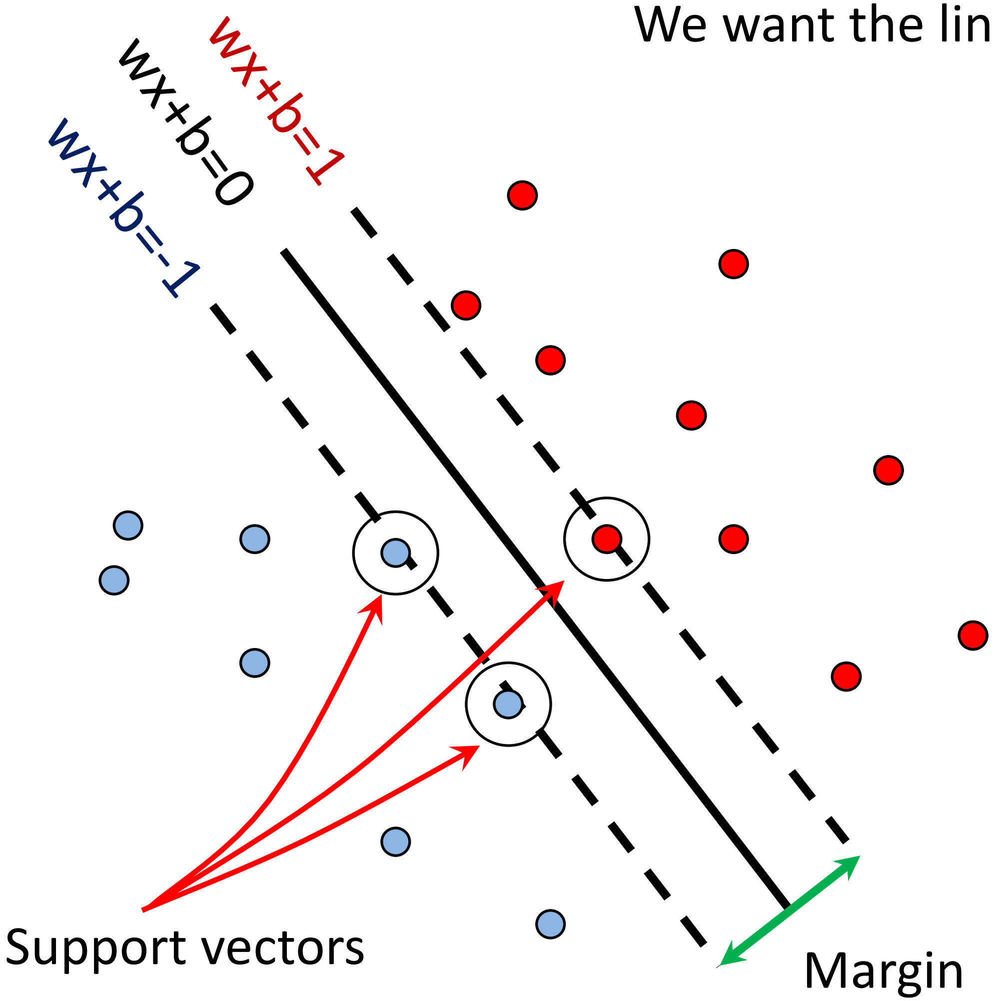
Distance between point and line: $\quad \dfrac{|\mathbf{x}_i \cdot \mathbf{w} + b|}{\|\mathbf{w}\|}$

For support vectors:

$$\frac{\mathbf{w}^T\mathbf{x}+b}{\|\mathbf{w}\|} = \frac{\pm 1}{\|\mathbf{w}\|} \qquad M = \left|\frac{1}{\|\mathbf{w}\|} - \frac{-1}{\|\mathbf{w}\|}\right| = \frac{2}{\|\mathbf{w}\|}$$

# Support Vector Machines

We want the line that maximizes the margin.




$\mathbf{x}_i$ positive $(y_i = 1)$: $\quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$

$\mathbf{x}_i$ negative $(y_i = -1)$: $\quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$

For support, vectors, $\quad \mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$

Distance between point and line: $\quad \dfrac{|\mathbf{x}_i \cdot \mathbf{w} + b|}{\|\mathbf{w}\|}$

Therefore, the margin is 2 / ||w||

# Finding the maximum margin line

1. Maximize margin $2/\|\mathbf{w}\|$
2. Correctly classify all training data points:

$$\mathbf{x}_i \text{ positive } (y_i = 1): \qquad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

$$\mathbf{x}_i \text{ negative } (y_i = -1): \qquad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

*Quadratic optimization problem*:

Minimize $\frac{1}{2}\mathbf{w}^T\mathbf{w}$

Subject to $y_i(\mathbf{w}\cdot\boldsymbol{x}_i+b) \geq 1$

# Finding the maximum margin line

- Solution: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$

learned weight

support vector

# Finding the maximum margin line

- Solution: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$

$$\mathbf{w} \cdot \mathbf{x} + b = \sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b$$

- Classification function:

$$\begin{aligned} f(x) &= \operatorname{sign}(\mathbf{w} \cdot \mathbf{x} + \mathrm{b}) \\ &= \operatorname{sign}\left(\sum_{\mathrm{i}} \alpha_{\mathrm{i}} \boxed{\mathbf{x}_i \cdot \mathbf{x}} + b\right) \end{aligned}$$

*If f(x) < 0, classify as negative,*
*if f(x) > 0, classify as positive*

# Questions

- **What if the features are not 2D?**
- What if the data is not linearly separable?
- What if we have more than just two categories?

# Questions

- What if the features are not 2D?
  - Generalizes to d-dimensions – replace line with "hyperplane"
- What if the data is not linearly separable?
- What if we have more than just two categories?

# Questions

- What if the features are not 2D?
- **What if the data is not linearly separable?**
- What if we have more than just two categories?

# Soft-margin SVMs

- Introduce **slack variable** and allow some instances to fall within the margin, but penalize them
- Constraint becomes: $y_i(w \cdot x_i + b) \geq 1 - \xi_i, \ \forall x_i$

$$\xi_i \geq 0$$

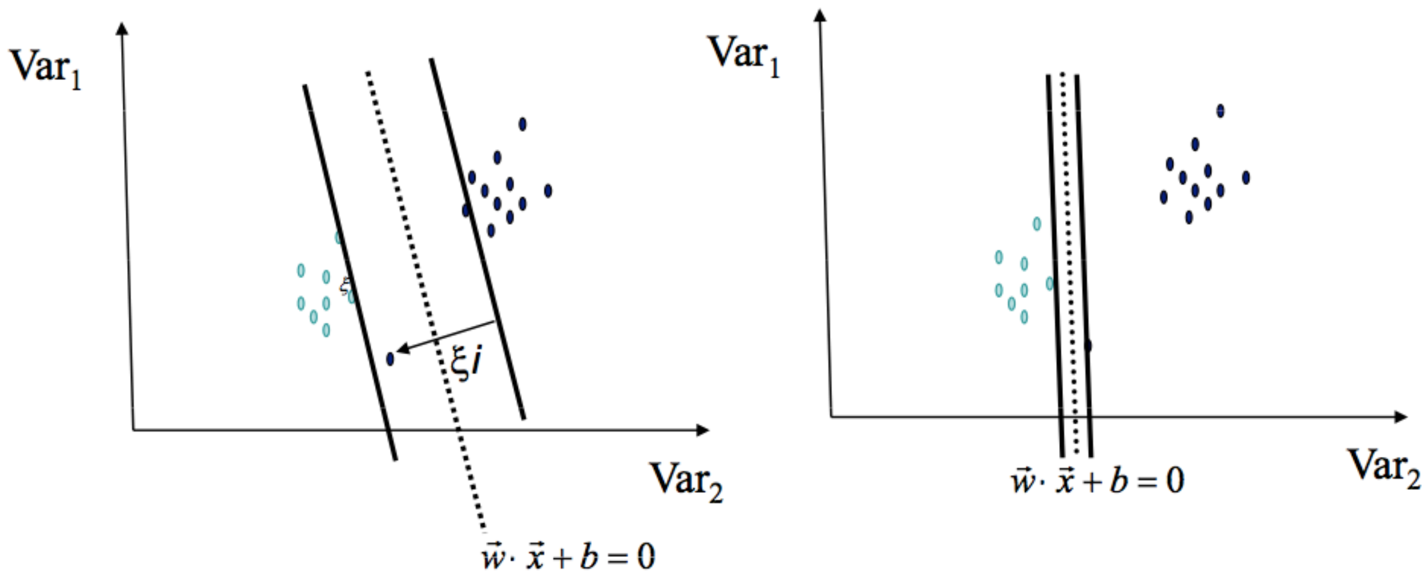
- Objective function penalizes for misclassified instances within the margin

$$\min \frac{1}{2}\|w\|^2 + C\sum_i \xi_i$$

- C trades-off margin width and classifications
- As *C*→∞, we get closer to the hard-margin solution

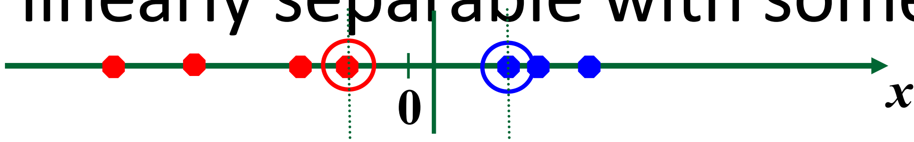
# Soft-margin vs Hard-margin SVMs

- Soft-Margin always has a solution
- Soft-Margin is more robust to outliers
  - Smoother surfaces (in the non-linear case)
- Hard-Margin does not require to guess the cost parameter (requires no parameters at all)



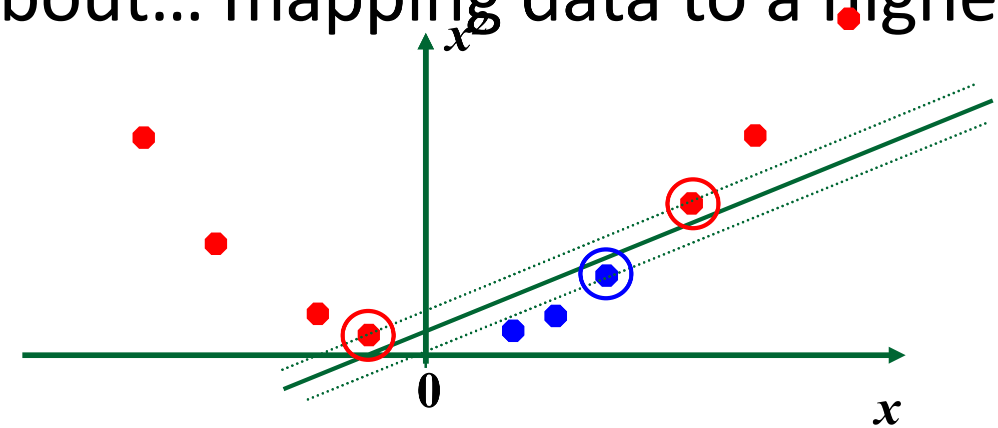
# Non-linear SVMs

- Datasets that are linearly separable with some noise work out great:



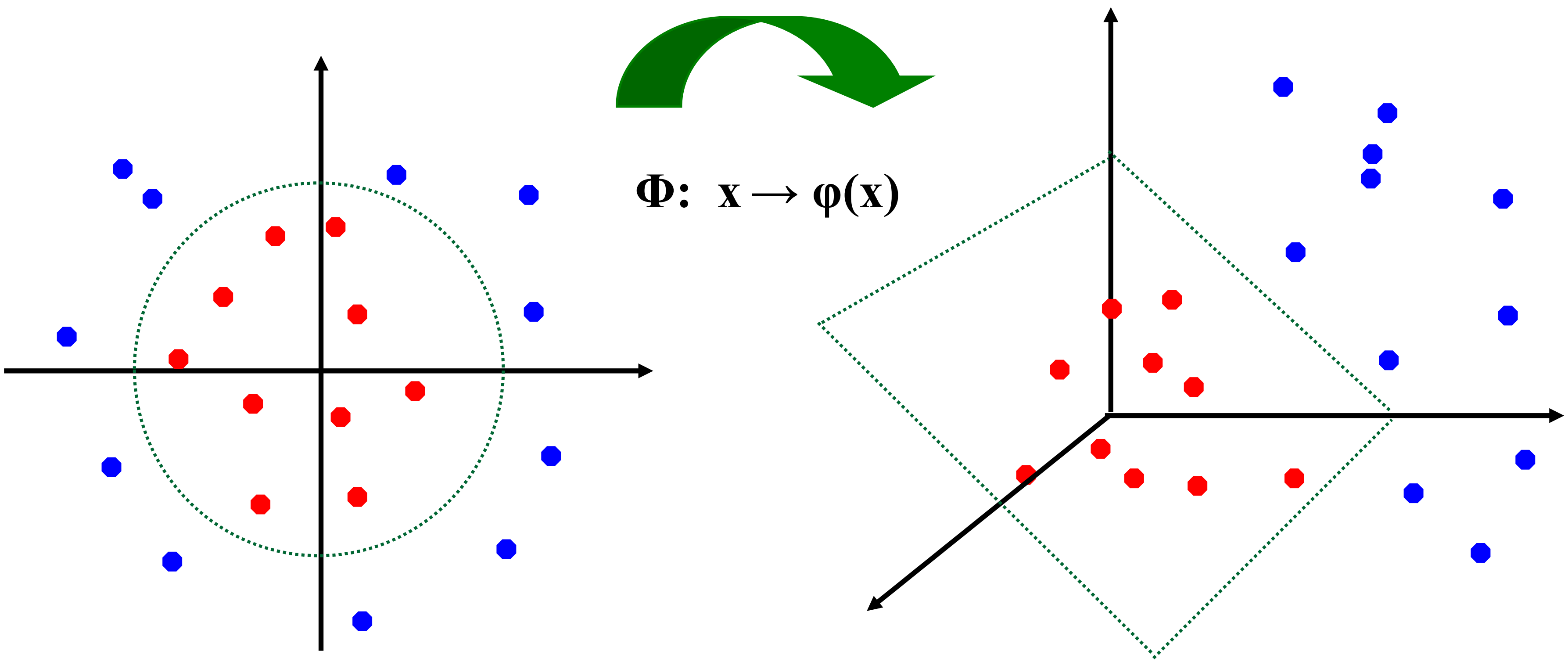

- But what are we going to do if the dataset is just too hard?


- How about… mapping data to a higher-dimensional space:



# Non-linear SVMs

- General idea: the original input space can be **mapped** to some higher-dimensional feature space where the training set is separable:



# The “Kernel Trick”

- The linear classifier relies on dot product between vectors $K(x_i,x_j)=x_i^T x_j$

- If every data point is mapped into high-dimensional space via some transformation $\Phi$: $x \rightarrow \phi(x)$, the dot product becomes:

$$K(x_i,x_j)= \phi(x_i)^T \phi(x_j)$$

- A *kernel function* is a similarity function that corresponds to an inner product in some expanded feature space.

# Non-linear SVMs

- *The kernel trick*: instead of explicitly computing the lifting transformation $\boldsymbol{\varphi}(\mathbf{x})$, define a kernel function K such that

$$K(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\varphi}(\mathbf{x}_i) \cdot \boldsymbol{\varphi}(\mathbf{x}_j)$$

- This gives a nonlinear decision boundary in the original feature space:

$$\sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$$

# Examples of kernel functions

- Linear:

$$K(x_i, x_j) = x_i^T x_j$$

- Gaussian RBF:

$$K(x_i, x_j) = \exp\left(-\frac{\left\| x_i - x_j \right\|^2}{2\sigma^2}\right)$$

- Histogram intersection:

$$K(x_i, x_j) = \sum_k \min(x_i(k), x_j(k))$$

# Questions

- What if the features are not 2D?
- What if the data is not linearly separable?
- **What if we have more than just two categories?**

# Multi-class SVMs

- Achieve multi-class classifier by combining a number of binary classifiers
- **One vs. all**
  - Training: learn an SVM for each class vs. the rest
  - Testing: apply each SVM to test example and assign to it the class of the SVM that returns the highest decision value
- **One vs. one**
  - Training: learn an SVM for each pair of classes
  - Testing: each learned SVM “votes” for a class to assign to the test example

# SVM as a classifier

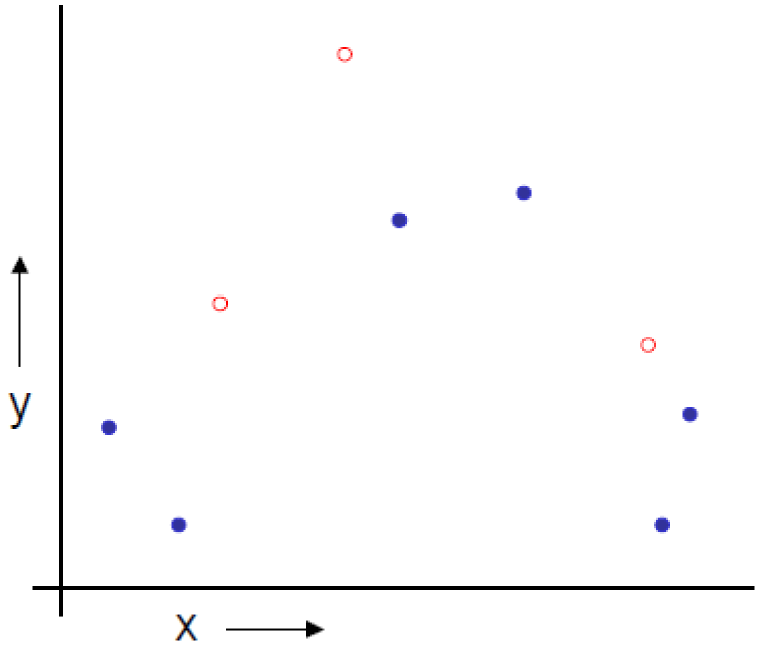
- Advantages
  - Many SVM packages available
  - Kernel-based framework is very powerful, flexible
  - Often a sparse set of support vectors – compact at test time
  - Works very well in practice, even with very small training sample sizes
- Disadvantages
  - No “direct” multi-class SVM, must combine two-class SVMs
  - Can be tricky to select best kernel function for a problem
  - Computation, memory
    - During training time, must compute matrix of kernel values for every pair of examples
    - Learning can take a very long time for large-scale problems

# CROSS VALIDATION

# Training - general strategy

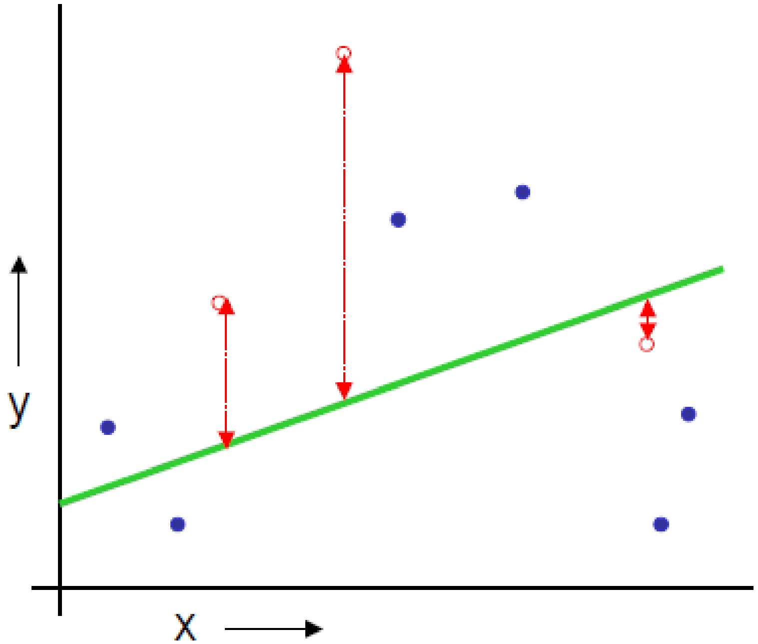
- We try to simulate the real world scenario.
- Test data is our future data.
- Validation set can be our test set - we use it to select our model.
- The whole aim is to estimate the models' true error on the sample data we have.

# Validation set method




- Randomly split some portion of your data. Leave it aside as the **validation set**
- The remaining data is the **training data**

# Validation set method



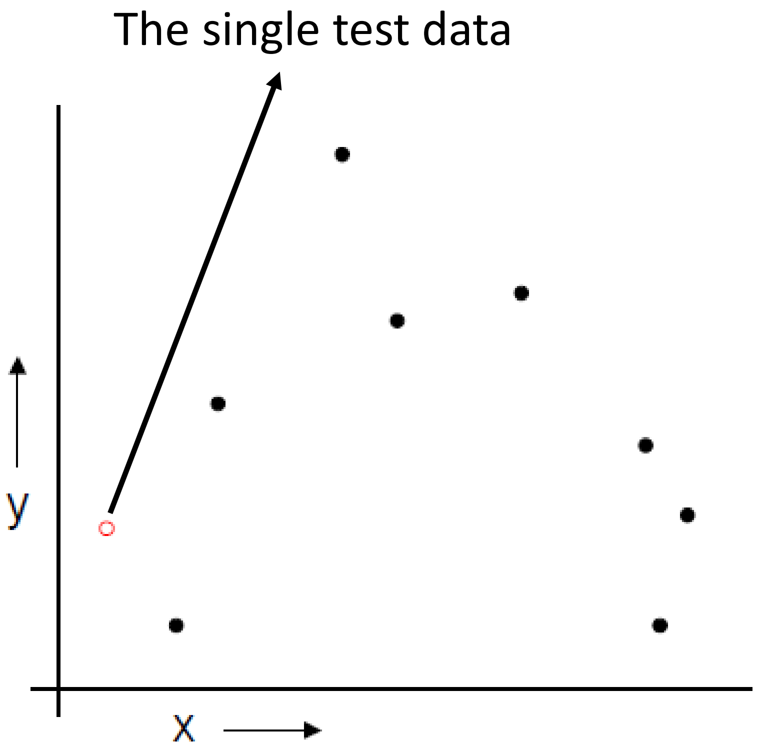

- Randomly split some portion of your data. Leave it aside as the **validation set**
- The remaining data is the **training data**
- Learn a **model** from the training set
- Estimate your future **performance** with the test data

# Test set method

- It is simple, however
  - We waste some portion of the data
  - If we do not have much data, we may be lucky or unlucky with our test data

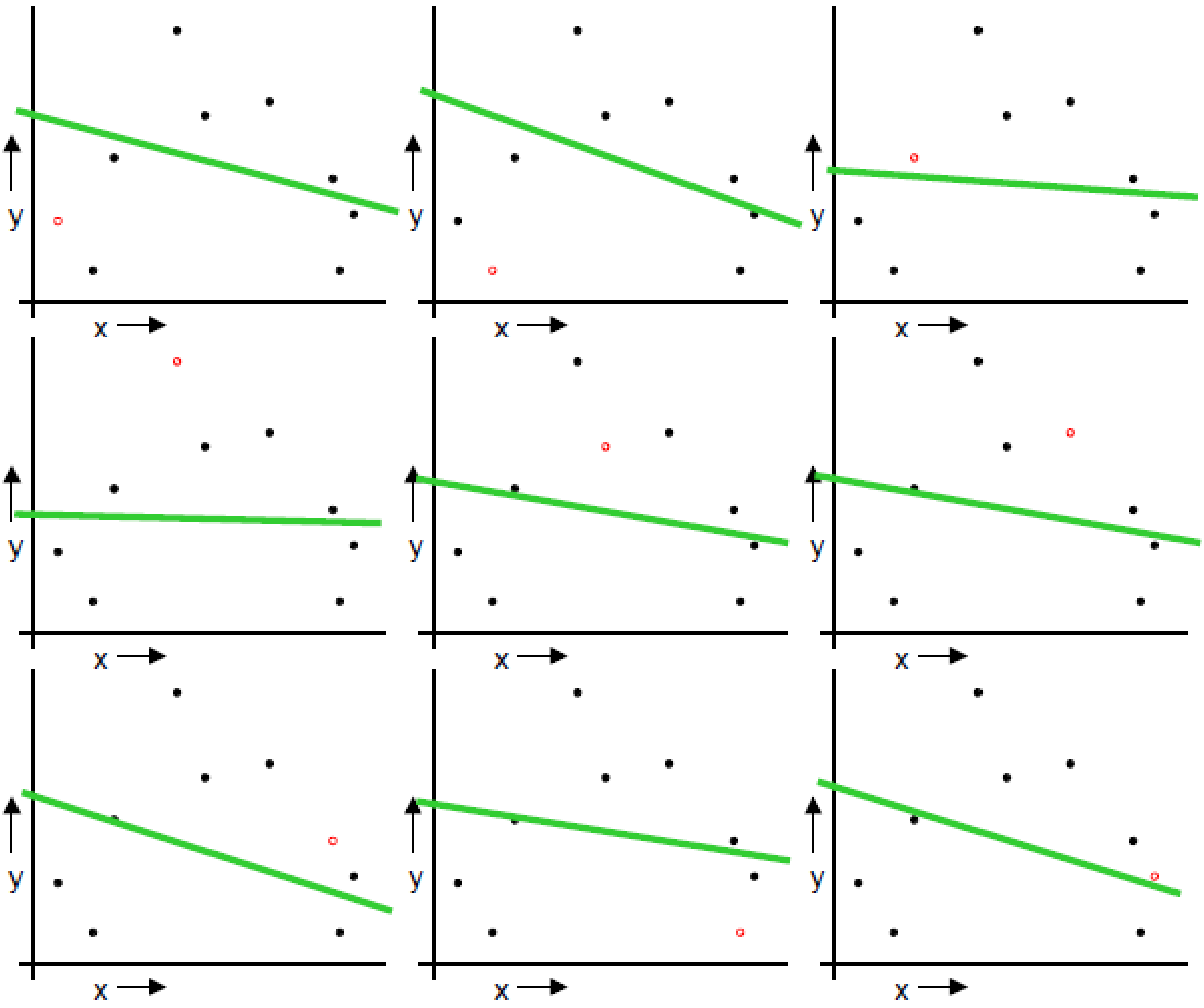
- With **cross-validation** we reuse the data

# LOOCV (Leave-one-out Cross Validation)



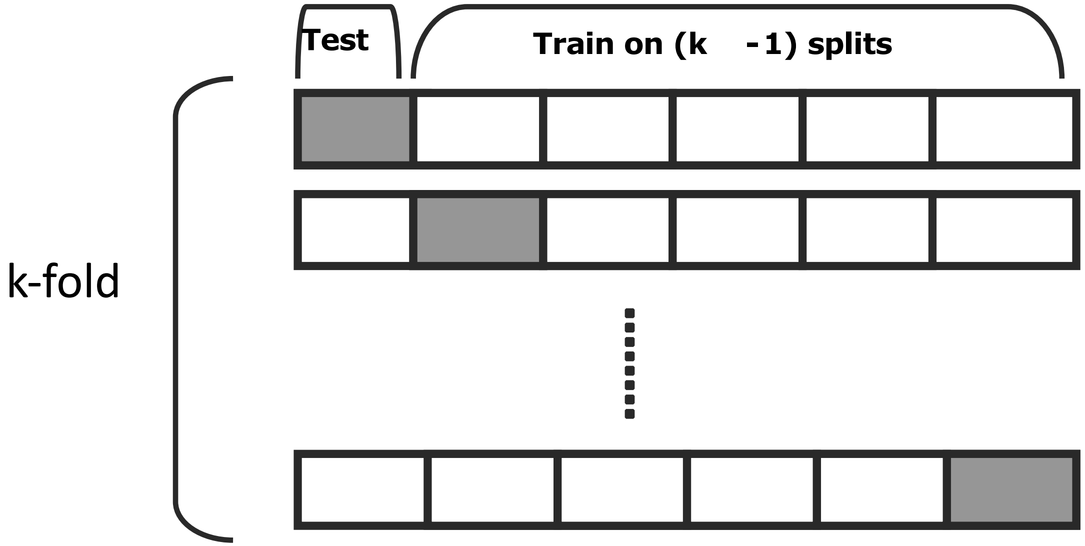

- Let us say we have N data points and k as the index for data points, k=1..N
- Let $(x_k, y_k)$ be the $k^{th}$ record
- Temporarily remove $(x_k, y_k)$ from the dataset
- Train on the remaining N-1 datapoints
- Test the error on $(x_k, y_k)$
- Do this for each k=1..N and report the mean error.

# LOOCV (Leave-one-out Cross Validation)




- Repeat the validation N times, for each of the N data points.
- The validation data is changing each time.

# K-fold cross validation




In 3 fold cross validation, there are 3 runs.
In 5 fold cross validation, there are 5 runs.
In 10 fold cross validation, there are 10 runs.

the error is averaged over all runs

# Tools

- OpenCV
  - http://opencv.org/
- WEKA
  - http://www.cs.waikato.ac.nz/ml/weka/
- RapidMiner
  - http://rapid-i.com/content/view/181/190/