

Data Mining

CS57300
Purdue University

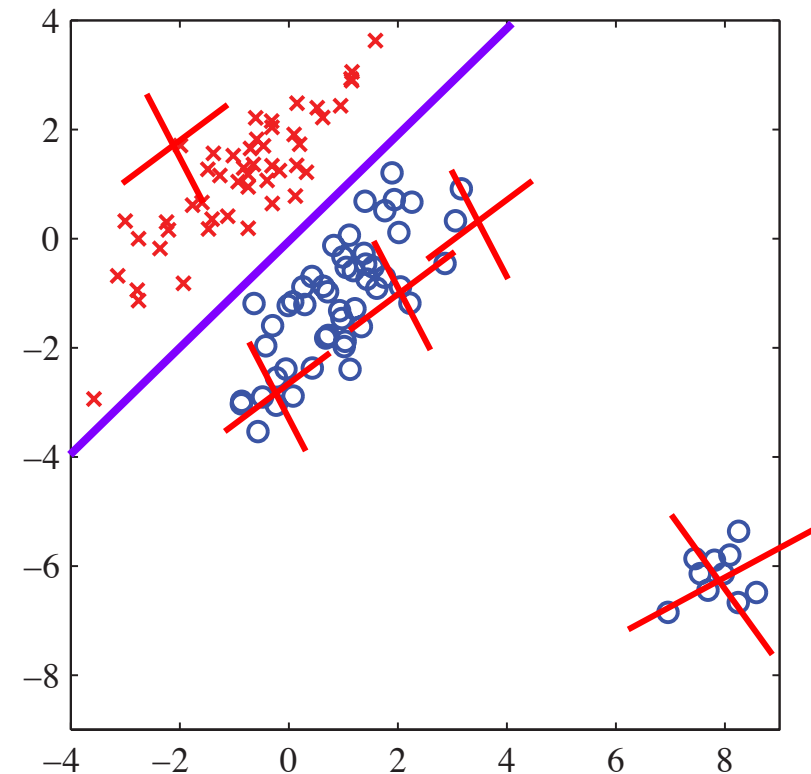
Bruno Ribeiro

January 25, 2018

Classification Tasks (cont)

Margin-based Classifiers

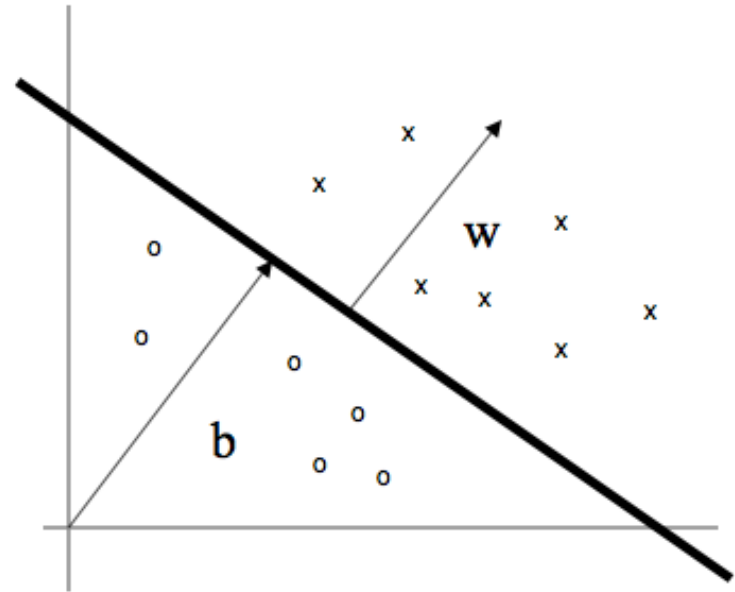
- Consider again the problem of finding a separating hyperplane between the classes
- Today we will "ignore" all points that are far away from the separating hyperplane
- This class:
 - Perceptron
 - SVMs



Perceptron

Model:

$$f(x) = \sum_{i=1}^m w_i x_i + b$$
$$y = \text{sign}[f(x)]$$



Dot product is product of:

(i) projection of x onto w , and (ii) the length of w

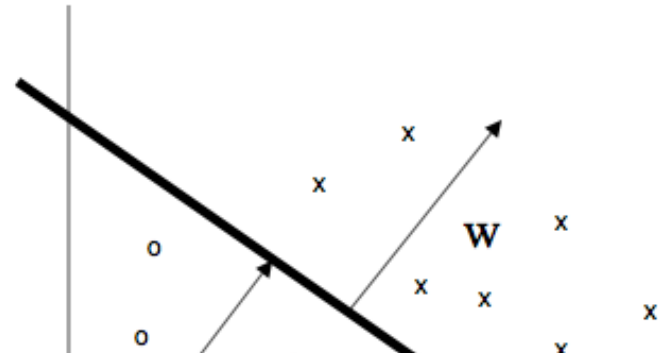
Dot product is 0 if x is perpendicular to w

Add b , if >0 then positive class, if <0 then negative class

Perceptron

Model:

$$f(x) = \sum_{i=1}^m w_i x_i + b$$
$$y = \text{sign}[f(x)]$$



Learning:

$$\text{if } y(j) \left(\sum_{i=1}^m w_i x_i(j) + b \right) \leq 0$$

$$\text{then } w \leftarrow w + \eta y(j) x(j) \quad \text{and} \quad b \leftarrow b + \eta y(j)$$

Iterate over training examples until error is below a pre-specified threshold γ

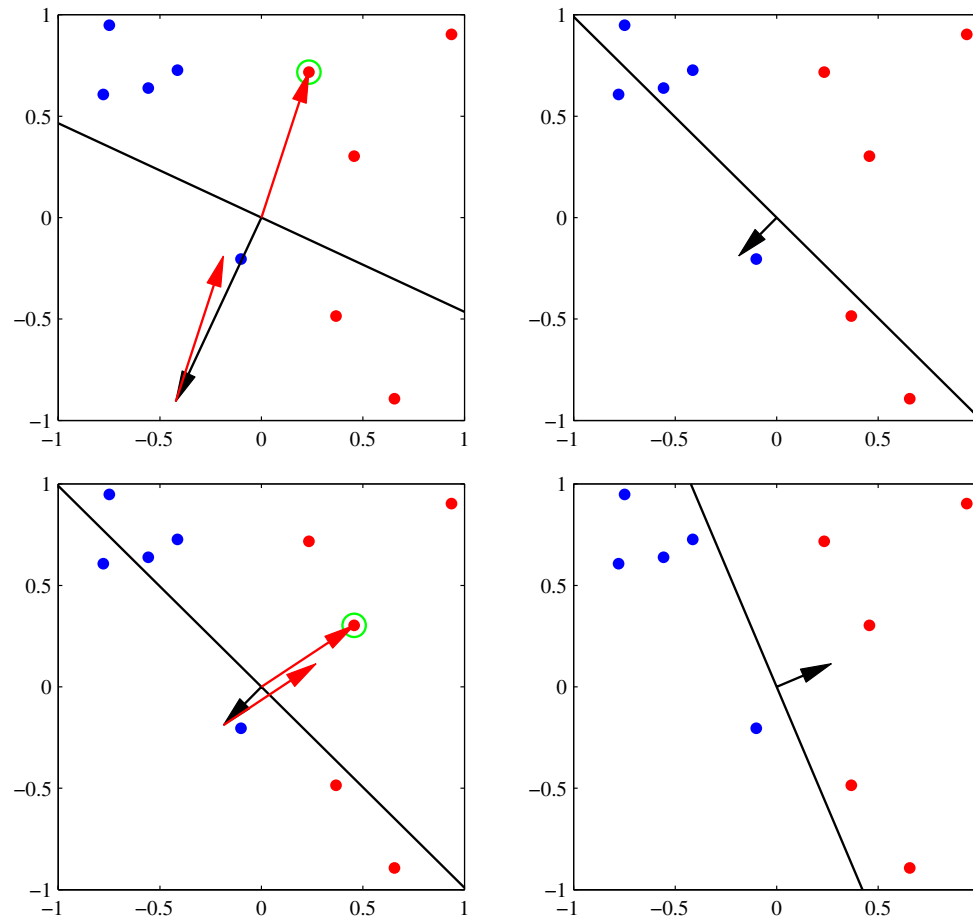


Figure 4.7 Illustration of the convergence of the perceptron learning algorithm, showing data points from two classes (red and blue) in a two-dimensional feature space (ϕ_1, ϕ_2) . The top left plot shows the initial parameter vector \mathbf{w} shown as a black arrow together with the corresponding decision boundary (black line), in which the arrow points towards the decision region which classified as belonging to the red class. The data point circled in green is misclassified and so its feature vector is added to the current weight vector, giving the new decision boundary shown in the top right plot. The bottom left plot shows the next misclassified point to be considered, indicated by the green circle, and its feature vector is again added to the weight vector giving the decision boundary shown in the bottom right plot for which all data points are correctly classified.

```
procedure LEARNPERCEPTRON(data,numIters,learnRate)  
   $\mathbf{w} \leftarrow 0$  (for  $p = 1 \dots \text{numAttrs}$ )  
   $b \leftarrow 0$   
   $\eta \leftarrow \text{learnRate}$   
  for  $\text{iter} \leq \text{numIters}$  do  
    for  $i = (\mathbf{x}_i, y_i) \in \text{data}$  do  
       $\hat{y}_i = \text{sign}(\mathbf{w} \cdot \mathbf{x}_i + b)$   
      if  $y_i \hat{y}_i \leq 0$  then  
         $e = \eta y_i$   
         $\mathbf{w} \leftarrow \mathbf{w}^{\text{old}} + e \mathbf{x}_i$   
         $b \leftarrow b + e$   
      end if  
    end for  
  end for  
  return  $\mathbf{w}, b$   
end procedure
```

Perceptron components

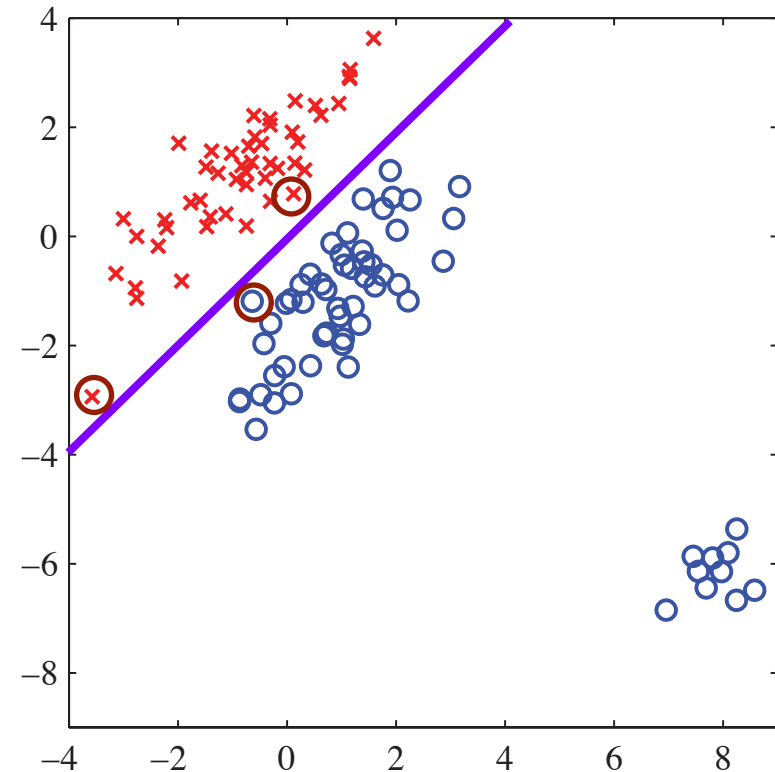
- **Model space**
 - Set of weights \mathbf{w} and b (hyperplane boundary)
- **Search algorithm**
 - Iterative refinement of weights and b
- **Score function**
 - Minimize misclassification rate

Perceptron Algorithm Convergence

- *Perceptron convergence theorem:*
 - If exact solution exists (i.e., if data is linearly separable), then perceptron algorithm will find an **exact solution** in a **finite number of steps**
 - The “finite” number of steps can be very large. The smaller the gap, the longer the time to find it
- **If data not linearly separable?**
 - **Algorithm will not converge**, and cycles develop. The cycles can be long and therefore hard to detect

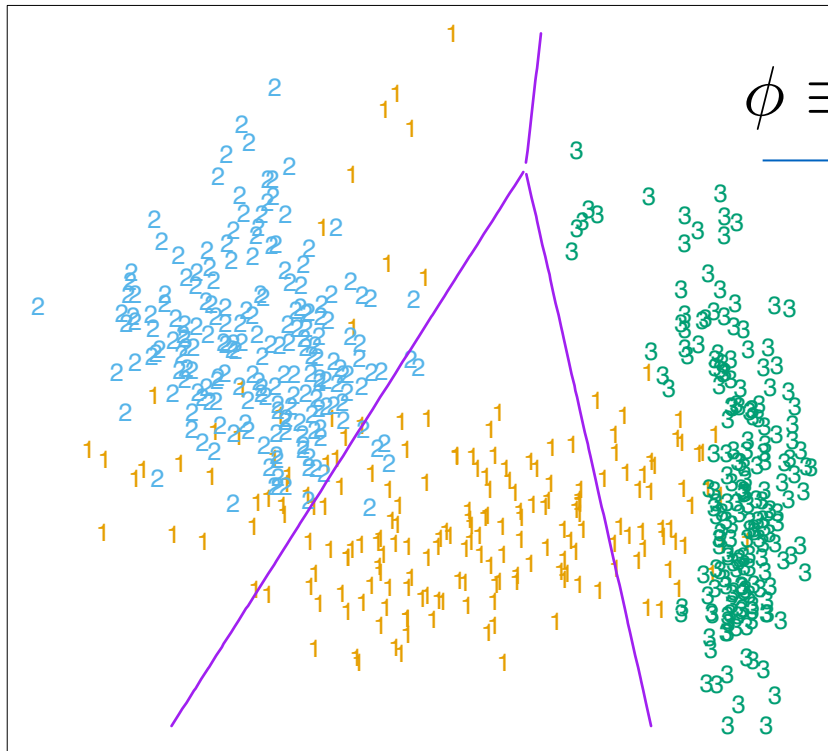
How Can We Extend the Perceptron Idea?

- Is there a better formulation for two classes?
- What about find hyperplane that is equidistant to circled points?
- We arrived at Maximum Margin Classifiers (e.g. Support Vector Machines)



Linear Methods → Linear Boundaries

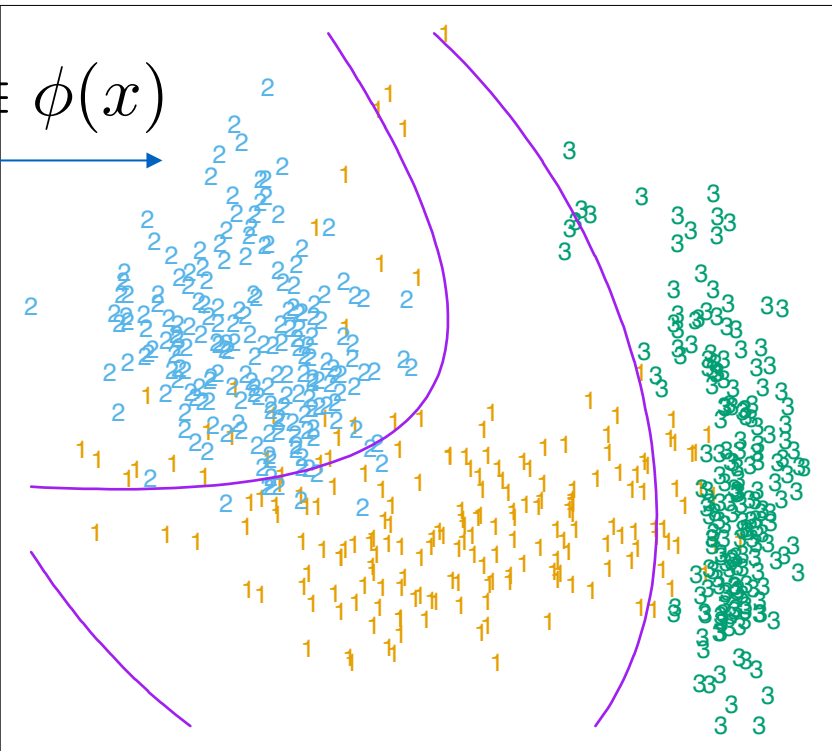
Data from three classes, with
linear decision boundaries



Linear boundaries in the
five-dimensional space

$$\phi(\mathbf{x}) = (X_1, X_2, X_1X_2, X_1^2, X_2^2)$$

$$\phi \equiv \phi(x)$$



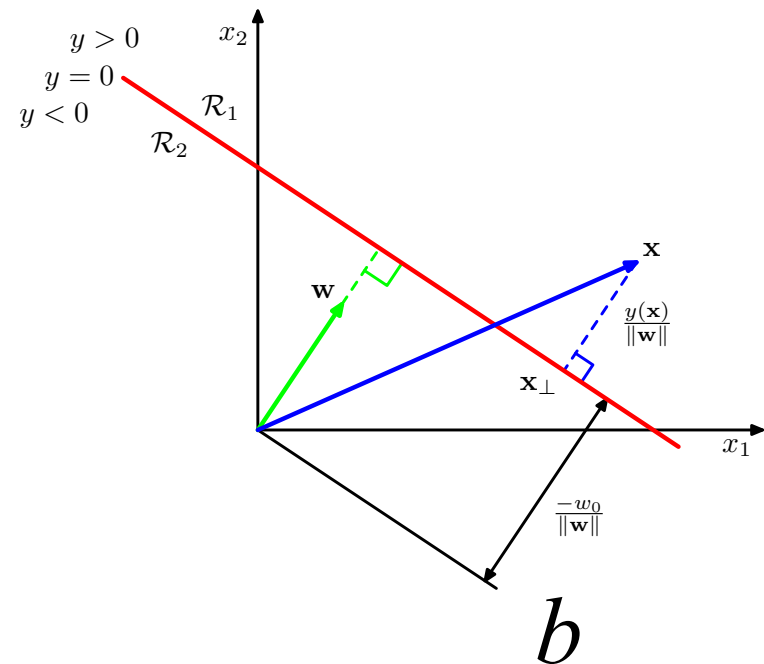
ϕ is known as a "kernel"

How to Find The "Support" Points? (assuming linearly separable data)

- Back to our linear model (with non-linear features ϕ)

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

- For two classes, if class $t_n \in \{-1, 1\}$ of item \mathbf{x}_n
- Then $t_n y(\mathbf{x}_n) > 0$ means correctly classified



$y(\mathbf{x})/\|\mathbf{w}\|$ determines distance to hyperplane

- Thus, distance of \mathbf{x}_n from decision hyperplane is the maximum minimum distance

$$\frac{t_n y(\mathbf{x}_n)}{\|\mathbf{w}\|} = \frac{t_n (\mathbf{w}^T \phi(\mathbf{x}_n) + b)}{\|\mathbf{w}\|} \Rightarrow \arg \max_{\mathbf{w}, b} \left\{ \frac{1}{\|\mathbf{w}\|} \min_n [t_n (\mathbf{w}^T \phi(\mathbf{x}_n) + b)] \right\}$$

几何距离, invariant with w Maximum margin solution

Modified Solution

$$\arg \max_{\mathbf{w}, b} \left\{ \frac{1}{\|\mathbf{w}\|} \min_n \left[t_n \left(\mathbf{w}^T \phi(\mathbf{x}_n) + b \right) \right] \right\}$$

- Too complex to compute
- Brings hyperplane too close to point

Constraint takes care of this part of the optimization
(because w and b are free variables, we can define the minimum to be one)

- We can recast problem into another optimization problem

$$t_n \left(\mathbf{w}^T \phi(\mathbf{x}_n) + b \right) \geq 1, \quad n = 1, \dots, N$$

- Data points for which the equality holds are said to be *active*
 - All other data points are *inactive*
 - There is always at least one active constraint
 - When margin is maximized there will be two active constraints

Modified Solution 2/2

$$\arg \max_{\mathbf{w}, b} \left\{ \frac{1}{\|\mathbf{w}\|} \min_n \left[t_n \left(\mathbf{w}^T \phi(\mathbf{x}_n) + b \right) \right] \right\}$$

- The original problem becomes (as $\operatorname{argmax} \|\mathbf{w}\|^{-1} = \operatorname{argmin} \|\mathbf{w}\|^2$) a quadratic programming problem

s.t.

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

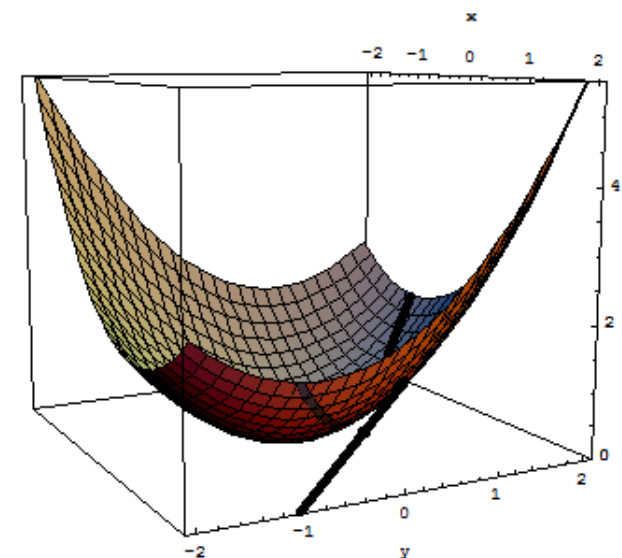
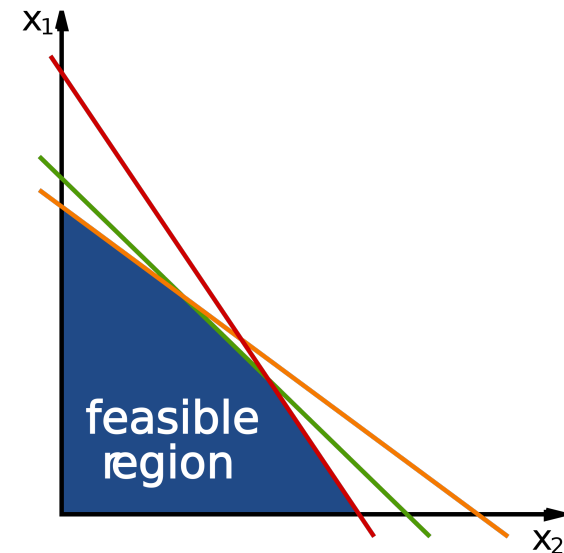
$$t_n \left(\mathbf{w}^T \phi(\mathbf{x}_n) + b \right) \geq 1, \quad n = 1, \dots, N$$

- We can solve constrained optimization problem via Lagrange multipliers $a_n \geq 0$

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{n=1}^N a_n \left\{ t_n \left(\mathbf{w}^T \phi(\mathbf{x}_n) + b \right) - 1 \right\}$$

Constrained optimization

- Linear programming (LP) is a technique for the optimization of a linear objective function, subject to linear constraints on the variables
- Quadratic programming (QP) is a technique for the optimization of a quadratic function of several variables, subject to linear constraints on these variables



Dual problem

- For a convex problem (no local minima) there is a dual problem that is equivalent to the primal problem (i.e. we can switch between them)
- Primal objective: minimize wrt to M feature variables
 - Quadratic programming problem with parameters w, b
- Dual objective: maximize wrt to N instances
 - Dual depends on inner product between feature vectors
 - → simpler quadratic programming problem

Solution to Modified Problem

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{n=1}^N a_n \{t_n(\mathbf{w}^T \phi(\mathbf{x}_n) + b) - 1\} \quad (1)$$

- Setting derivative of $L(\mathbf{w}, b, \mathbf{a})$ w.r.t. \mathbf{w} and b to zero we get:

$$\mathbf{w} = \sum_{n=1}^N a_n t_n \phi(\mathbf{x}_n)$$

$$0 = \sum_{n=1}^N a_n t_n.$$

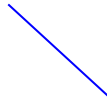
- Eliminating \mathbf{w} and b from equation (1) using these conditions:

s.t.

$$\tilde{L}(\mathbf{a}) = \sum_{n=1}^N a_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

$$a_n \geq 0, \quad n = 1, \dots, N,$$

$$\sum_{n=1}^N a_n t_n = 0.$$


$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$$

k is the "kernel"

Important Properties

- Satisfies *Karush-Kuhn-Tucker* (KKT) conditions so a solution to the nonlinear programming is optimal, as the following properties hold:

-

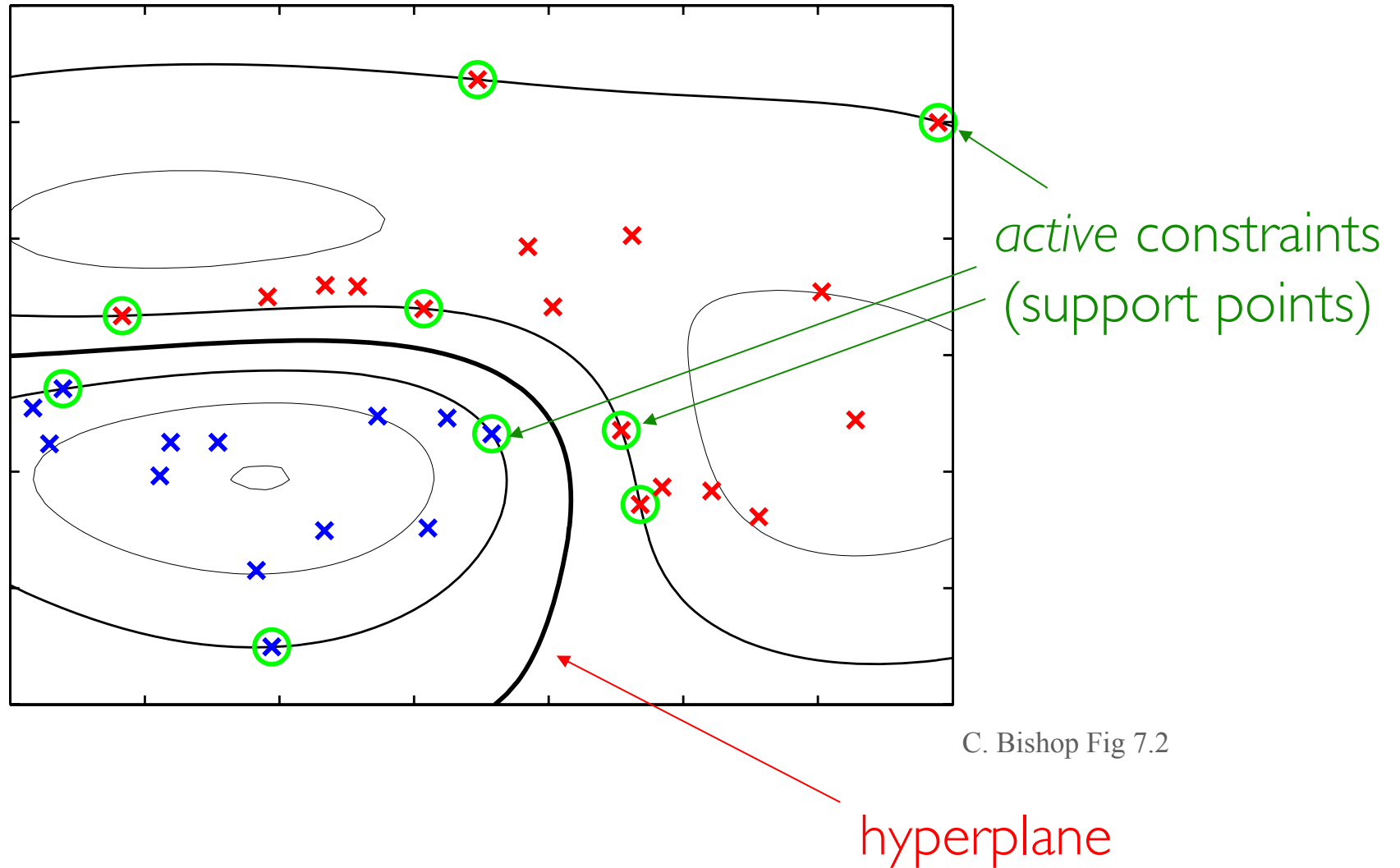
$$\begin{aligned}a_n &\geq 0 \\t_n y(\mathbf{x}_n) - 1 &\geq 0 \\a_n \{t_n y(\mathbf{x}_n) - 1\} &= 0.\end{aligned}$$

- Note that either $a_n = 0$ or $t_n y(\mathbf{x}_n) = 1$
 - What does it mean?
 - Point is either at the margin or we don't care about it in the sum

Modified
equation:

$$\tilde{L}(\mathbf{a}) = \sum_{n=1}^N a_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

Example with Gaussian Kernel



C. Bishop Fig 7.2

SVM components

- **Model space**
 - Set of weights \mathbf{w} and b (hyperplane boundary)
- **Search algorithm**
 - Quadratic programming over dual problem
- **Score function**
 - Minimize misclassification rate

Limitations of Standard SVMs

- Standard SVM classifier problems:
 - Cannot “naturally” deal with multiple classes
 - We must know a good kernel in advance
 - Cannot deal with noisy data (no buffer for mistakes)
- Solutions:
 - Multiple classes: a number of proposed academic approaches, not used much in practice
 - No known kernel: these days, practitioners will train a neural network (NN) and used as a kernel (latent representation of the data). Use SVM on top of this NN kernel (a hack of the highest caliber, but works in practice)
 - Noisy data: Soft margin (e.g., allow mistakes in training data [next])

What if data is not linearly separable?

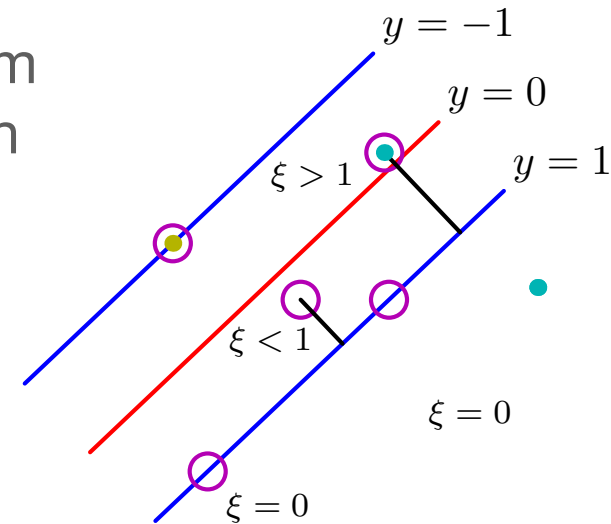
- Add slack variables $\xi_n \geq 0$ to optimization problem defined as 0 if correctly classified not within margin and $\xi_n = |t_n - y(\mathbf{x}_n)|$ otherwise
- The new constraints are

$$t_n y(\mathbf{x}_n) \geq 1 - \xi_n, \quad n = 1, \dots, N$$

- And it is easy to show that the optimization is now to minimize

for $C > 0$

$$C \sum_{n=1}^N \xi_n + \frac{1}{2} \|\mathbf{w}\|^2$$



SVMs for Non-linearly Separable Data

- Minimize

$$\begin{aligned} \text{s.t.} \quad & \tilde{L}(\mathbf{a}) = \sum_{n=1}^N a_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \\ & 0 \leq a_n \leq C \\ & \sum_{n=1}^N a_n t_n = 0 \end{aligned}$$

where C can be seen as a penalty for misclassification

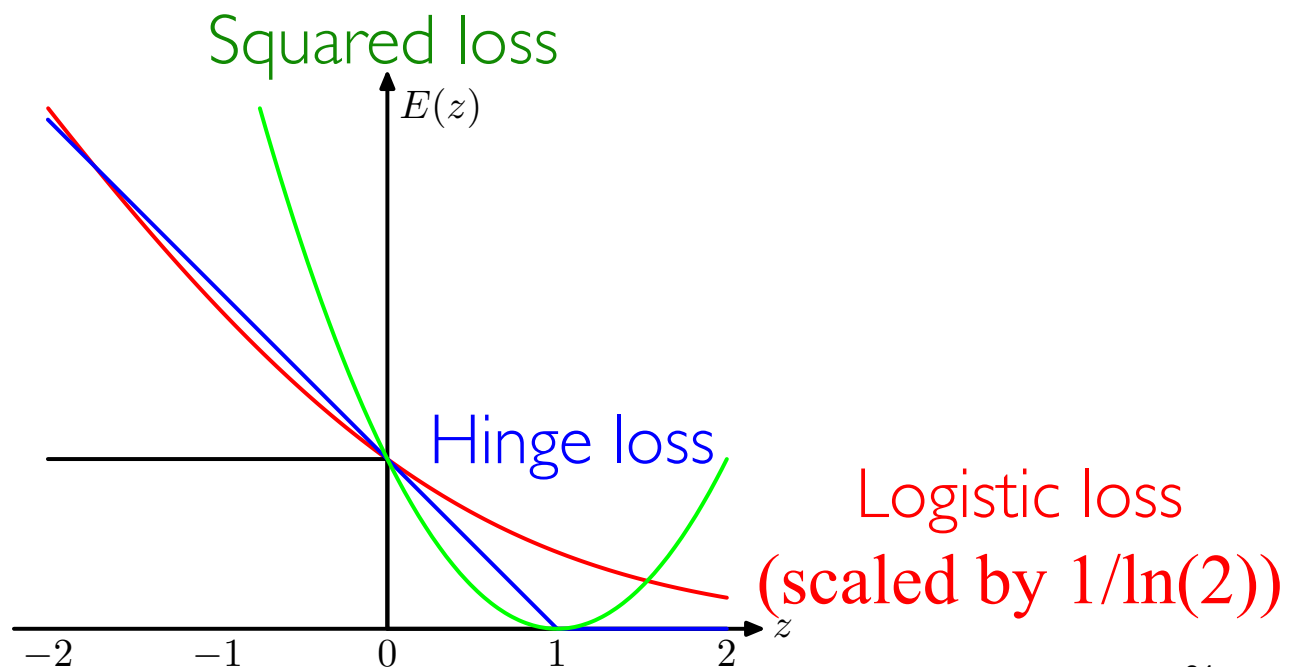
- For $C \rightarrow \infty$ we recover the linearly separable case

Relationship Between Logistic Regression and SVMs

- SVM can be described as optimizing

$$\sum_{n=1}^N E_{SV}(y_n, t_n) + \frac{1}{2C} \|\mathbf{w}\|^2$$

where $E_{SV}(y_n t_n) = [1 - y_n t_n]_+$
is known as *hinge loss* function,
 $[\cdot]_+$ is positive part



Other predictive models

Nearest neighbor

- Instance-based method
- Learning
 - Stores training data and delays processing until a new instance must be classified
 - Assumes that all points are represented in p -dimensional space
- Prediction
 - **Nearest neighbors** are calculated using Euclidean distance
 - Classification is made based on class labels of neighbors

1NN

- Training set: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)$
where $\mathbf{x}_i = [x_{i1}, x_{i2}, \dots, x_{ip}]$ is a feature vector of p continuous attributes
and y_i is a discrete class label

- **1NN algorithm**

To predict a class label for new instance j :

Find the training instance point \mathbf{x}_i such that $d(\mathbf{x}_i, \mathbf{x}_j)$ is minimized

Let $f(\mathbf{x}_j) = y_i$

- Key idea: Find instances that are “similar” to the new instance and use their class labels to make prediction for the new instance
 - 1NN generalizes to kNN when more neighbors are considered

kNN

- **kNN algorithm**

To predict a class label for new instance j :

Find the k nearest neighbors of j , i.e., those that minimize $d(\mathbf{x}_k, \mathbf{x}_j)$

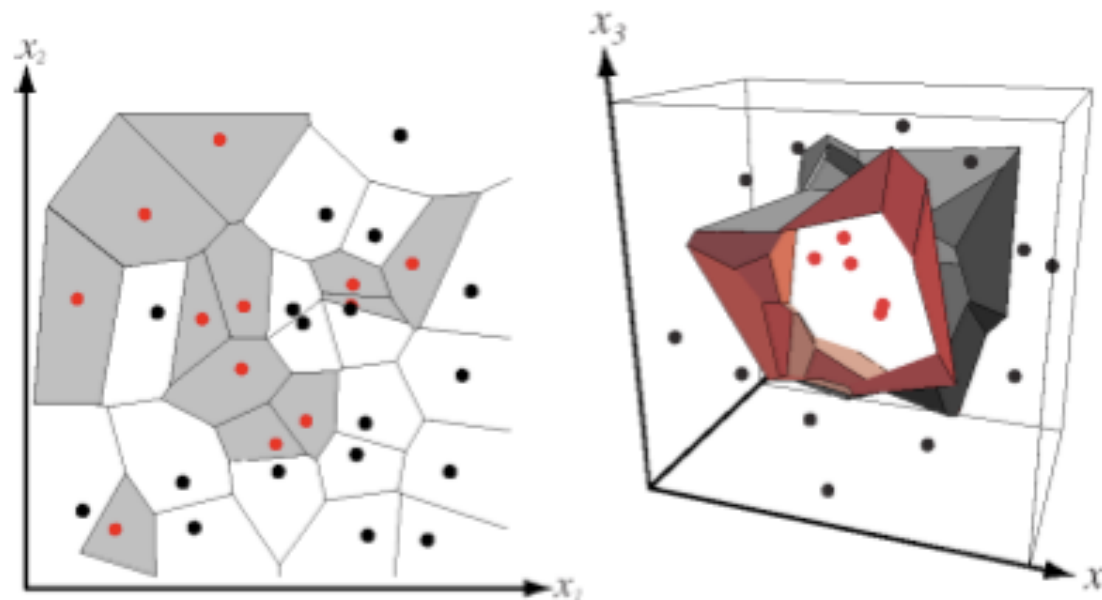
Let $f(\mathbf{x}_j) = g(\mathbf{y}_k)$, e.g., majority label in \mathbf{y}_k

- *Algorithm choices*

- How many neighbors to consider (i.e., choice of k)?
... Usually a small value is used, e.g. $k < 10$
- What distance measure $d()$ to use?
... Euclidean L2 distance is often used
- What function $g()$ to combine the neighbors' labels into a prediction?
... Majority vote is often used

1NN decision boundary

- For each training example i , we can calculate its **Voronoi cell**, which corresponds to the space of points for which i is their nearest neighbor
- All points in such a Voronoi cell are labeled by the class of the training point, forming a Voronoi tessellation of the feature space



Nearest neighbor

- Strengths:
 - Simple model, easy to implement
 - Very efficient learning: $O(1)$
- Weaknesses:
 - Inefficient inference: time and space $O(n)$
 - Curse of dimensionality:
 - As number of features increase, you need an exponential increase in the size of the data to ensure that you have nearby examples for any given data point

k-NN learning

- Parameters of the model:
 - k (number of neighbors)
 - any parameters of distance measure d (e.g., weights on features)
- **Model space**
 - Possible tessellations of the feature space
- **Search algorithm**
 - Implicit search: choice of k (number of neighbors), d (distance function), and g (classification function) uniquely define a tessellation
- **Score function**
 - Depends on g , but if majority vote then minimize misclassification rate