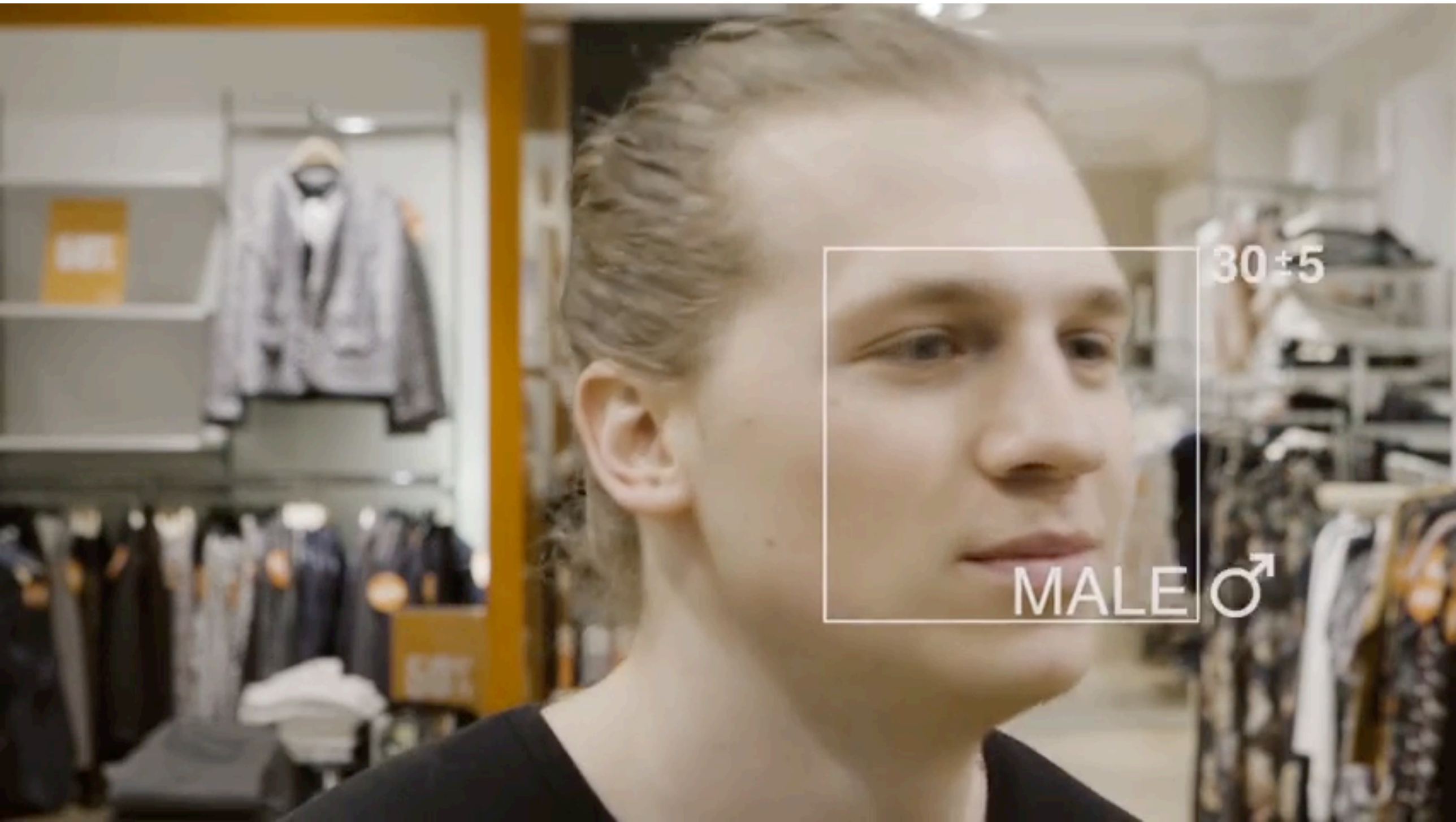
Image Analysis (FMAN20) Lecture 6, 2018

MAGNUS OSKARSSON

Image Analysis - Motivation



Overview – Machine Learning 2

- Machine Learning – supervised vs unsupervised
- More Classification
  - Nearest Neighbor
  - Logistic Regression
  - Support Vector Machines
  - Discriminants
  - Multiclass problems
  - Regression Trees

Machine Learning

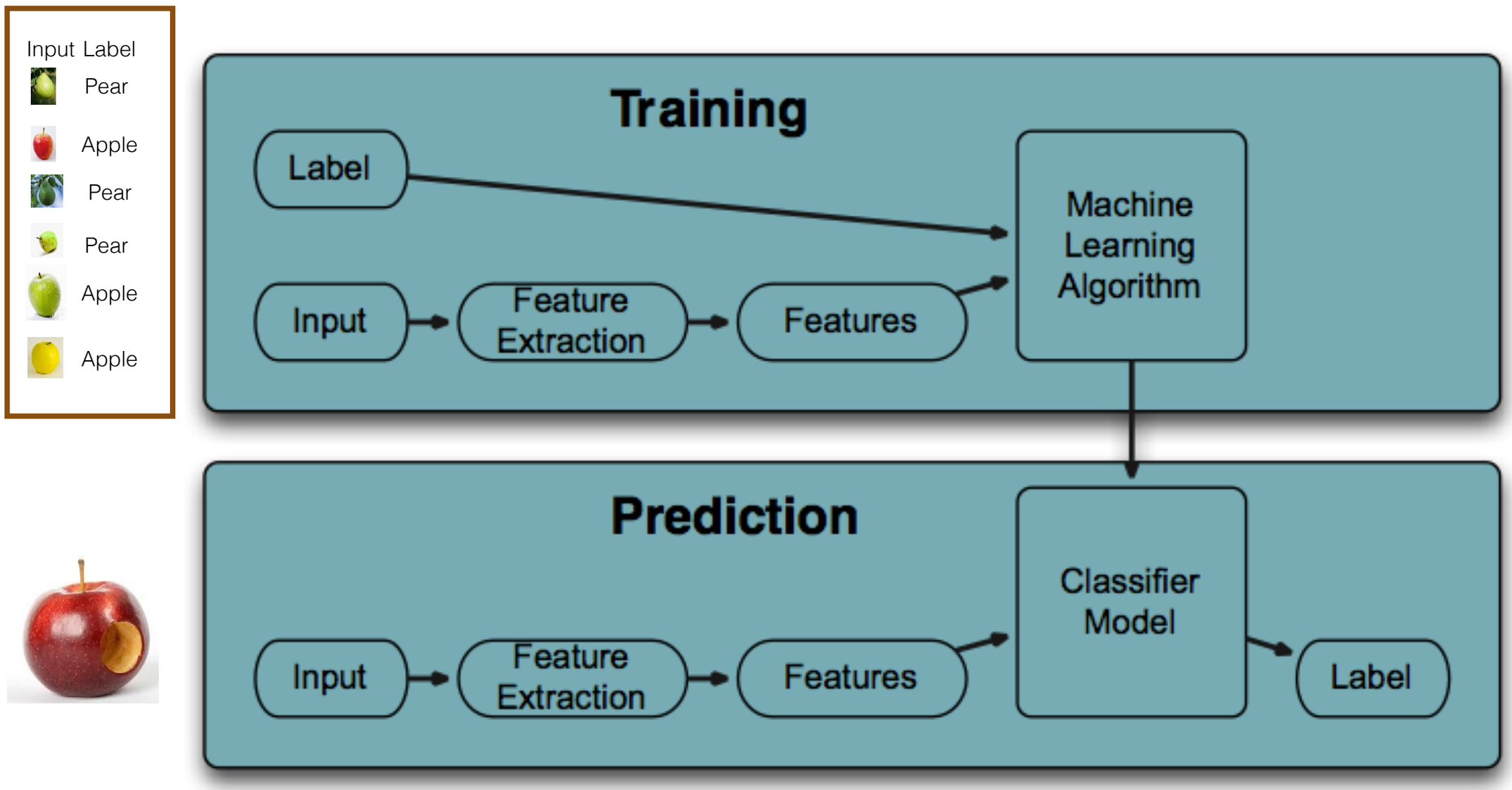
- Supervised learning - classification
  - Training data consists of many pairs (x1,y1), … (xn,yn)
  - Here xi are examples of input and yi are the corresponding correct output
  - The estimated model is used to classify future data f(x)=y
- Unsupervised learning - clustering
  - Training data consists of input data only x1, …, xn
  - After training, the examples are clustered in groups
  - Also future data x can be assigned to groups

Training Dataset

Input	Label
	Pear
	Apple
	Pear
	Pear
	Apple
	Apple

Input	Label
	?

- Machine Learning typically has two phases
- Phase 1 – Training
  - A training dataset is used to estimate model parameters. Store these parameters. Code usually assumes that input are vectors
- Phase 2 – Prediction
  - Once the parameters have been estimated, we can use the model to classify future data



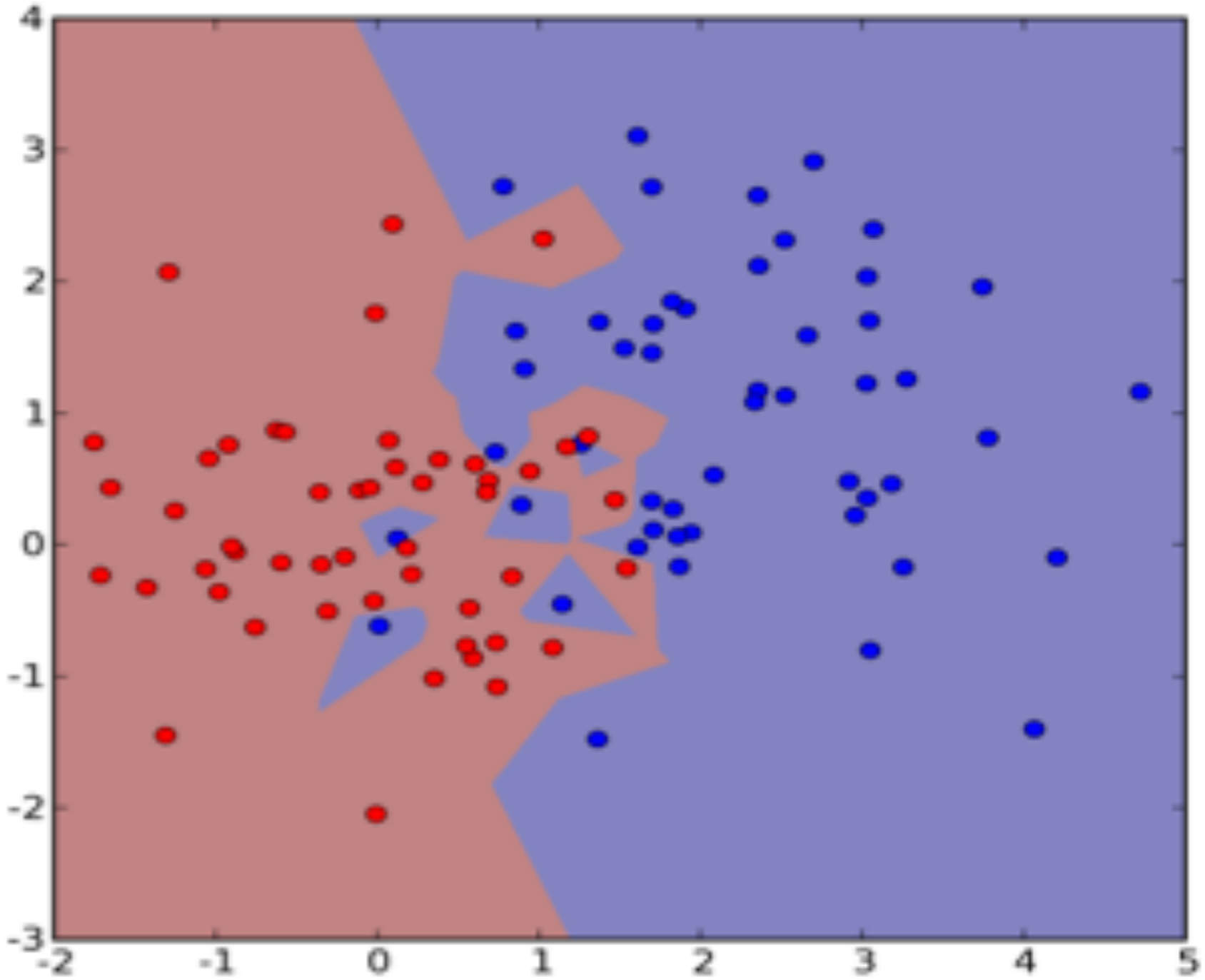
Overview – Machine Learning 2

- Machine Learning
- More Classification
  - **Nearest Neighbor**
  - Logistic Regression
  - Support Vector Machines
  - Discriminants
  - Multiclass problems
  - Regression Trees

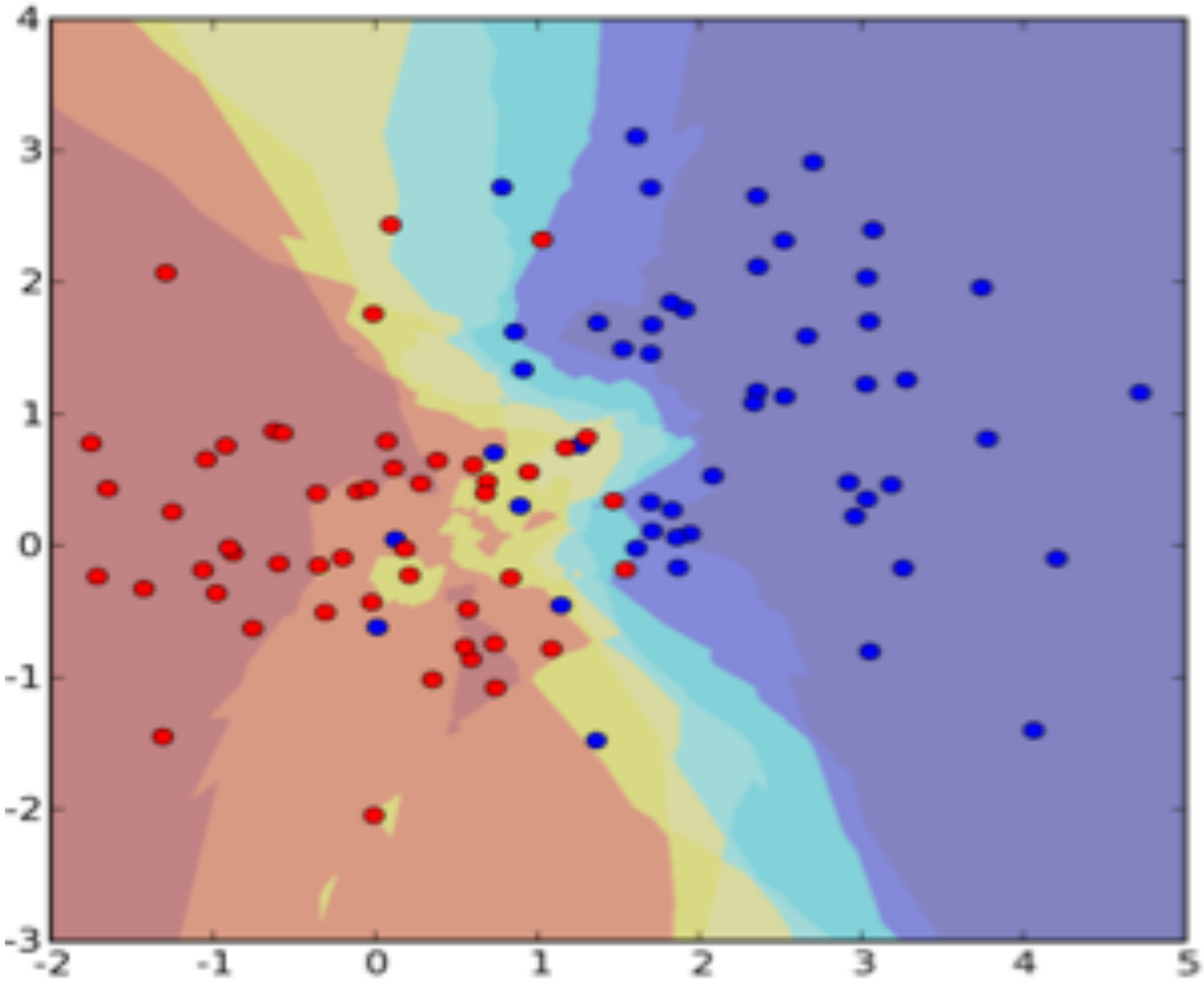
Nearest Neighbor Classification
NN and K-NN

- Classify using training data (x_i, y_i)
- NN: Use the label of the nearest neighbor
- KNN: Use the label of the majority of the k nearest neighbors
- Regression: Use the average of the value of the k nearest neighbors
- Easy to implement and understand
- Can use arbitrary distance functions between images
- Converges to the optimum
- Slow when using lots of data, need to store all training data, not smooth regression

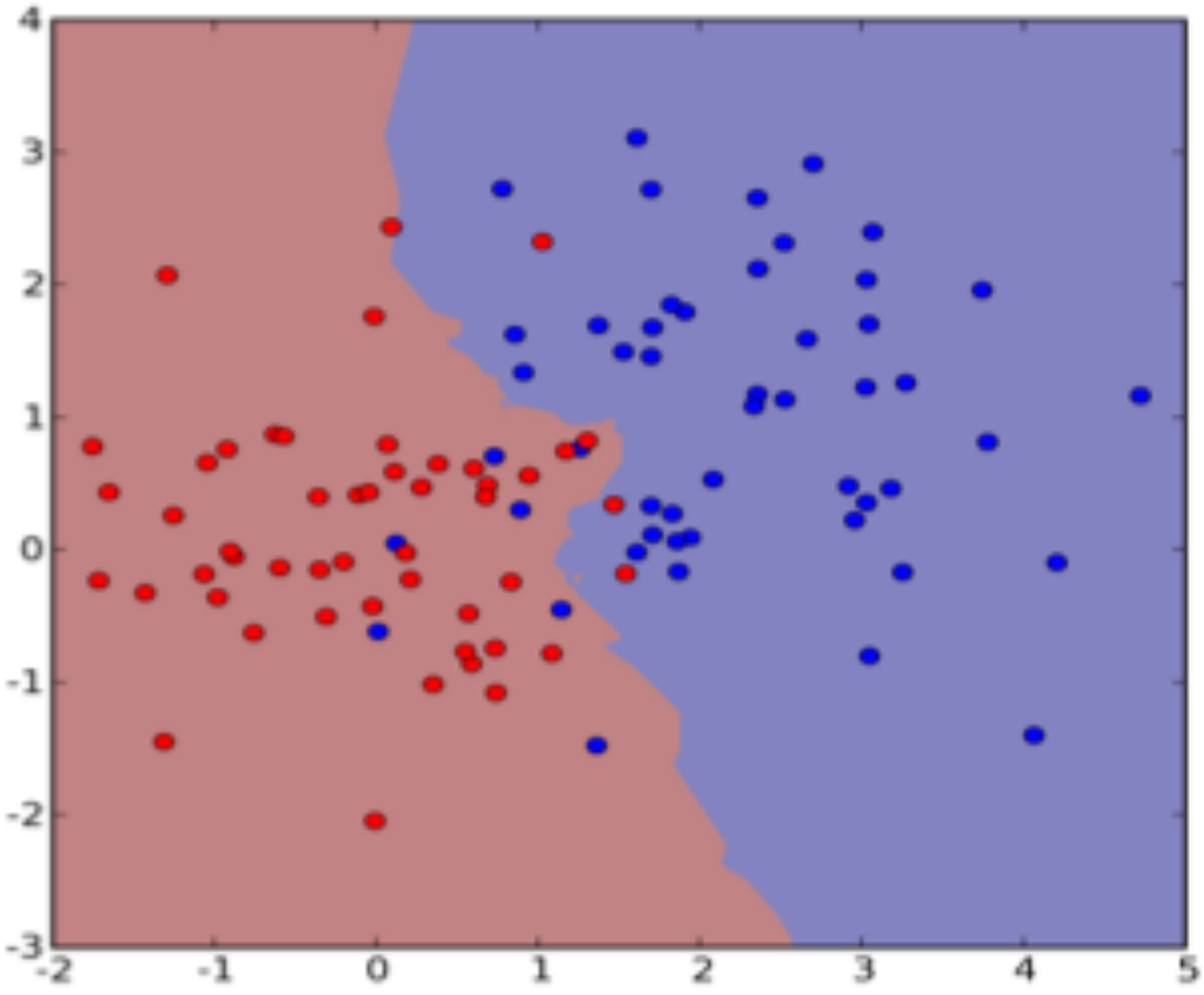
Nearest Neighbor Classification



7 Nearest Neighbor Classification



7 Nearest Neighbor Classification



Nearest Neighbor Classification
NN and K-NN

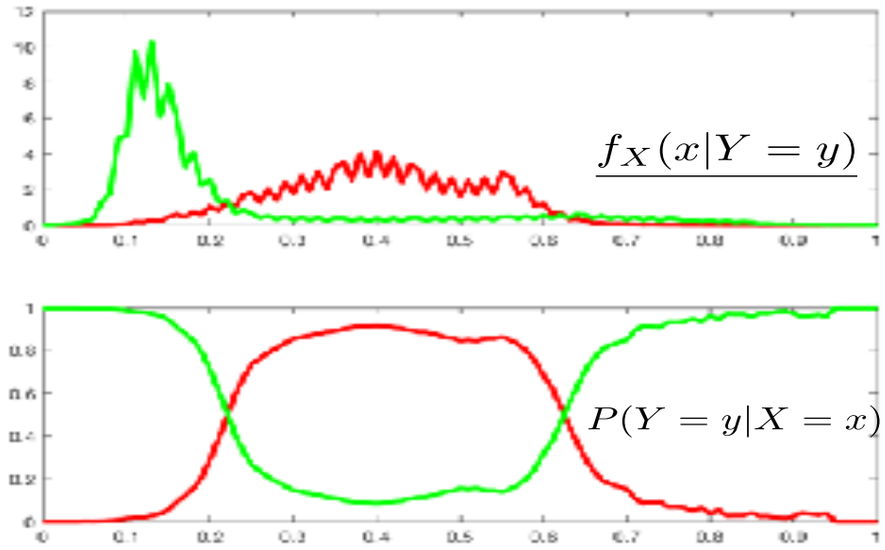
- Training is easy, just store the training data $T = \{(x_1,y_1), \ldots, (x_N,y_N)\}$
- Works in any dimension
- Works for regression also: Use the average of the value of the k nearest neighbors
- Easy to implement and understand
- Can use arbitrary distance functions between images
- Converges to the optimum

- Slow when using lots of data,
- Need to store all training data
- Not smooth regression

Overview – Machine Learning 2

- Machine Learning
- More Classification
  - Nearest Neighbor
  - **Logistic Regression**
  - Support Vector Machines
  - Discriminants
  - Multiclass problems
  - Regression Trees

Logistic regression



- Motivation
- In the end we are only interested in the posterior distribution

$$P(Y = y|X = x)$$

- Why not estimate this instead
- Skip the step of estimating the measurement densities

$$\underline{f_X(x|Y = y)}$$

- Details far away from the transition points are uninteresting (perhaps)
- Notice that the posterior looks like a smoothed step function

Logistic regression

- z = simple function of x, e.g. Linear $z = w^T x + b$
- Output y = smooth threshold of z, for example

$$s(z) = \frac{1}{1 + e^{-z}}$$

- Notice that s(z) looks like a typical P(Y=1 | x) function

$$x \in R^d, w \in R^d, b \in R, f(x) = s(w^T x + b)$$

$$P(Y = 1|x) = \frac{1}{1 + e^{-z}}$$

Derivation

- Estimate parameters

$$P(Y = 1|x) = \frac{1}{1 + e^{-z}}$$

$$P(Y = -1|x) = 1 - \frac{1}{1 + e^{-z}} = \frac{e^{-z}}{1 + e^{-z}} = \frac{1}{e^{z} + 1}$$

- For both cases we have

$$P(Y = y|x) = \frac{1}{1 + e^{-yz}}$$

- Calculate likelihood for training data

$$T = (x_1, y_1), \ldots, (x_n, y_n)$$

Estimate parameters

- Parameters $\theta = (w, b)$

$$P(Y = y|x) = \frac{1}{1 + e^{-yz}}$$

$$T = (x_1, y_1), \ldots, (x_n, y_n)$$

$$log(P) = log(\prod_i P(Y = y_i|x_i, \theta))$$

$$\sum_i \log(\frac{1}{1 + e^{y_i(w^T x_i + b)}})$$

$$\min_w \ \frac{1}{2} w^T w + C \sum_{i=1}^{l} \log(1 + e^{-y_i w^T x_i}) \, .$$

Logistic regression

Standard logistic function

$$s(z) = \frac{1}{1 + e^{-x}}$$

- Linear logistic regression
- Estimate the posterior $P(Y = y|X = x)$
- As linear function followed by standard logistic function

$$s(w^T x + b)$$

- Convex optimization problem

Overview – Machine Learning 2

- Machine Learning
- More Classification
  - Nearest Neighbor
  - Logistic Regression
  - **Support Vector Machines**
  - Discriminants
  - Multiclass problems
  - Regression Trees

Support Vector Machines

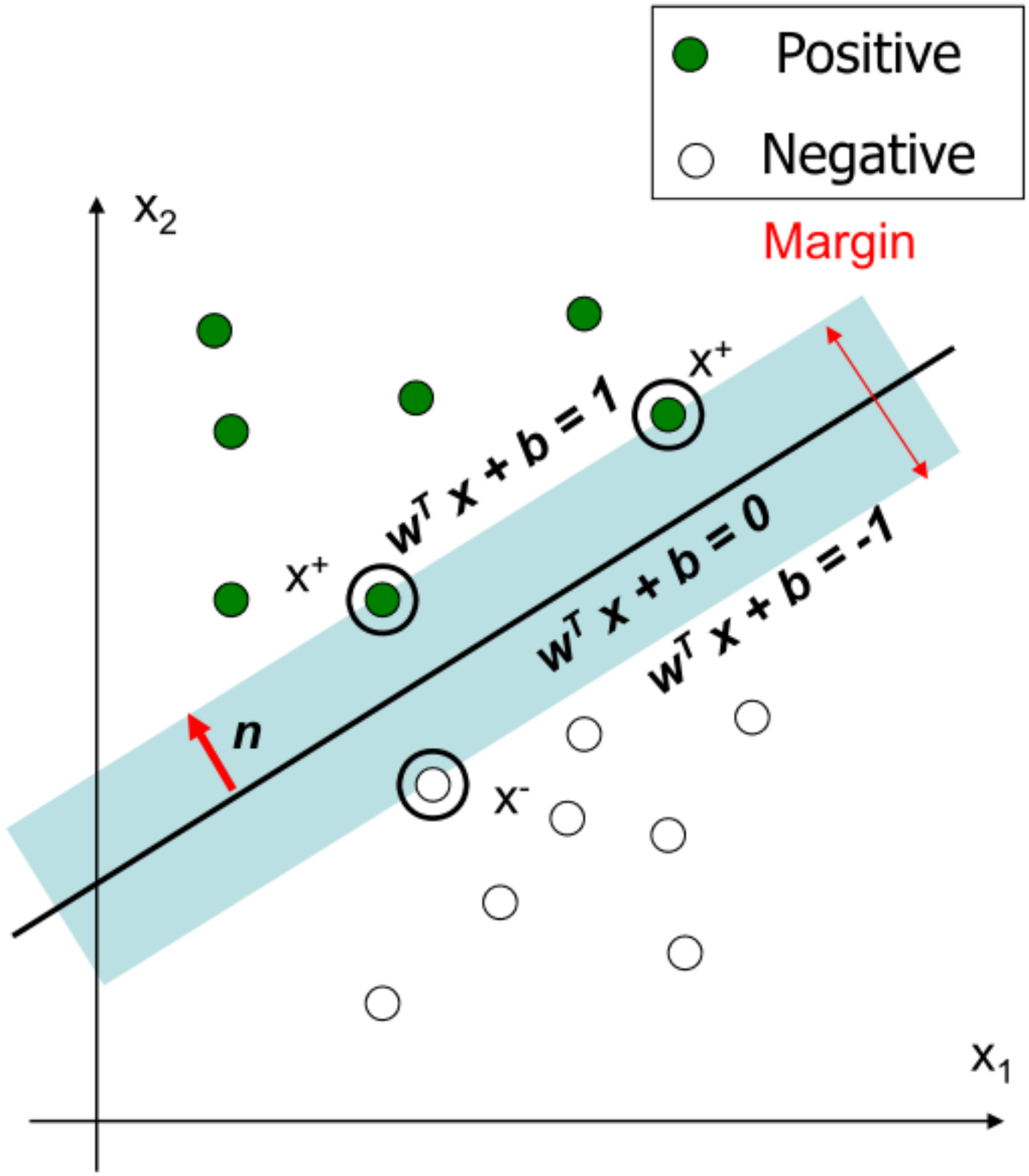
- Formulation:

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2$$

such that

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$$

- Quadratic program with linear constraints



Solving the Optimization Problem

Quadratic programming with linear constraints

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2$$

$$\text{s.t.} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$$

Lagrangian Function

$$\min_{\mathbf{w},b} \max_{\boldsymbol{\alpha}} \; L_p(\mathbf{w},b,\alpha_i) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 \right)$$

$$\text{s.t.} \quad \alpha_i \geq 0$$

The Lagrangian needs to be minimized w.r.t. $\mathbf{w}$, b, and maximized w.r.t α_i

Solving the Optimization Problem

$$\min_{\mathbf{w},b} \max_{\boldsymbol{\alpha}} \; L_p(\mathbf{w},b,\alpha_i) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 \right)$$

$$\text{s.t.} \quad \alpha_i \geq 0$$

$$\frac{\partial L_p}{\partial \mathbf{w}} = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

Solution is an expansion in terms of training examples

$$\frac{\partial L_p}{\partial b} = 0 \quad \Rightarrow \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

Due to strict convexity, $\mathbf{w}$ is unique although $\alpha_i's$ need not be

Solving the Optimization Problem

$$\min_{\mathbf{w},b} \max_{\boldsymbol{\alpha}} \; L_p(\mathbf{w},b,\alpha_i) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 \right)$$

$$\text{s.t.} \quad \alpha_i \geq 0$$

Lagrangian Dual Problem

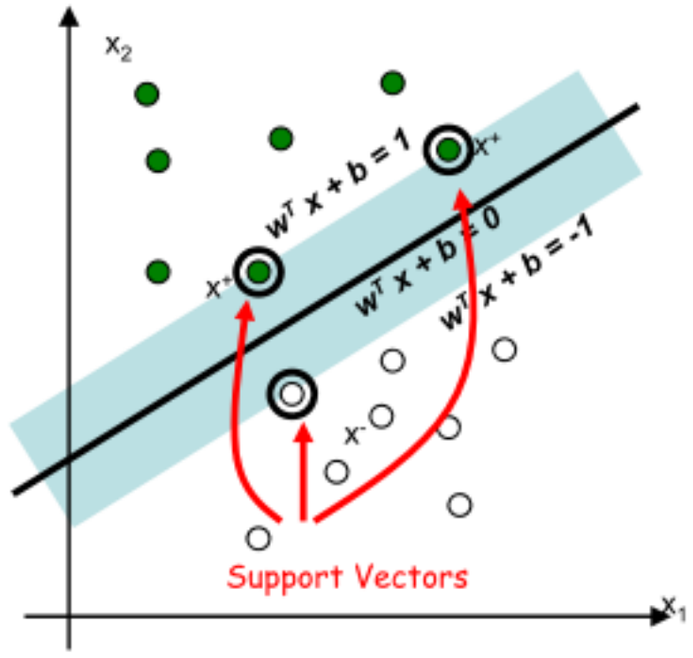
$$\max_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{s.t.} \quad \alpha_i \geq 0 \text{ , and } \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

Solving the Optimization Problem

- From the KKT conditions, we know:

$$\alpha_i \left(y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 \right) = 0$$



- Thus, only support vectors have $\alpha_i \neq 0$

- The solution has the form:

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = \sum_{i \in \mathrm{SV}} \alpha_i y_i \mathbf{x}_i$$

get b from $y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 = 0$,
where $\mathbf{x}_i$ is support vector $b = \dfrac{1}{N_{SV}} \sum_{i=1}^{N_{SV}} (\mathbf{w}^T \mathbf{x}_i - y_i)$

more robust estimate, average over SVs

Solving the Optimization Problem

- The linear discriminant function is:

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = \sum_{i \in \mathrm{SV}} \alpha_i \mathbf{x}_i^T \mathbf{x} + b$$

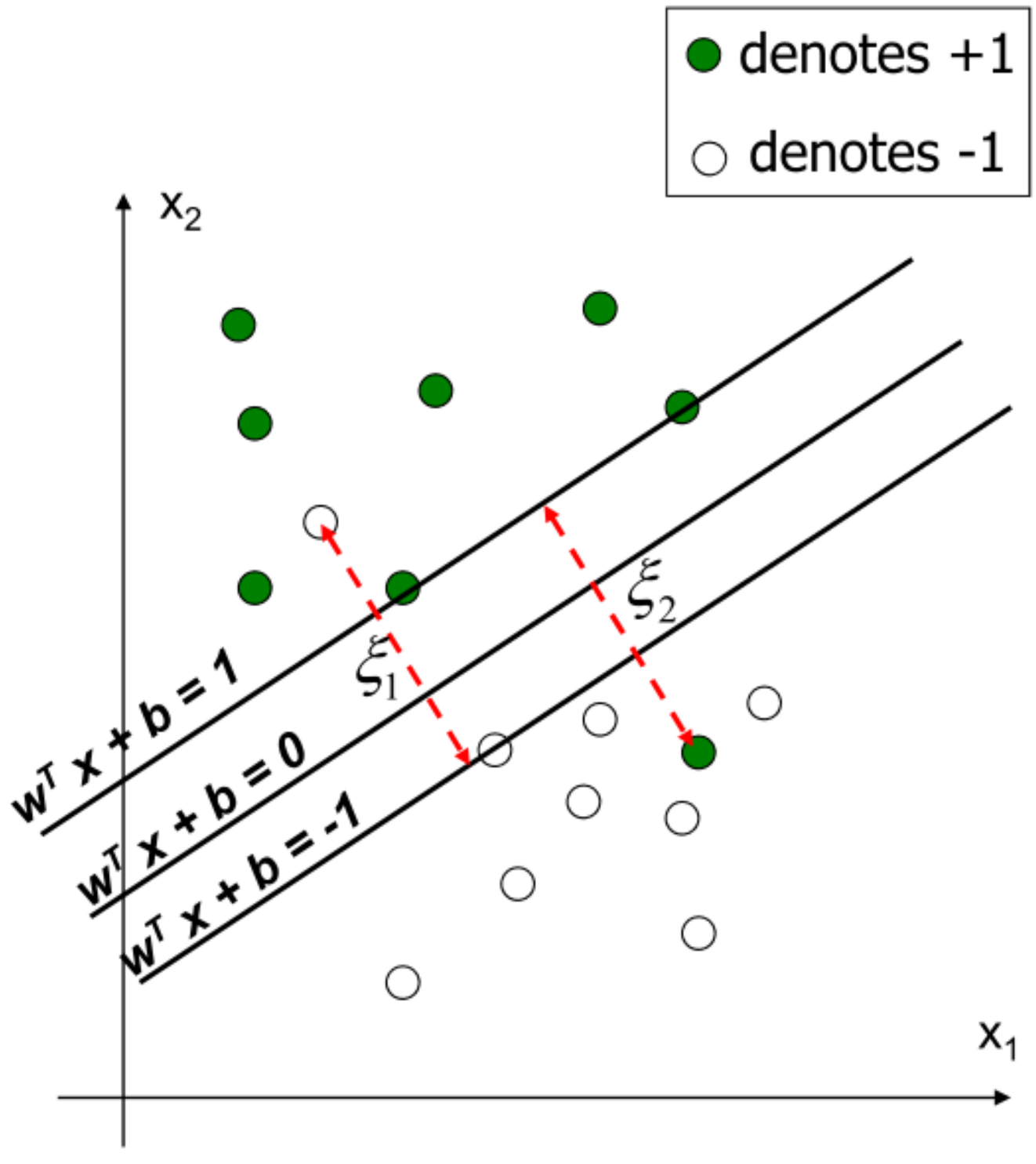
- Relies on a *dot product* between the test point $\mathbf{x}$ and the support vectors $\mathbf{x}_i$

- Solving the optimization problem involved computing the **dot products** $\mathbf{x}_i\mathbf{x}_i^T$ between all pairs of training points

`Soft Margin' Linear Classifier

- What if data is not linear separable due to noise or outliers?

- Slack variables ξ_i can be added to allow for the mis-classification of difficult or noisy data



`Soft Margin' Linear Classifier

- Formulation:

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

such that

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0$$

- for $0 \leq \xi \leq 1$, point is between margin and correct side of hyperplane
- for $\xi > 1$, point is misclassified

- Parameter C can be viewed as a means to control over-fitting
  - small C allows constraints to be easily ignored: *large margin*
  - large C makes constraints hard to ignore: *narrow margin*
  - $C = \infty$ enforces all constraints: *hard margin*

`Soft Margin' Linear Classifier

- Formulation (Lagrangian Dual Problem)

$$\max_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

such that

$$0 \le \alpha_i \le C$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

`Soft Margin' Interpretation (I)

- The constraint $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$ can be written more concisely as

$$y_i g(\mathbf{x}_i) \geq 1 - \xi_i \Leftrightarrow \xi_i = \max(0, 1 - y_i g(\mathbf{x}_i))$$

- Hence we need to solve the learning problem

$$\min_{\mathbf{w},b} \left|\left|\mathbf{w}\right|\right|^2 + C \sum_{i=1}^{N} \max(0, 1 - y_i g(\mathbf{x}_i))$$

`Soft Margin' Interpretation (II)

We need to solve the learning problem

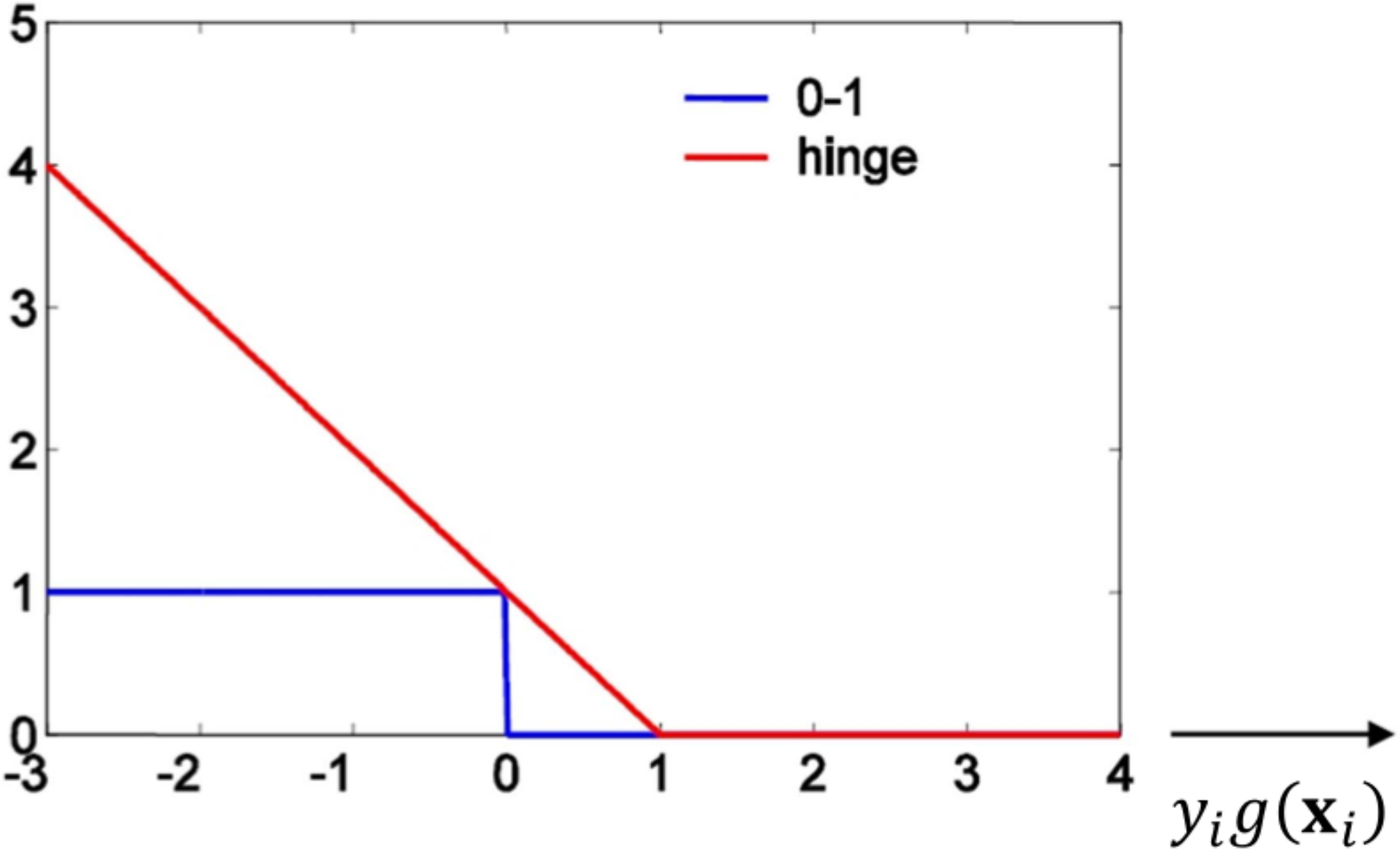
$$\min_{\mathbf{w},b} \left|\left|\mathbf{w}\right|\right|^2 + C\sum_{i=1}^{N} \max(0,1 - y_i g(\mathbf{x}_i))$$

- $y_i g(\mathbf{x}_i) > 1 \Rightarrow$ point is outside margin and does not contribute to loss

- $y_i g(\mathbf{x}_i) = 1 \Rightarrow$ point is on margin and does not contribute to loss (as in hard margin)

- $y_i g(\mathbf{x}_i) < 1 \Rightarrow$ point violates margin constraint and contributes to loss

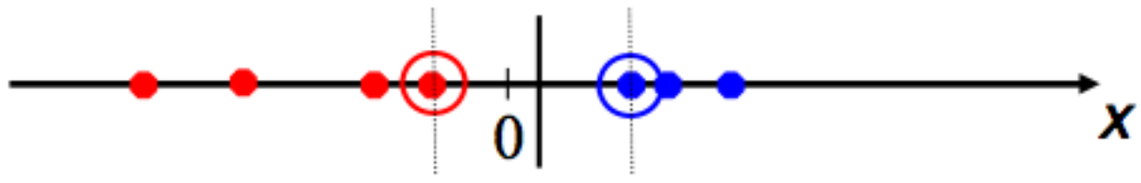
SVM uses Hinge Loss



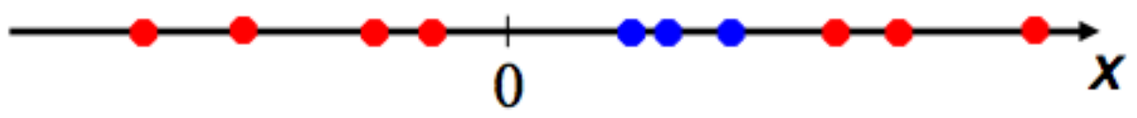
Can be viewed as an approximation to the 0-1 loss

Non-linear SVMs

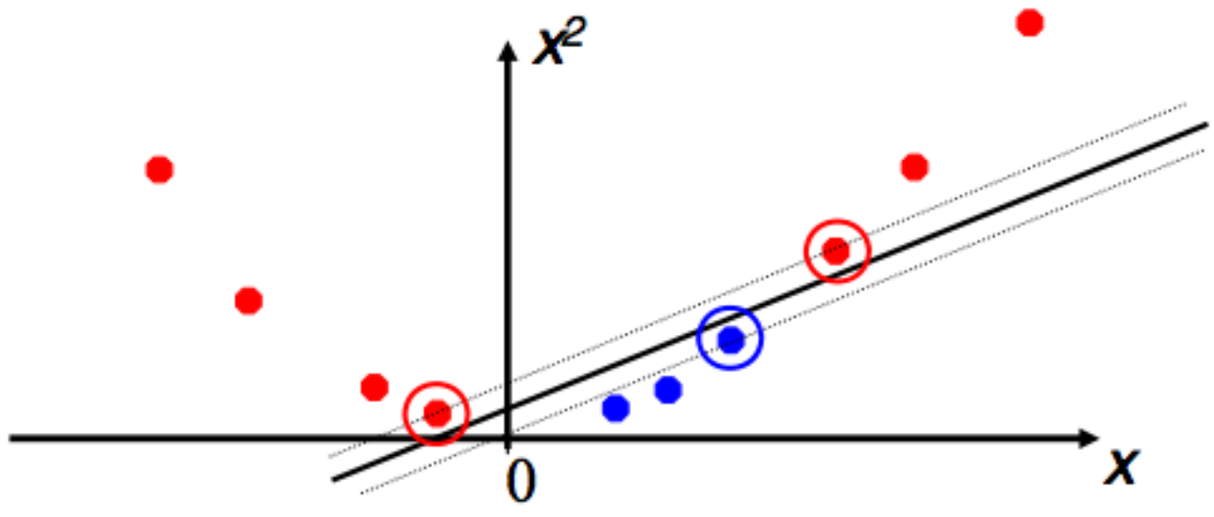
- Datasets that are linearly separable with noise work out great:



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

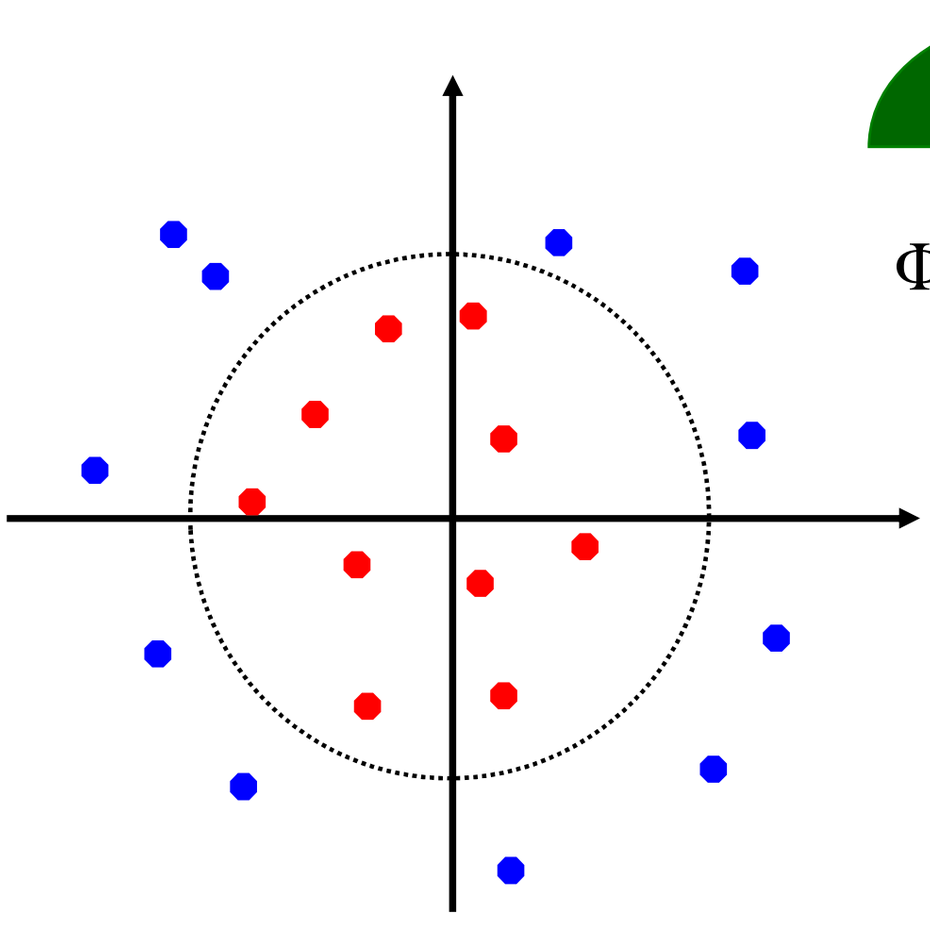


- How about…mapping data to a higher-dimensional space:

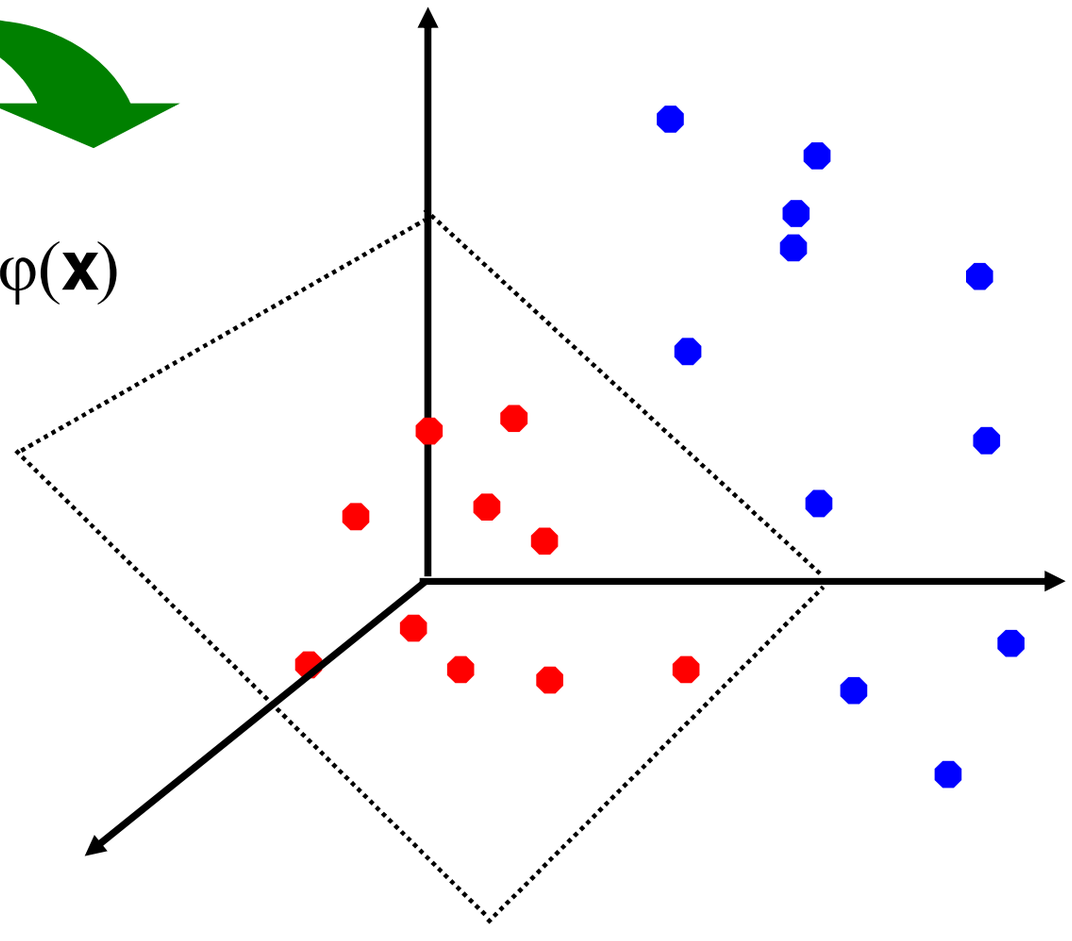


Non-linear SVMs: Feature Space

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



$\Phi: \mathbf{x} \rightarrow \varphi(\mathbf{x})$



How to Use the Feature Space?

- The feature point $\mathbf{z} = \phi(\mathbf{x})$ corresponding to an input point $\mathbf{x}$ is called the image (or the lifting) of $\mathbf{x}$; the input point $\mathbf{x}$, if any, corresponding to a given feature vector $\mathbf{z}$ is called the pre-image of $\mathbf{z}$

- The naive way to use a feature space is to explicitly compute the image of every training and testing point, and run algorithm fully in feature space

- Two potential problems
  - The feature space may be very high dimensional or infinite dimensional, so direct (explicit) calculations in such feature space may not be practical, or even possible
  - We may sometimes want to map back an answer from feature space to the input space. This is called the pre-image problem. For some feature maps (kernels), analytical expressions are available, but in most other cases some form of (local) optimization may be necessary

Nonlinear SVMs: The Kernel Trick

- With this mapping, our discriminant function is now:

$$g(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b = \sum_{i \in \mathrm{SV}} \alpha_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) + b$$

- No need to know this mapping explicitly, because we only use the *dot product* of feature vectors both in training and in testing

- A *kernel function* is defined as a function that corresponds to a dot product of two feature vectors in some expanded feature space:

$$K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

Positive Definite Kernels

- **Gram Matrix.** Given a function $k: \mathrm{X}^2 \to \mathbf{R}$ (or $\mathbf{C}$), and patterns $x_1, \ldots, x_m \in \mathrm{X}$, the m x m matrix K with elements $K_{ij} := k(x_i, x_j)$ is called the Gram matrix (or kernel matrix) of k w.r.t $x_1, \ldots, x_m$.

- **Positive definite kernel.** A complex m x m matrix K satisfying $\sum_{ij} c_i \bar{c}_j K_{ij} \geq 0, \forall\, c_i \in \mathbf{C}$ is called positive definite. Similarly, a real symmetric m x m matrix K satisfying the above for all $c_i \in \mathbf{R}$ is called positive definite.

positive definite kernels ≡Mercer kernels ≡reproducing kernels ≡admissible kernels ≡ support vector kernels ≡covariance functions

Examples of Kernels

Examples of commonly-used kernel functions:

- Linear kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
- Polynomial kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p$
- Gaussian (Radial-Basis Function (RBF)) kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2}{2\sigma^2}\right)$$

- Sigmoid:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta_0 \mathbf{x}_i^T \mathbf{x}_j + \beta_1)$$

Generality of Kernel Trick

- Given an algorithm expressed in terms of a positive-definite kernel k, we can construct an alternative algorithm by replacing k with another positive-definite kernel $\tilde{k}$
- This is not limited to only cases when k is a dot product in the input domain
- Any algorithm that only depends on dot products (i.e. is rotationally invariant) can be kernelized
- Kernels are defined on general sets (rather than just dot product spaces!) and their use leads to an embedding of general data types in linear spaces

Nonlinear SVM: Optimization

- Formulation (Lagrangian Dual Problem)

$$\max_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

such that

$$0 \le \alpha_i \le C$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

- The solution of the discriminant function is

$$g(\mathbf{x}) = \sum_{i \in \mathrm{SV}} \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b$$

Support Vector Machine: Algorithm

1. Choose a kernel function

2. Choose a value for C

3. Solve the quadratic programming problem
   (many software packages available, e.g. libsvm)

4. Construct the discriminant function from the support vectors

Support Vector Machines

- SVM Applet demo
- http://cs.stanford.edu/people/karpathy/svmjs/demo/
- Other demos
- http://cs.stanford.edu/people/karpathy/convnetjs/

Properties of Kernels

- Kernels are symmetric in their arguments:

$$K(\mathbf{x}_1, \mathbf{x}_2) = K(\mathbf{x}_2, \mathbf{x}_1)$$

- They are positive valued for any inputs: $K(\mathbf{x}_1, \mathbf{x}_2) \geq 0$

- The Cauchy-Schwartz inequality holds:

$$K^2(\mathbf{x}_1, \mathbf{x}_2) \leq K(\mathbf{x}_1, \mathbf{x}_1)K(\mathbf{x}_2, \mathbf{x}_2)$$

- Technically, to use a function as a kernel, it must satisfy Mercer's conditions for a positive-definite operator

- The intuition is easy to grasp for finite spaces
  - Discretize $\mathbf{x}$ space as densely as desired into buckets $\mathbf{x}_i$
  - Between each two cells $\mathbf{x}_i, \mathbf{x}_j$, compute the kernel function, and write these values as a (symmetric) matrix $M_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$
  - If the matrix is positive definite, the kernel is OK

Kernel Closure Rules

Very useful for designing new kernels from existing kernels

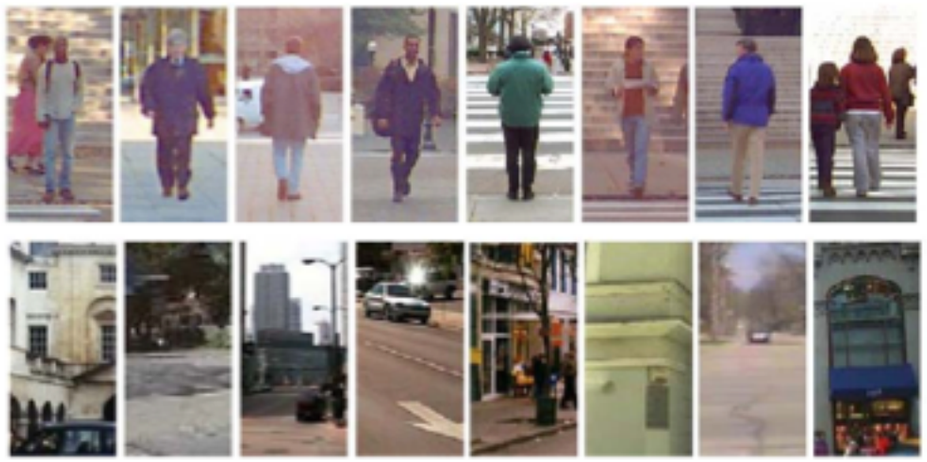
– The sum of any two kernels is a kernel

– The product of any two kernels is a kernel

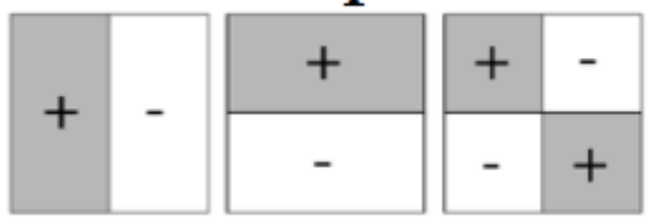
– A kernel plus a constant is a kernel

– A scalar times a kernel is a kernel

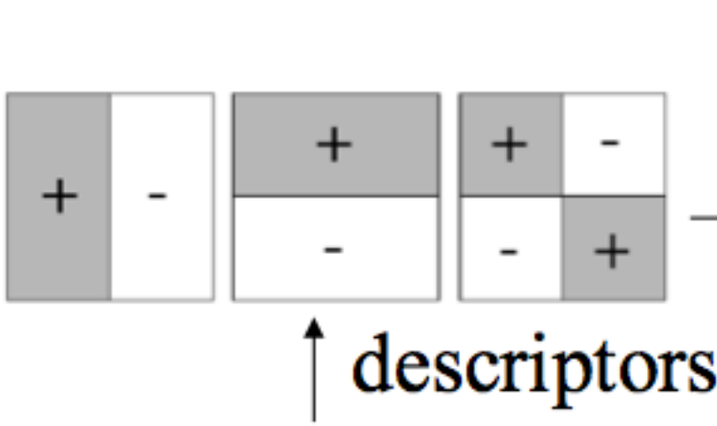
Support Vector Machine Detector



Training set

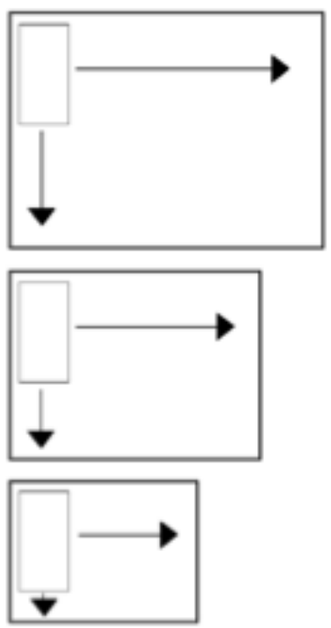


training



descriptors

Test image



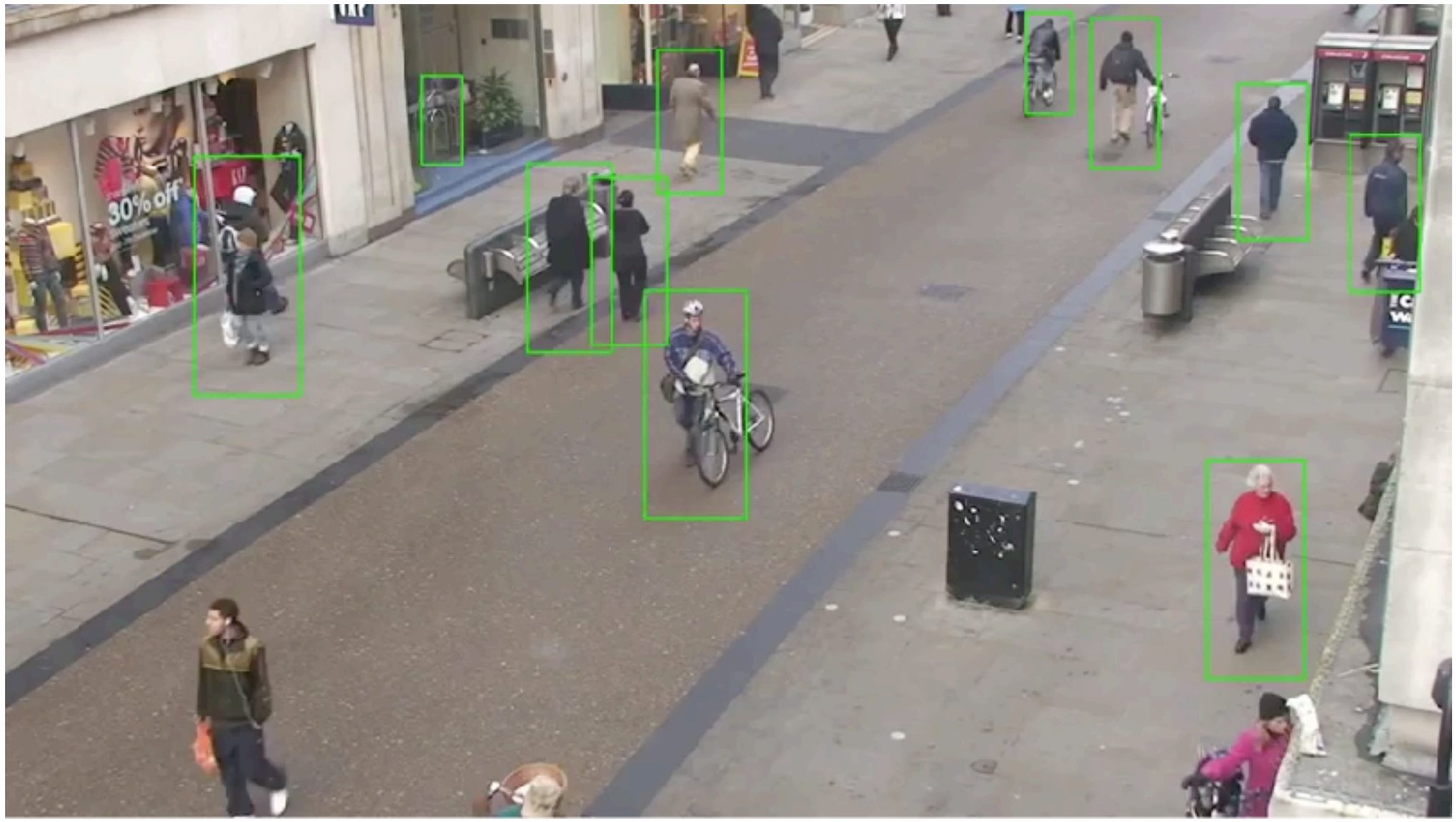
Multi-scale search

test

Support vector machine

results

SVM – Pedestrian detection



Overview – Machine Learning 2

- Machine Learning
- More Classification
  - Nearest Neighbor
  - Logistic Regression
  - Support Vector Machines
  - **Discriminants**
  - **Multiclass problems**
  - Regression Trees

Multi-Class Decision Problems

- The goal in classification is to take an input vector $\mathbf{x}$ and to assign it to one of K discrete classes C_k where $k = 1, \ldots, K$.

- The classes are taken to be disjoint, so that each input is assigned to one and only one class (N.B. This does not necessarily mean that data is separable!)

- The input space is thereby divided into decision regions whose boundaries are called decision boundaries or decision surfaces.

- We consider linear models for classification, by which we mean that the decision surfaces are linear functions of the input vector x and hence are defined by $(D-1)-$ dimensional hyperplanes within the $D-$ dimensional input space.

- Data sets whose classes can be separated exactly by linear decision surfaces are said to be linearly separable.

Linear Classification

- Classification is intrinsically non-linear because of the training constraints that ***place non-identical inputs in the same class***
- Differences in the input vector sometimes causes 0 change in the answer
- The *presence of more than two classes* makes formulation harder (see next)
- Linear classification means that the adaptive part $\mathbf{w}$ is linear
  - The adaptive part is cascaded with a fixed non-linearity f
  - It may also be preceded by a fixed non-linearity ϕ when nonlinear basis functions are used

fixed non-linear functions

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + w_0, \qquad c = f(y(\mathbf{x}))$$

adaptive linear parameters decision

Since the decision boundary is $y(\mathbf{x}) = constant$, for $\phi(\mathbf{x}) = \mathbf{x}$, the decision surfaces are linear functions of $\mathbf{x}$ even if f is non-linear!

Approach 1: Discriminant Function

- Use discriminant functions directly, and do not compute probabilities
- Convert the input vector into one or more real values so that a simple process (threshold, or majority vote) can be applied to assign the input to the class
- The real values should be chosen to maximize the useable information about the class label present in the real value
- Given discriminant functions $f_1(\mathbf{x}), \ldots, f_K(\mathbf{x})$

  Classify $\mathbf{x}$ as class C_k, **iff** $f_k(\mathbf{x}) > f_j(\mathbf{x}), \forall j \neq k$

Approach 2: Class-conditional Probabilities

- Infer conditional class probabilities $p(C_k|\mathbf{x})$
- Use conditional distribution to make optimal decisions, e.g. by minimizing some loss function
- Example, 2 classes

$$p(C_1|\mathbf{x}) = \sigma(\mathbf{w}^T\phi), \qquad p(C_2|\mathbf{x}) = 1 - p(C_1|\mathbf{x})$$

$$\text{where } \sigma(a) = \frac{1}{1+\exp(-a)}$$

Approach 3: Class Generative Model

- Compare the probability of the input under separate, class-specific, generative models
- Model both the class conditional densities $p(\mathbf{x}|C_k)$, and the prior class probabilities $p(C_k)$
- Compute posterior using Bayes' theorem

class conditional density class prior

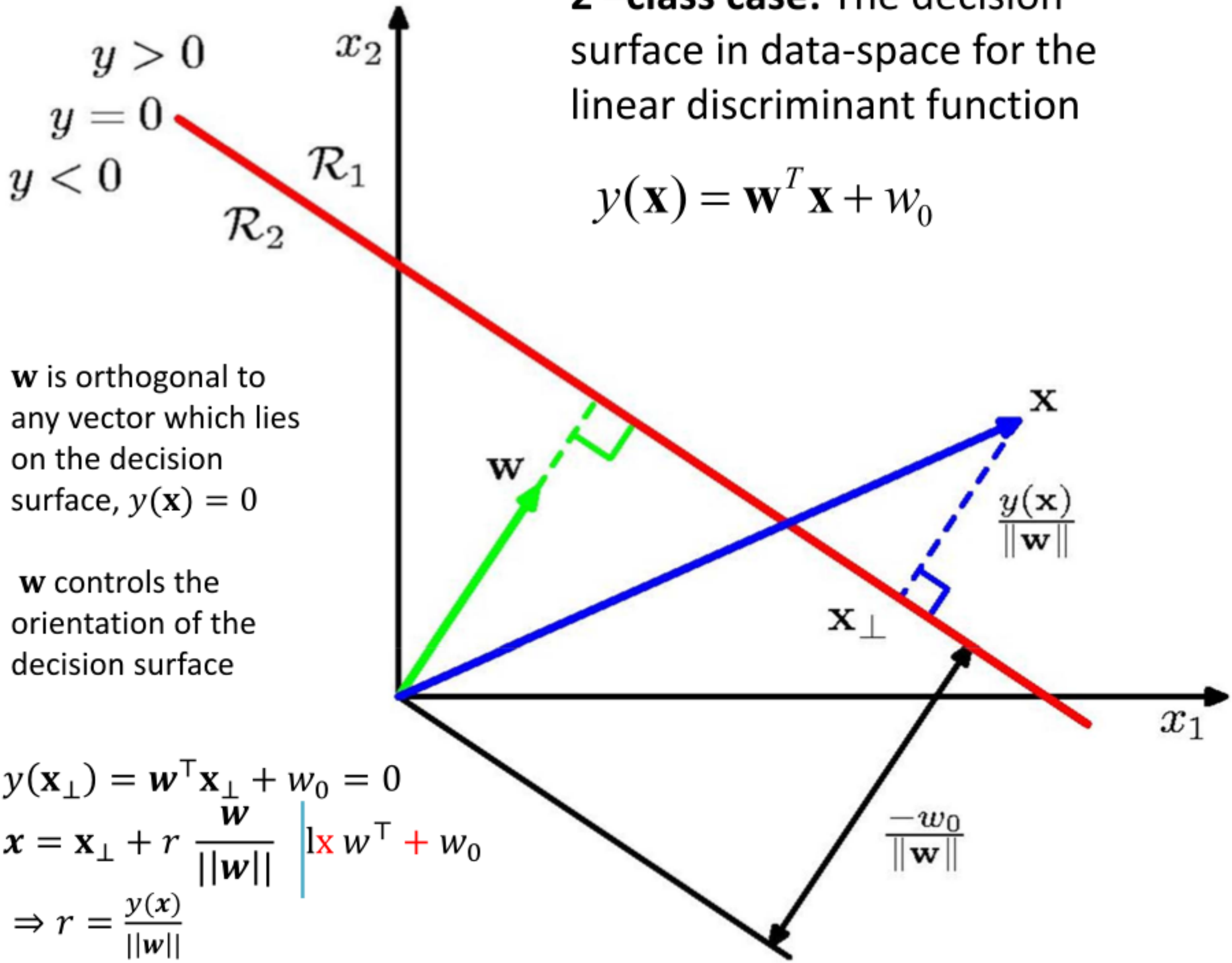
$$p(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)p(C_k)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|C_k)p(C_k)}{\sum_j p(\mathbf{x}|C_j)p(C_j)}$$

posterior for class

- Example: fit a multivariate Gaussian to the input vectors corresponding to each class, model class prior probabilities by training data frequency counts, and see which Gaussian makes a test data vector most probable using Bayes' theorem

2 –class case: The decision surface in data-space for the linear discriminant function

$$y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$$



w is orthogonal to any vector which lies on the decision surface, $y(\mathbf{x}) = 0$

w controls the orientation of the decision surface

$$y(\mathbf{x}_\perp) = \boldsymbol{w}^\top\mathbf{x}_\perp + w_0 = 0$$

$$\boldsymbol{x} = \mathbf{x}_\perp + r\,\frac{\boldsymbol{w}}{||\boldsymbol{w}||} \quad \text{lx } w^\top + w_0$$

$$\Rightarrow r = \frac{y(x)}{||\boldsymbol{w}||}$$

Represent Target Values: *Binary* vs. *Multiclass*

- **Two classes (*N=2*):** typically use a single real valued output that has target values of 1 for the *positive class* and 0 (or -1) for the *negative class*
  - For probabilistic class labels, the target can be the probability of the positive class and the output of the model can be the probability the model assigns to the positive class

- **For the multiclass (*N>2*)**, we use a vector of *N* target values containing a single 1 for the correct class and zeros elsewhere
  - For probabilistic labels we can then use a vector of class probabilities as the target vector

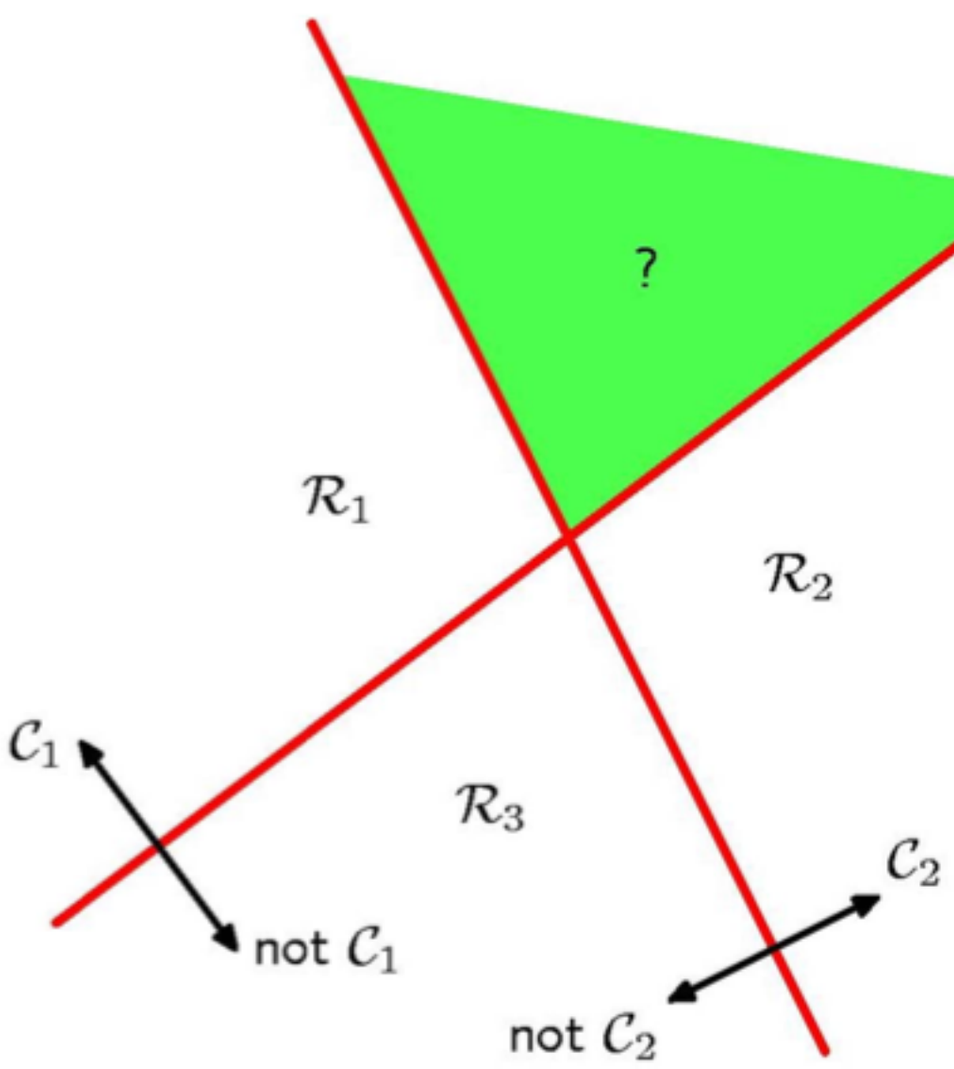
Discriminant Functions for Multiclass

- One possibility is to use N **binary** (2-way) **discriminants**
  - Each function separates one class from the rest.

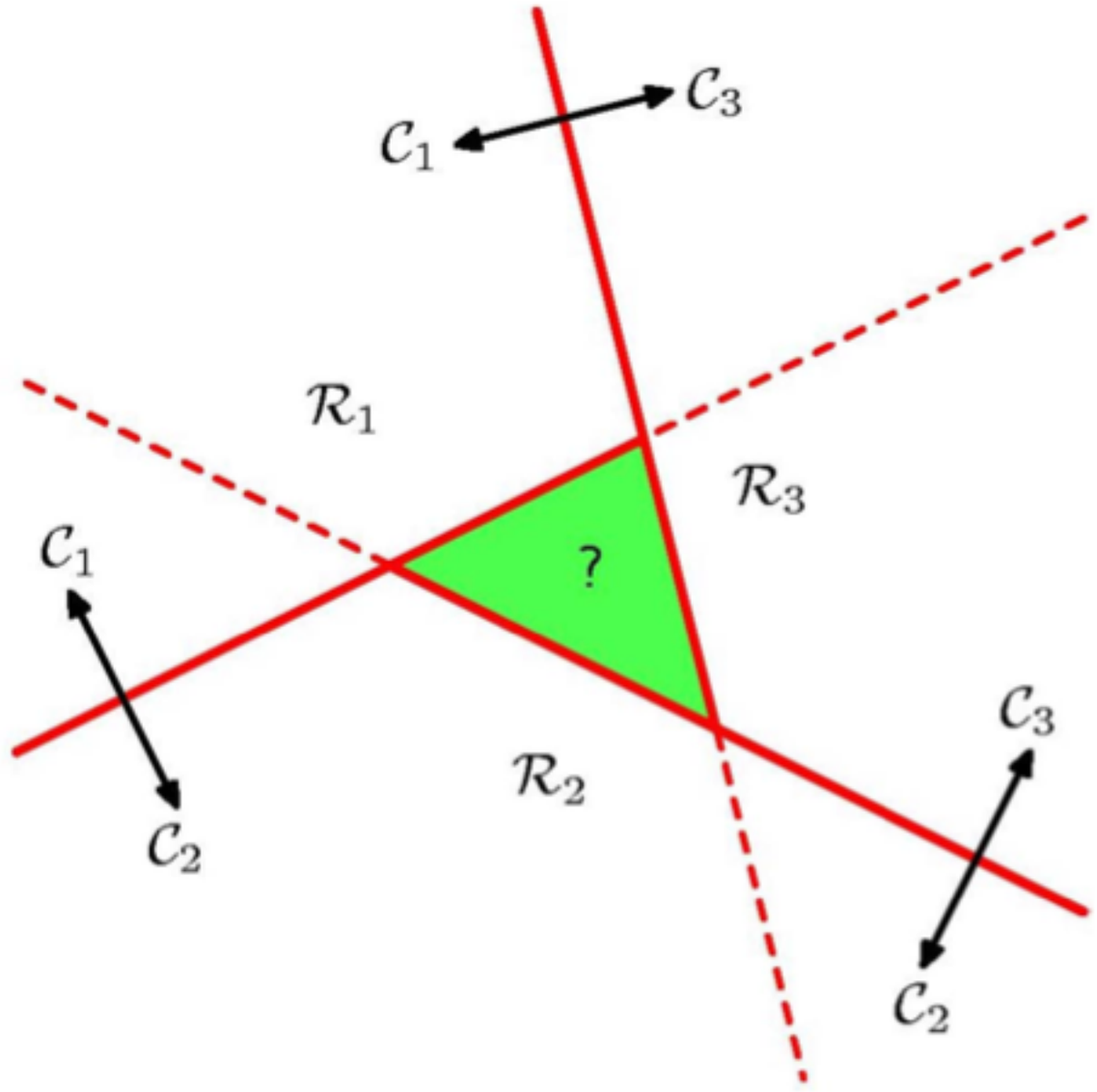
- Another possibility is to use $\frac{N(N-1)}{2}$ **binary** (2-way) **discriminants**
  - Each function discriminates between two specific classes. We have 1 discriminant for each class pair. For decision, we can take a majority vote

- Both methods have ambiguities

Problems with Multi-class Discriminant Functions Constructed from Binary Classifiers



1 - vs.- all



1 vs. 1

If we base our decision on binary classifiers, we can encounter ambiguities

Simple Solution

Use N discriminant functions, $y_i, y_j, y_k, \ldots$, and take the max over their response

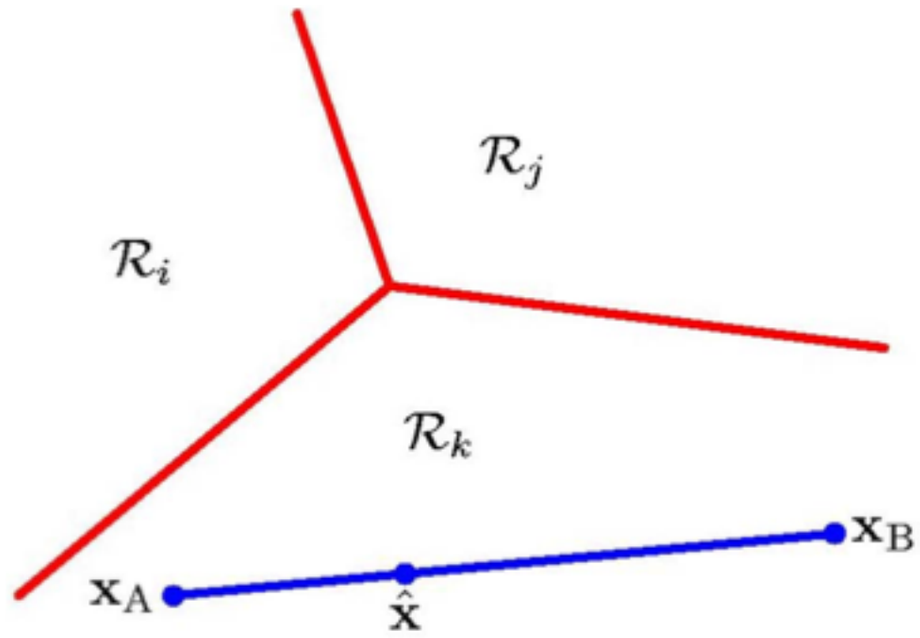
- Consider linear discriminants y

$$y_k(\mathbf{x}) = w_k^{\mathrm{T}}\mathbf{x} + w_{k0}$$

- The decision boundary between class k and j is given by the $D-1$ hyperplane

$$(w_k^{\mathrm{T}} - w_j^{\mathrm{T}})\mathbf{x} + (w_{k0} - w_{j0}) = 0$$

(analogous geometrical properties as in the two class case apply)



In this linear case the decision regions are convex (max will give consistent results)

$\mathbf{x}_A, \mathbf{x}_B \in R_K,\ \hat{\mathbf{x}} = \lambda\mathbf{x}_A + (1-\lambda)\mathbf{x}_B, 0 \le \lambda \le 1$

From the linearity of $y \Rightarrow y_k(\hat{\mathbf{x}}) = \lambda y_k(\mathbf{x}_A) + (1-\lambda)y_k(\mathbf{x}_B)$

But $y_k(\mathbf{x}_A) > y_j(\mathbf{x}_A)$ and $y_k(\mathbf{x}_B) > y_j(\mathbf{x}_B)$ $\quad \forall j \neq k \Rightarrow y_k(\hat{\mathbf{x}}) > y_j(\hat{\mathbf{x}})$

$\Rightarrow \hat{\mathbf{x}}$ also lies inside R_k

Hence R_k is convex

Fisher's Linear Discriminant

- We can view classification in terms of dimensionality reduction
- A simple linear discriminant function is a projection of the $D-$ dimensional data $\mathbf{x}$ down to 1 dimension

$$\underline{\text{Project}}: \; y = \boldsymbol{w}^T \mathbf{x};$$

$$\underline{\text{Classify}}: \; \text{if } \; y \geq -w_0 \text{ then } C_1 \text{ else } C_2$$

- However projection results in loss of information. Classes well separated in the original input space may strongly overlap in 1d
- However, by adjusting the projection (weight vector $\mathbf{w}$) we can aim at the best separation among classes. But what do we mean by *best separation*?

Fisher's View of Class Separation (I)

- Consider a two-class problem. There are N_1 points of class C_1 and N_2 of class C_2
- The simplest measure of class separation when projected onto $\mathbf{w}$ is the one between the projected class means. This suggests choosing $\mathbf{w}$ so to maximize

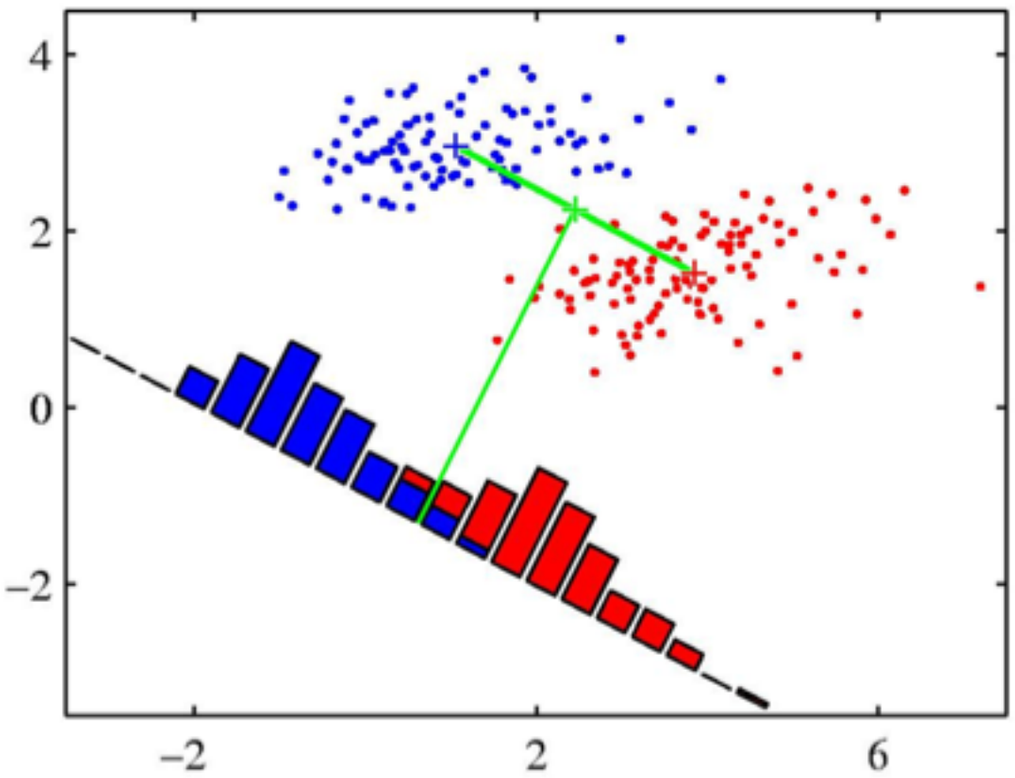
$$m_2 - m_1 = \mathbf{w}^{\mathbf{T}}(\mathbf{m}_2 - \mathbf{m}_1), m_k = \mathbf{w}^{\mathbf{T}}\mathbf{m}_{\mathrm{k}}, \mathbf{m}_1 = \frac{1}{N_1}\sum_{n\in C_1}\mathbf{x}_{\boldsymbol{n}}, \mathbf{m}_2 = \frac{1}{N_2}\sum_{n\in C_2}\mathbf{x}_{\boldsymbol{n}}$$

- This can be made arbitrarily large by increasing $\left|\left|\mathbf{w}\right|\right|$. We could handle this by imposing unit norm constraints using Lagrange multipliers. We get

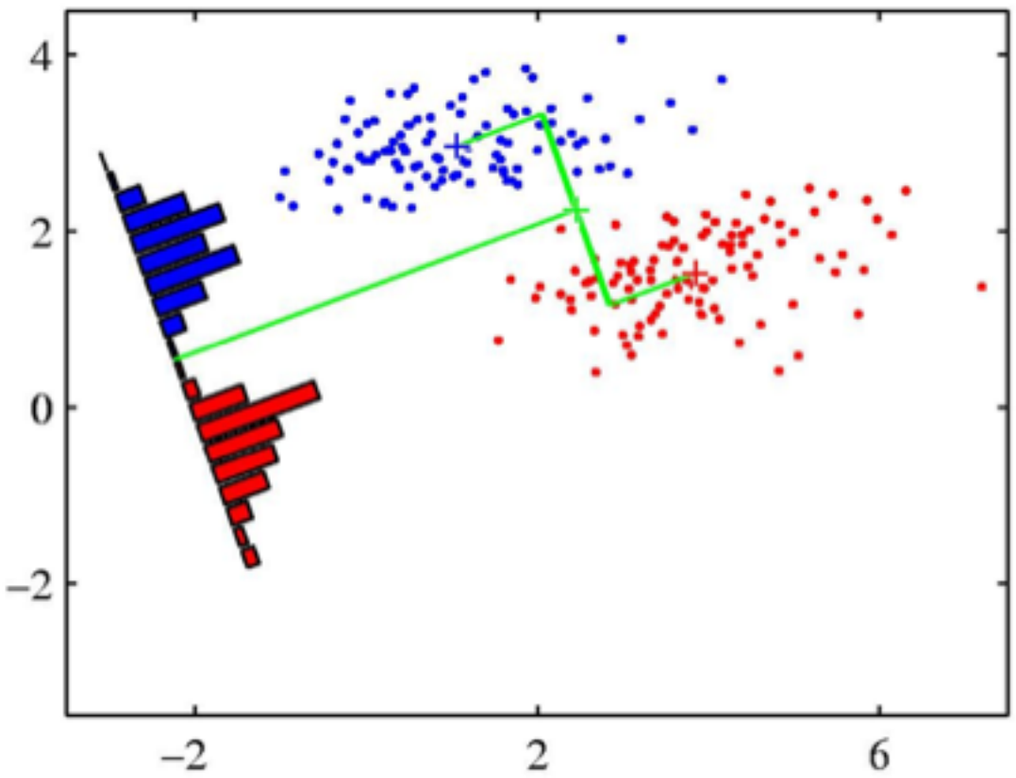
$$\max_{\mathbf{w}}\mathbf{w}^{\mathbf{T}}(\mathbf{m}_2 - \mathbf{m}_1),\ \text{s.th.}\ \left|\left|\mathbf{w}\right|\right| = \mathbf{1} \Rightarrow \mathbf{w} \propto \mathbf{m}_2 - \mathbf{m}_1$$

- However, still, if the main direction of variance in each class is not orthogonal to the direction between means, we will not get good separation (see next slide). This is often due to strongly non-diagonal class distribution covariances

Advantage of using Fisher's Criterion



Although well separated in the original 2d space, when projected onto the 1d (line) joining their means, the classes are not well separated



Fisher chooses a direction that makes the projected classes much tighter, *even though their projected means are less far apart*

Fisher's View of Class Separation (II)

Fisher: maximize a function that gives a large separation between the projected class means, while also giving a small variance within each class, thereby minimizing class overlap

- Choose direction maximizing the **ratio** of **between class** variance to **within class** variance
- This is the direction in which the projected points contain the most information about class membership (under Gaussian assumptions)

Fisher's Linear Discriminant

- We seek a linear transformation that is best for discrimination

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

- Projection onto the vector separating the class seems right, initially

$$\mathbf{w} \propto \mathbf{m}_2 - \mathbf{m}_1$$

- But we also want small variance within each class

$$s_k^2 = \sum_{n \in C_k} (y_n - m_k)^2 , \qquad m_k = \mathbf{w}^{\mathbf{T}} \mathbf{m}_{\mathrm{k}}$$

- Fisher's objective function

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

← Between class

← Within class

Fisher's Linear Discriminant Derivations

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

where

$$\mathbf{S}_B = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^T$$

$$\mathbf{S}_W = \sum_{n \in C_1} (\mathbf{x}_n - \mathbf{m}_1)(\mathbf{x}_n - \mathbf{m}_1)^T + \sum_{n \in C_2} (\mathbf{x}_n - \mathbf{m}_2)(\mathbf{x}_n - \mathbf{m}_2)^T$$

scalar scalar

$$\frac{\boldsymbol{d}\, J(\mathbf{w})}{\boldsymbol{d}\mathbf{w}} = 0 \Rightarrow (\mathbf{w}^T \mathbf{S}_B \mathbf{w}) \mathbf{S}_W \mathbf{w} = (\mathbf{w}^T \mathbf{S}_W \mathbf{w}) \mathbf{S}_B \mathbf{w} \mid (\mathrm{lx}) \mathbf{S}_W^{-1}$$

Optimal solution: $\mathbf{w} \propto \mathbf{S}_W^{-1} (\mathbf{m}_2 - \mathbf{m}_1) \propto (\mathbf{m}_2 - \mathbf{m}_1)$

The above result is known as Fischer's linear discriminant. Strictly *it is not a discriminant*, but rather a direction of projection that can be used for classification in conjunction with a decision (e.g. threshold) operation.

Fischer's Linear Discriminant Computation

However, the objective $J(\mathbf{w})$ is invariant to rescaling $\mathbf{w} \rightarrow \alpha\mathbf{w}$. We can chose the denominator to be unity. We can then minimize

$$\min_{\mathbf{w}} -\frac{1}{2}\mathbf{w}^T\mathbf{S}_B\mathbf{w}$$

$$\mathbf{w}^T\mathbf{S}_W\mathbf{w} = 1$$

This corresponds to the (primal) Lagrangian

$$L_P = -\frac{1}{2}\mathbf{w}^T\mathbf{S}_B\mathbf{w} + \frac{1}{2}\lambda(\ \mathbf{w}^T\mathbf{S}_W\mathbf{w} - 1)$$

From the *KKT conditions*

$$\mathbf{S}_B\mathbf{w} = \boldsymbol{\lambda}\mathbf{S}_W\mathbf{w} \Rightarrow \mathbf{S}_W^{-1}\mathbf{S}_B\mathbf{w} = \boldsymbol{\lambda}\mathbf{w}$$

Generalized eigenvalue problem, as $\mathbf{S}_W^{-1}\mathbf{S}_B$ not symmetric

Fischer's Linear Discriminant Computation

$$\mathbf{S}_W^{-1}\mathbf{S}_B\mathbf{w} = \boldsymbol{\lambda}\mathbf{w}$$

- Given that $\mathbf{S}_B$ is symmetric positive definite, we can write

$$\mathbf{S}_B = \mathbf{S}_B{}^{1/2}\mathbf{S}_B{}^{1/2}$$

where $\mathbf{S}_B = \mathbf{U}\boldsymbol{\Sigma}\mathbf{U}^{\mathrm{T}}, \mathbf{S}_B{}^{1/2} = \mathbf{U}\boldsymbol{\Sigma}^{1/2}\mathbf{U}^{\mathrm{T}}$

- Defining $\mathbf{v} = \mathbf{S}_B{}^{1/2}\mathbf{w}$, we get

$$\mathbf{S}_B{}^{1/2}\mathbf{S}_W^{-1}\mathbf{S}_B{}^{1/2}\mathbf{v} = \lambda\mathbf{v}$$

- We have to solve a regular eigenvalue problem for a symmetric, positive definite matrix $\mathbf{S}_B{}^{1/2}\mathbf{S}_W^{-1}\mathbf{S}_B{}^{1/2}$
  - We can find solutions λ_k and $\mathbf{v}_k$ corresponding to $\mathbf{S}_B{}^{1/2}\mathbf{w}$
- *Which eigenvector and eigenvalue should we chose? The largest! Why?*
- Transforming to dual

$$\mathbf{w}^T\mathbf{S}_B\mathbf{w} = 1,\ \mathbf{w}^T\mathbf{S}_W\mathbf{w} = \frac{1}{\lambda_k} \Rightarrow L_D = \text{const.} + \frac{1}{2}\lambda_k$$

(need to maximize over λ)

The Logistic **S**igmoid (due to **S**-shape)

- This is also called a squashing function because it maps the entire real axis into a finite interval

- For classification, the output a is a smooth function of the inputs and the weights $\mathbf{w}$ $\longrightarrow$ $a = \mathbf{w}^T\mathbf{x} + w_0$
- Properties

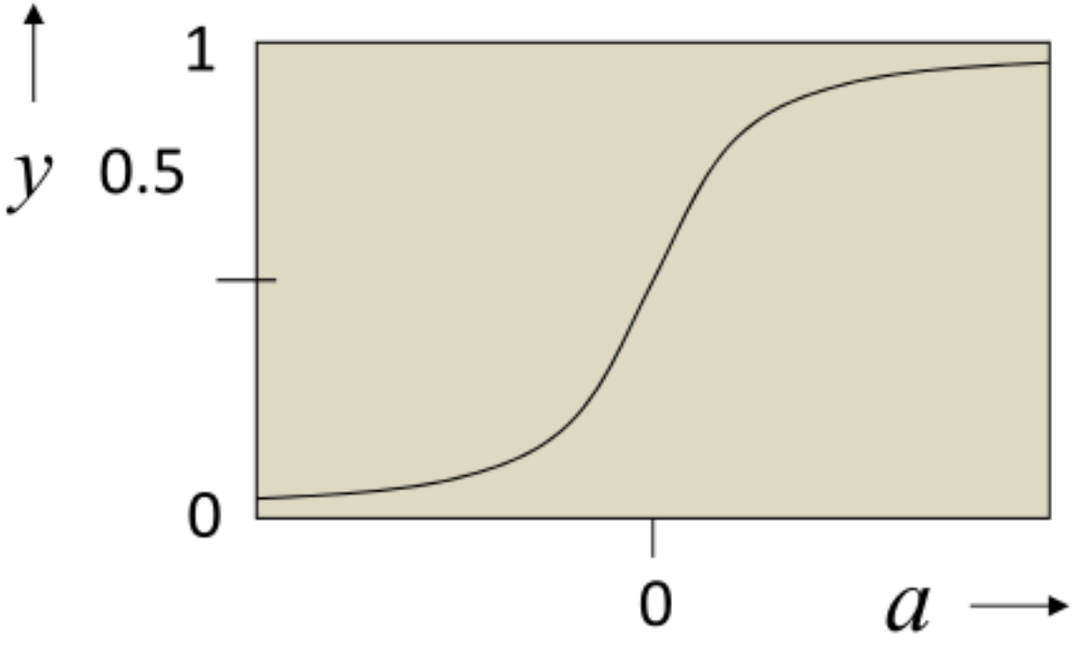
$$\sigma(-a) = 1 - \sigma(a), \qquad a = \ln\left(\frac{\sigma}{1-\sigma}\right)$$

logit function

$$y = \sigma(a) = \frac{1}{1+e^{-a}}$$

$$\frac{\partial a}{\partial w_i} = x_i \qquad \frac{\partial a}{\partial x_i} = w_i$$

$$\frac{dy}{da} = y\,(1-y)$$



Probabilistic Generative Models

- Use a class prior and a separate generative model of the input vectors for each class, and compute which model makes a test input vector most probable
- The posterior probability of class 1 is given by:

Logistic sigmoid

$$p(C_1 \mid \mathbf{x}) = \frac{p(C_1)p(\mathbf{x} \mid C_1)}{p(C_1)p(\mathbf{x} \mid C_1) + p(C_2)p(\mathbf{x} \mid C_2)} = \boxed{\frac{1}{1+e^{-a}}}$$

$$where \quad a = \ln \frac{p(C_1)p(\mathbf{x} \mid C_1)}{p(C_2)p(\mathbf{x} \mid C_2)} = \boxed{\ln \frac{p(C_1 \mid \mathbf{x})}{1 - p(C_1 \mid \mathbf{x})}}$$

z is called the logit and is given by the log odds

Multiclass Model (Softmax)

$$p(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)p(C_k)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|C_k)p(C_k)}{\sum_j p(\mathbf{x}|C_j)p(C_j)}$$

$$= \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

where $a_k = \ln p(\mathbf{x}|C_k)p(C_k)$

- This is known as the *normalized exponential*
- Can be viewed as a multiclass generalization of the logistic sigmoid
- It is also called a *softmax function* (it is a smoothed version of `max')

$$\text{if } a_k \gg a_j \ \forall j \neq k, \text{then } p(C_k|x) \simeq 1 \text{ and } p(C_j|\ x) \simeq 0$$

Gaussian Class-Conditionals

- Assume that the input vectors for each class are from a Gaussian distribution, and all classes have the same covariance matrix. The class conditionals are

normalizer (same) inverse covariance matrix

$$p(\mathbf{x} \mid C_k) = 1/Z \exp\left\{-\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)\right\}$$

- For two classes, C_1 and C_2, the posterior turns out to be a logistic

$$p(C_1 \mid \mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + w_0)$$

$$\mathbf{w} = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$

Quadratic terms in **x** canceled due to common covariance

$$w_0 = -\tfrac{1}{2}\boldsymbol{\mu}_1^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 + \tfrac{1}{2}\boldsymbol{\mu}_2^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_2 + \ln\frac{p(C_1)}{p(C_2)}$$

Interpretation of Decision Boundaries

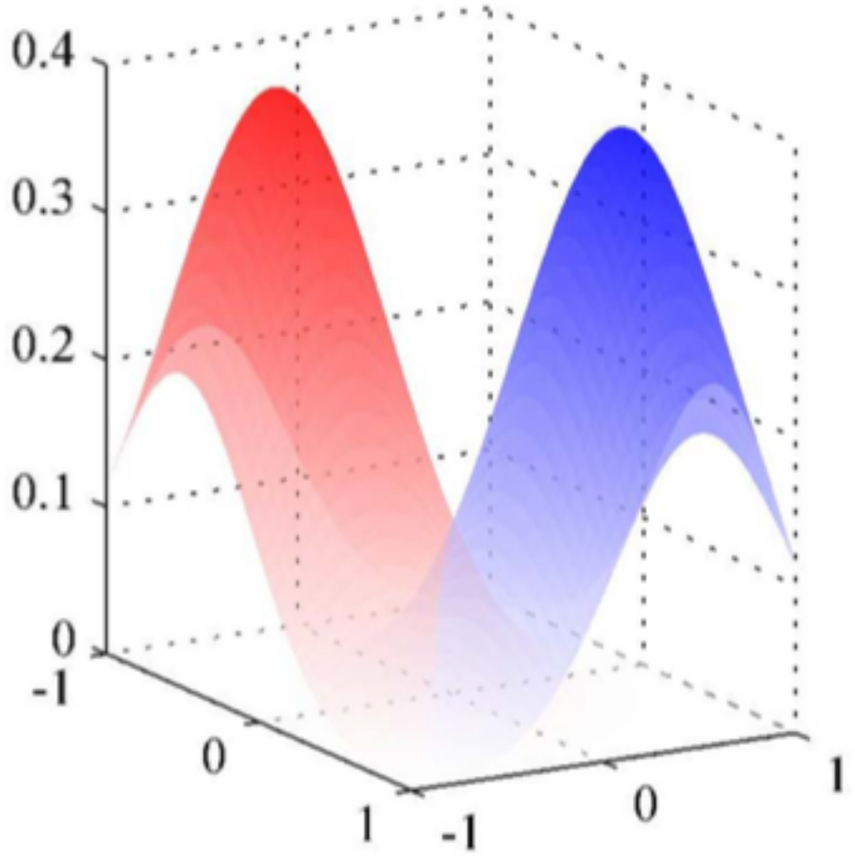
$$p(C_1 \mid \mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + w_0)$$

$$\mathbf{w} = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$

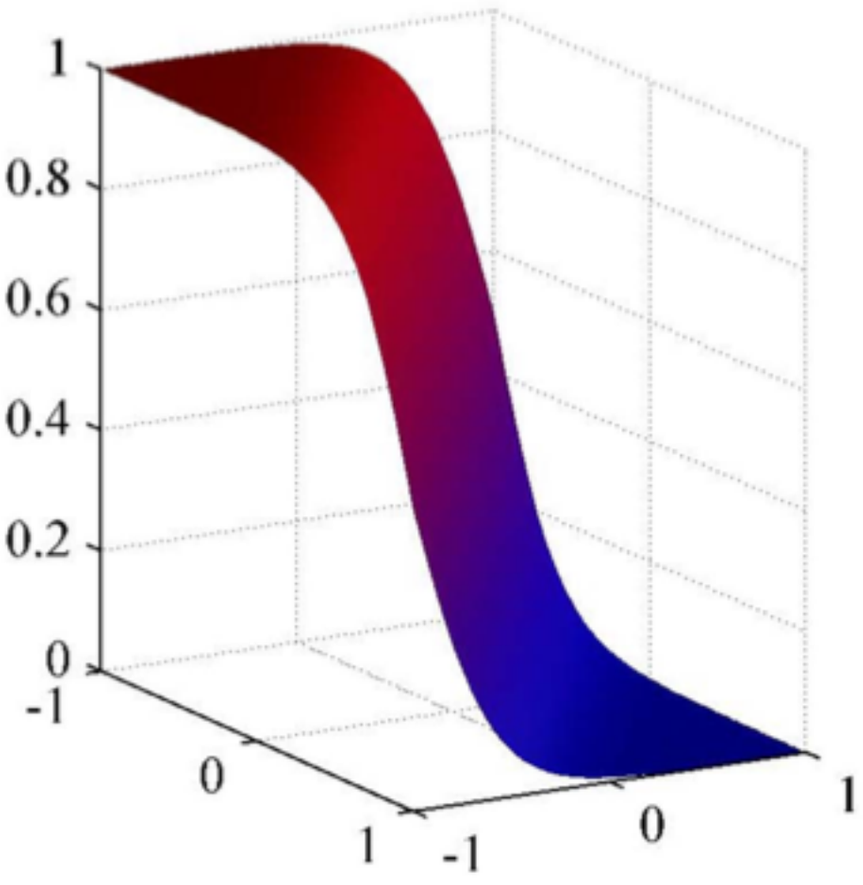
$$w_0 = -\tfrac{1}{2}\boldsymbol{\mu}_1^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 + \tfrac{1}{2}\boldsymbol{\mu}_2^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_2 + \ln\frac{p(C_1)}{p(C_2)}$$

- Quadratic terms canceled due to common covariance
- The sigmoid takes a linear function of $\mathbf{x}$ as argument
- The decision boundaries correspond to surfaces along which the posteriors $p(C_k|\mathbf{x})$ are constant, so they will be given by linear functions of $\mathbf{x}$. Thus, decision boundaries are linear functions in input space
- The prior probabilities $p(C_k)$ enter only through the bias parameter w_0, so changes in priors have the effect of making parallel shifts of the decision boundary (more generally of the parallel contours of constant posterior probability)

A picture of the two Gaussian models and the resulting posterior for the red class



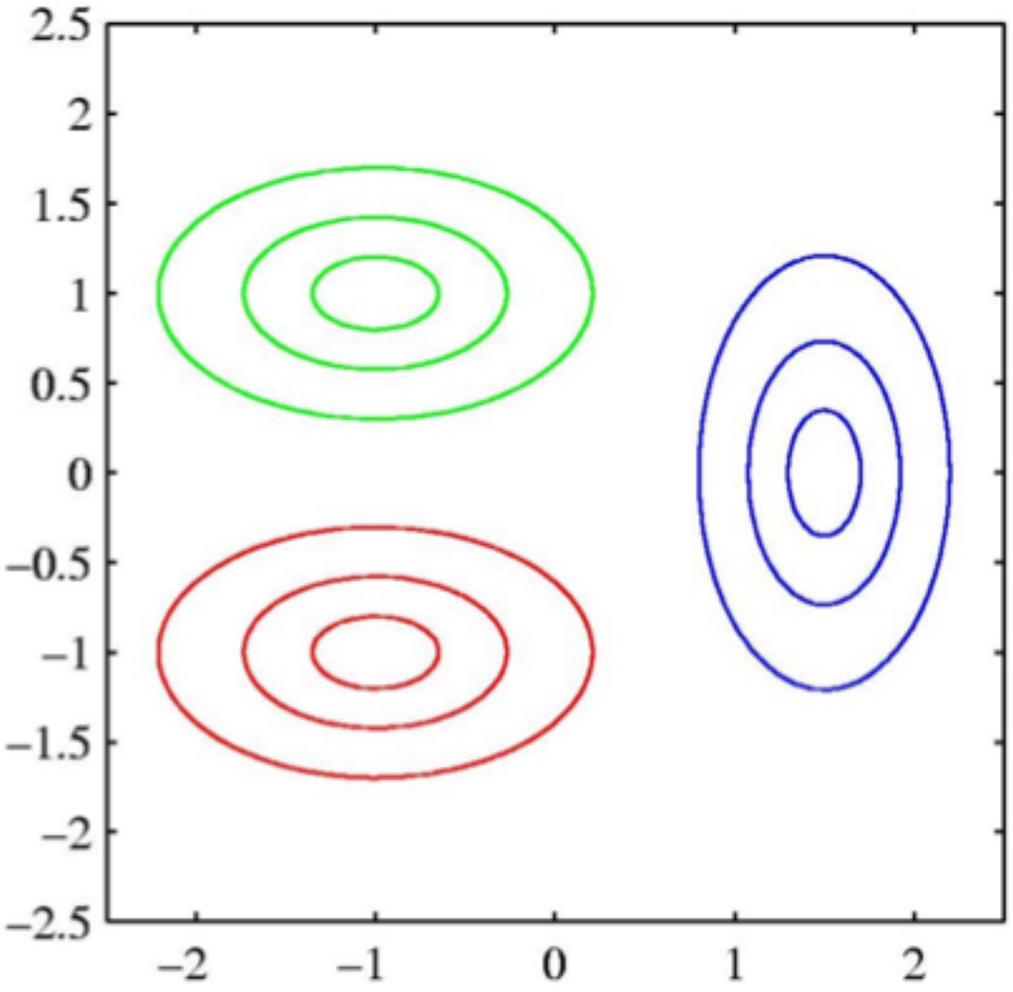
$p(\mathbf{x}|C_1), p(\mathbf{x}|C_2)$



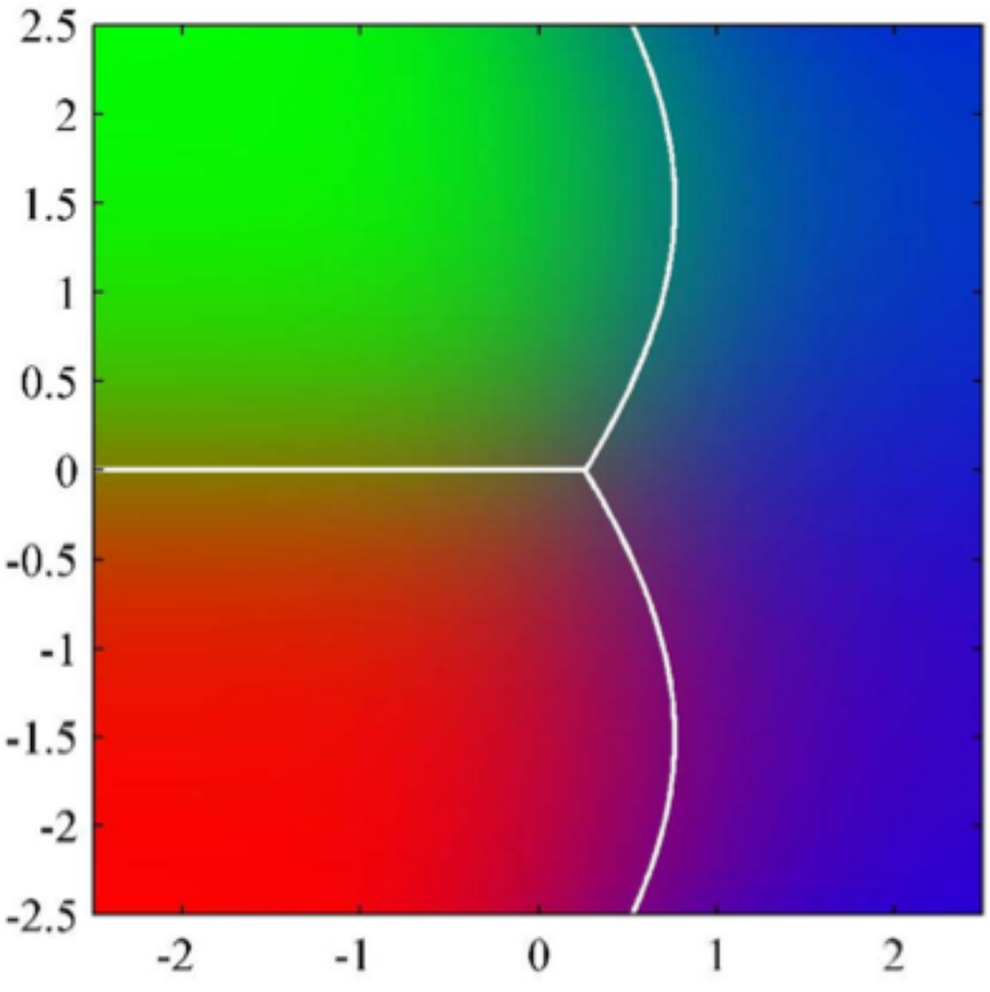
$p(C_1|\mathbf{x})$

The logistic sigmoid in the right-hand plot is coloured using a proportion of red tone given by $p(C_1|\boldsymbol{x})$ and a proportion of blue tone given by $p(C_2|\boldsymbol{x}) = 1 - p(C_1|\boldsymbol{x})$.

Class posteriors when covariance matrices are different for different classes



$p(\mathbf{x}|C_k)$



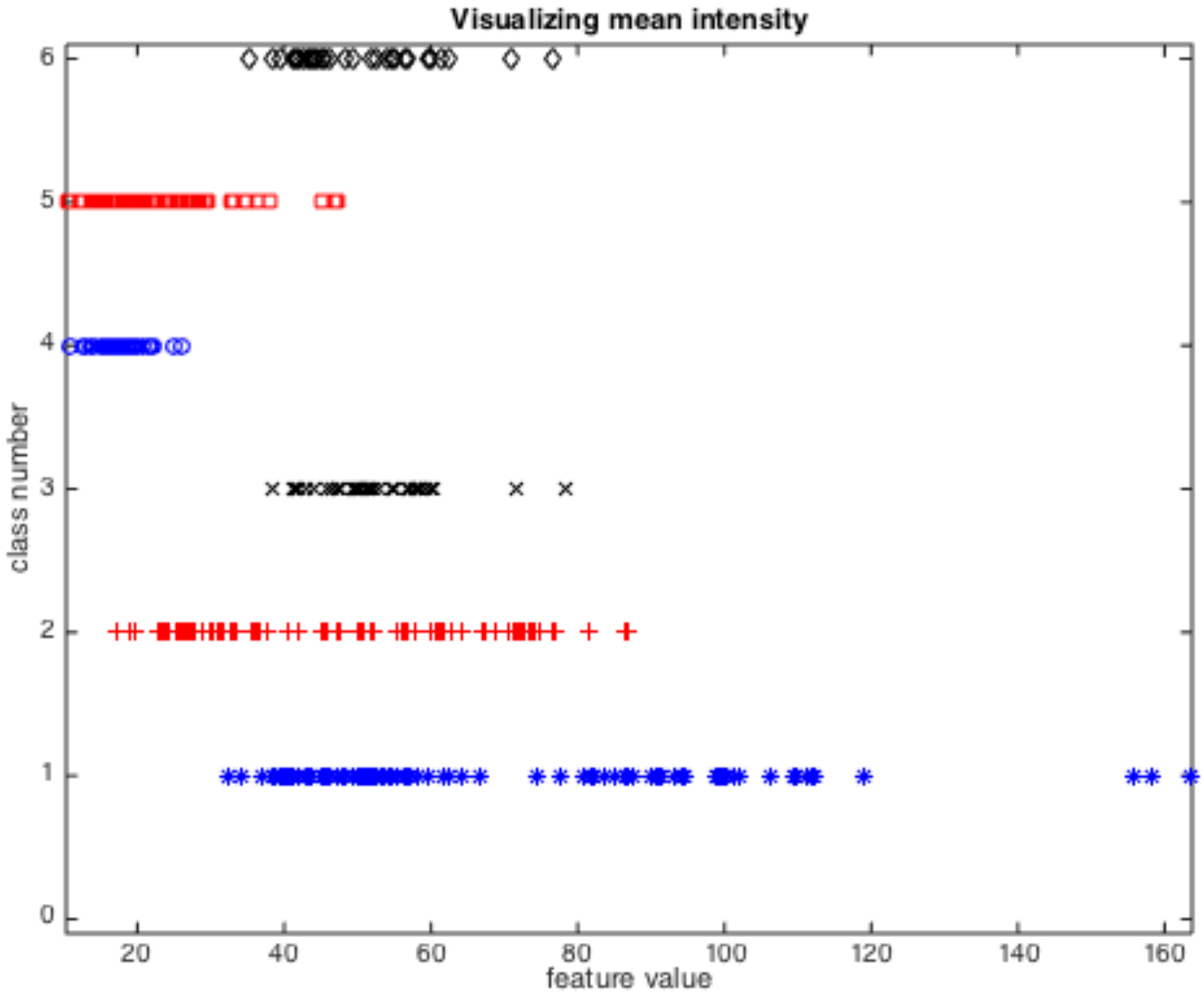
$p(C_k|\mathbf{x})$

The decision surface is planar when the covariance matrices are the same and quadratic when they are not

Overview – Machine Learning 2

- Machine Learning
- More Classification
  - Nearest Neighbor
  - Logistic Regression
  - Support Vector Machines
  - Discriminants
  - Multiclass problems
  - **Regression Trees**

Regression trees



Decision trees advantages

- Simple to understand and interpret
- Requires little preparation
- Can handle both continuous and discrete data
- ’white box’ model. You can easily explain a decision afterwards
- Robust
- Performs well with large datasets

Decision tree limitations

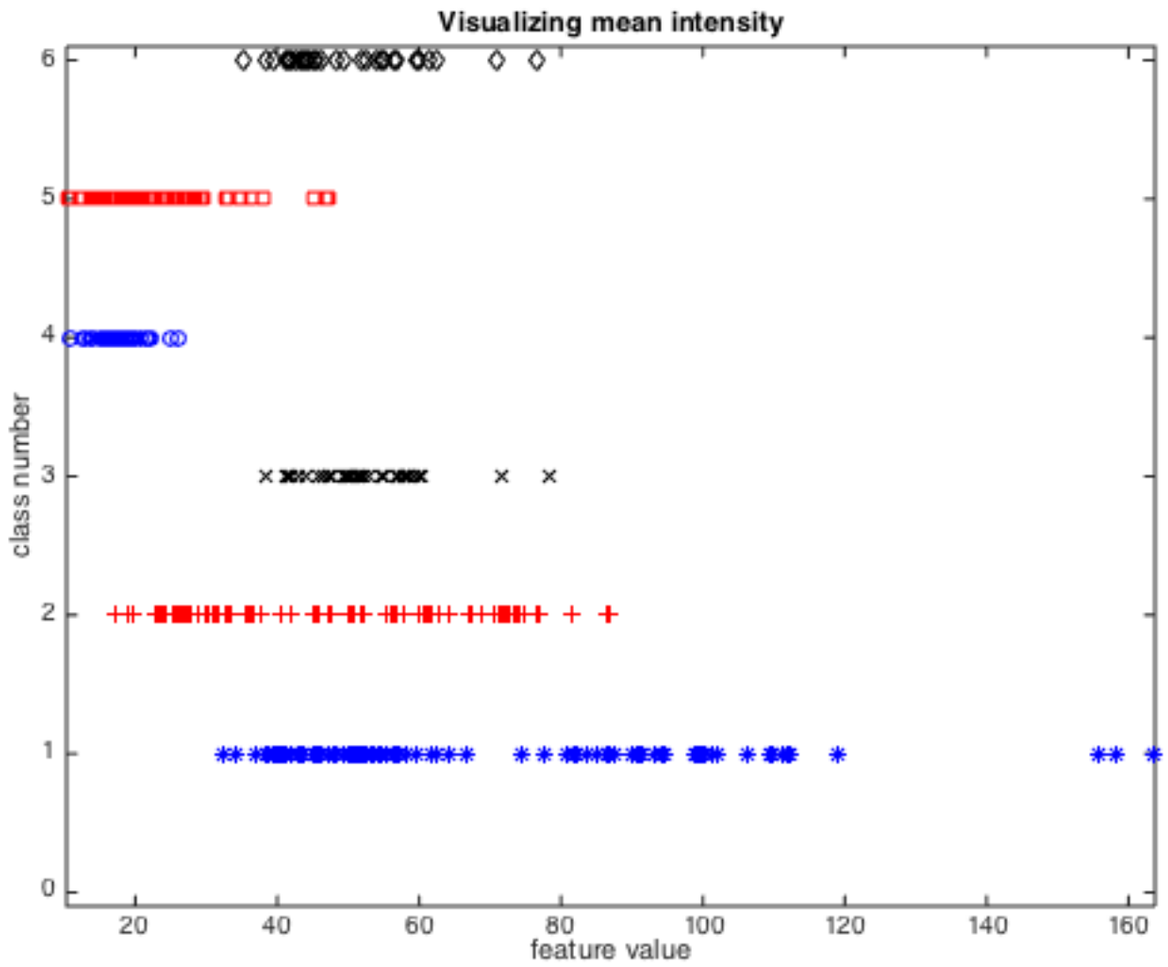
- Optimal learning is NP-complete (use heuristics)
- Problems with over fitting

Regression trees learning

- Try each variable
- Try each threshold
- Calculate score e g
  - Entropy

$$I_E(f) = -\sum_{i=1}^{m} f_i \log_2 f_i$$

  - Gini Impurity

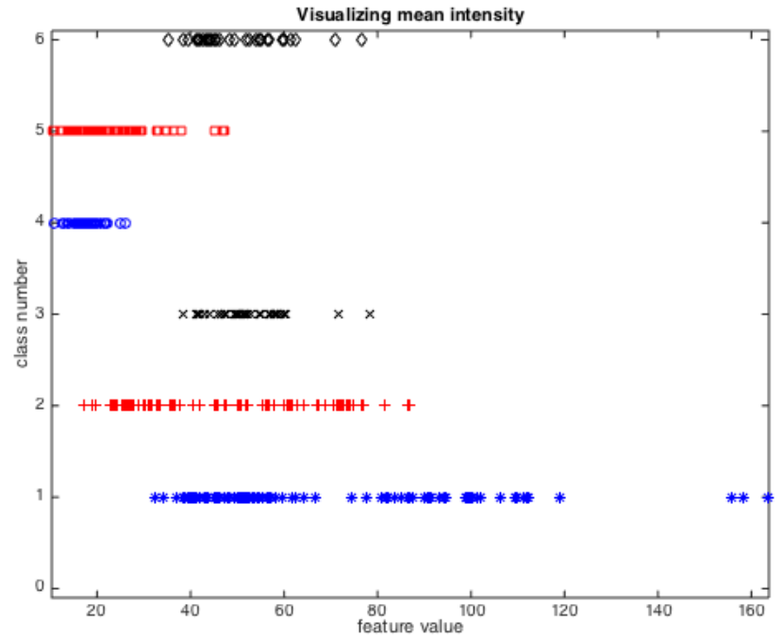


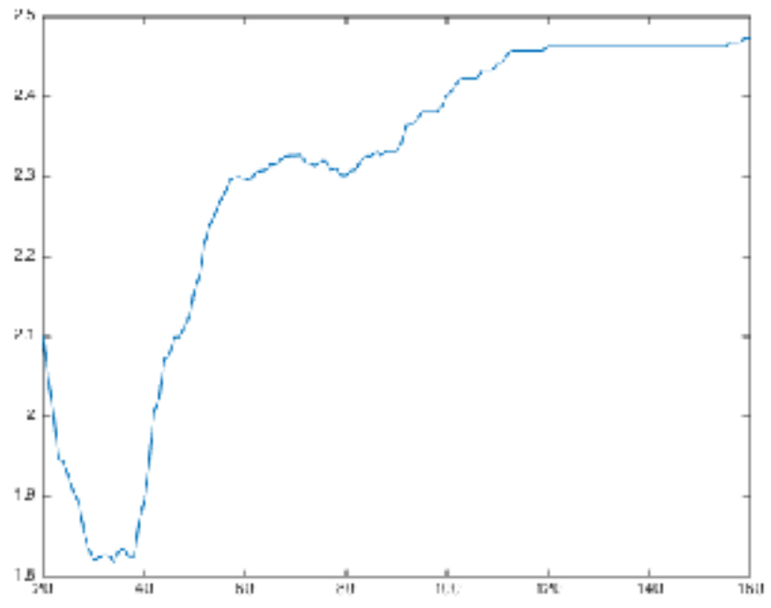
$$I_G(f) = \sum_{i=1}^{m} f_i(1 - f_i) = \sum_{i=1}^{m} (f_i - {f_i}^2) = \sum_{i=1}^{m} f_i - \sum_{i=1}^{m} {f_i}^2 = 1 - \sum_{i=1}^{m} {f_i}^2$$

Regression trees learning

- Log2(6) = 2.58 (bits)
- f = [0.24, 0.23, 0.11, 0.13, 0.21, 0.08]
- I(f) = 2.48 (bits)
- Try each threshold
- Calculate score e g
  - Entropy

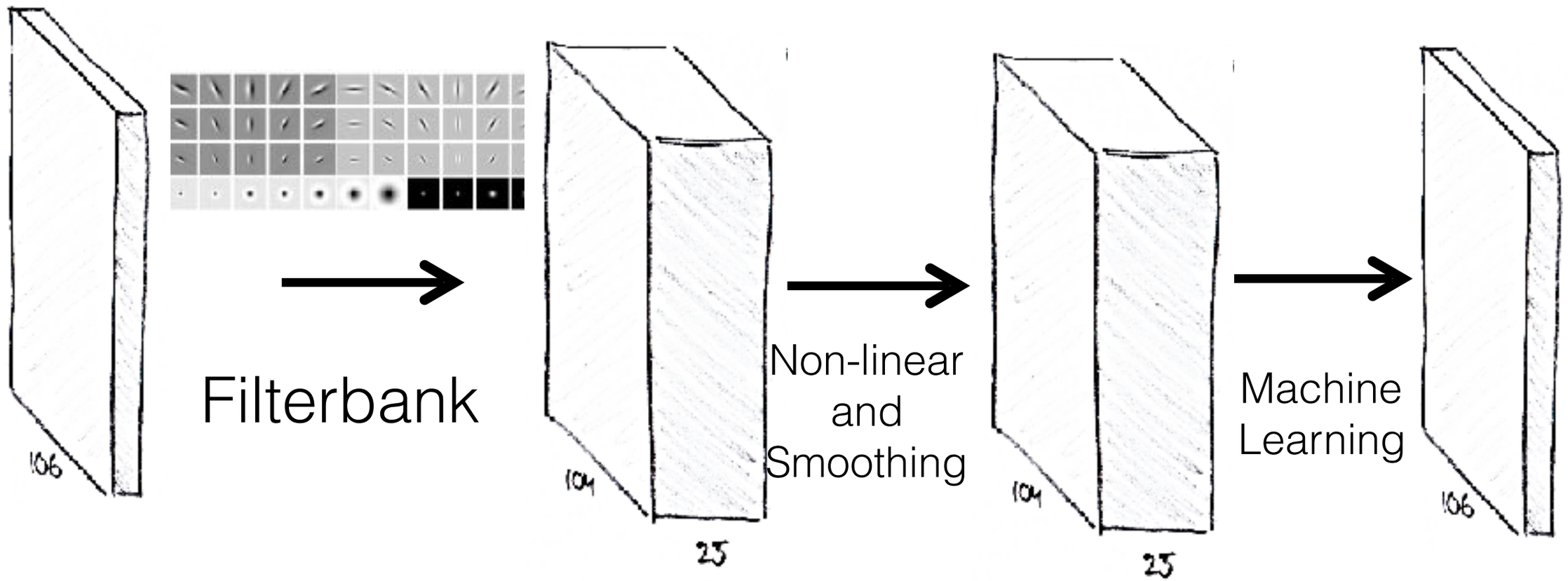
$$I_E(f) = -\sum_{i=1}^{m} f_i \log_2 f_i$$





Summary

- Machine Learning
  - Unsupervised Learning -
  - Supervised Learning -
- More Classification
  - Nearest Neighbor
  - Logistic Regression
  - Support Vector Machines
  - Discriminants
  - Multiclass problems
  - Regression Trees



Master’s Thesis Suggestion
Collarborative SLAM
How can many cars map the world


LUND
UNIVERSITY