# KERNELS, SVM

CS5011- MACHINE LEARNING

Content Credits:
Murphy - Sections 14.1, 14.2, 14.3, 14.4, 14.5

# Introduction

- How do we represent a text document or protein sequence, which can be of variable length?
- One approach is to define a generative model for the data, and use the inferred latent representation and/or the parameters of the model as features, and then to plug these features in to standard methods
- Another approach is to assume that we have a way of measuring the similarity between objects, that doesn't require preprocessing them into feature vector format
- For example, when comparing strings, we can compute the edit distance between them

# Kernel functions

- We define a kernel function to be a real-valued function of two arguments, $\kappa(\mathbf{x}, \mathbf{x}') \in \mathbb{R}$, for $\mathbf{x}, \mathbf{x}' \in \mathrm{X}$.
- Typically the function has the following properties:
  - Symmetric
  - Non-negative
  - Can be interpreted as a measure of similarity
- We will discuss several examples of kernel functions

# RBF kernels

- **Squared exponential kernel** (SE kernel) or **Gaussian kernel**

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{x}')^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\mathbf{x}')\right)$$

- If $\boldsymbol{\Sigma}$ is diagonal, this can be written as

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}\sum_{j=1}^{D}\frac{1}{\sigma_j^2}(x_j - x_j')^2\right)$$

  We can interpret the $\sigma_j$ as defining the **characteristic length scale** of dimension *j*

- If $\boldsymbol{\Sigma}$ is spherical, we get the isotropic kernel

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x}-\mathbf{x}'||^2}{2\sigma^2}\right)$$

  An example of RBF (Radial basis function)kernel (since it is a function of $||\mathbf{x} - \mathbf{x}'||$) where $\sigma^2$ is known as the **bandwidth**

# Mercer (positive definite) kernels

- Gram matrix is defined as

$$\mathbf{K} = \begin{pmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_1, \mathbf{x}_N) \\ & \vdots & \\ \kappa(\mathbf{x}_N, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix}$$

- If the Gram matrix is positive definite for any set of inputs, the Kernel is a Mercer kernel
- Mercer's theorem: If the Gram matrix is positive definite, we can compute an eigenvector decomposition of it as follows: $\mathbf{K} = \mathbf{U}^T \mathbf{\Lambda} \mathbf{U}$
- where $\mathbf{\Lambda}$ is a diagonal matrix of eigenvalues $\lambda_i > 0$
- Now consider an element of $\mathbf{K}$

$$k_{ij} = (\mathbf{\Lambda}^{\frac{1}{2}} \mathbf{U}_{:,i})^T (\mathbf{\Lambda}^{\frac{1}{2}} \mathbf{U}_{:j})$$

$$k_{ij} = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \qquad \phi(\mathbf{x}_i) = \mathbf{\Lambda}^{\frac{1}{2}} \mathbf{U}_{:i}$$

# Mercer (positive definite) kernels

- In general, if the kernel is Mercer, then there exists a function $\phi$ mapping $\mathbf{x} \in X$ to $\mathbb{R}^D$ such that

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

- For example, consider the (non-stationary) polynomial kernel

$$\kappa(\mathbf{x}, \mathbf{x}') = (\gamma \mathbf{x}^T \mathbf{x}' + r)^M$$

If $M = 2$, $\gamma = r = 1$ and $\mathbf{x}, \mathbf{x}' \in R^2$, we have

$$\begin{aligned}(1 + \mathbf{x}^T \mathbf{x}')^2 &= (1 + x_1 x_1' + x_2 x_2')^2 \\ &= 1 + 2x_1 x_1' + 2x_2 x_2' + (x_1 x_1)^2 + (x_2 x_2')^2 + 2x_1 x_1' x_2 x_2'\end{aligned}$$

This can be written as $\phi(\mathbf{x})^T \phi(\mathbf{x}')$, where

$$\phi(\mathbf{x}) = [1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2]^T$$

So using this kernel is equivalent to working in a 6 dimensional feature space.

# Using kernels inside GLMs

- We define a kernel machine to be a GLM (generalized linear model) where the input feature vector has the form

$$\phi(\mathbf{x}) = [\kappa(\mathbf{x}, \boldsymbol{\mu}_1), \ldots, \kappa(\mathbf{x}, \boldsymbol{\mu}_K)]$$

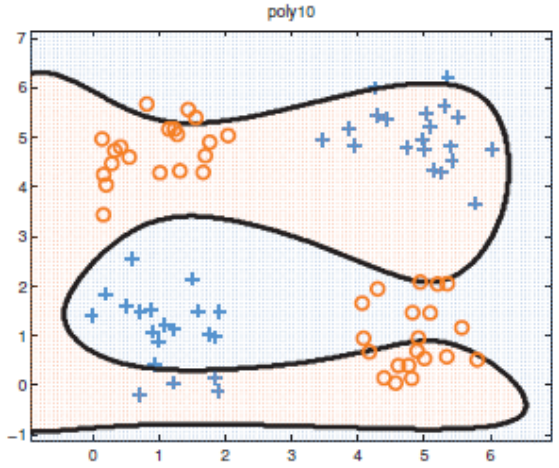
  where $\boldsymbol{\mu}_k \in X$ are a set of $K$ centroids
- If $\kappa$ is an RBF kernel, this is called an RBF network
- We will discuss ways to choose the $\boldsymbol{\mu}_k$ parameters
- Note that in this approach, the kernel need not be a Mercer kernel.

# Using kernels inside GLMs

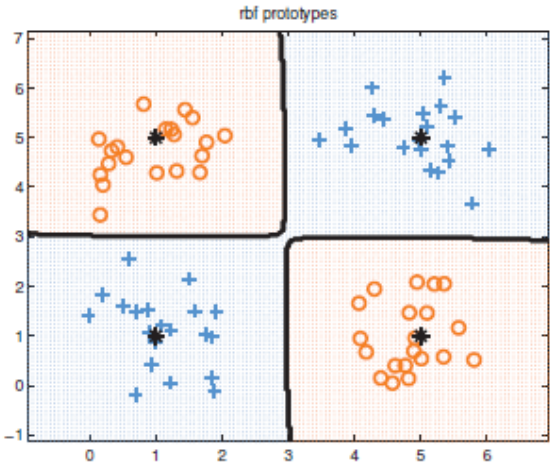
- This provides a simple way to define a non-linear decision boundary
- As an example, consider the data coming from the *exclusive or* or *xor* function.

| $x_1$ | $x_2$ | $y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

(a)




(b)




(c)

(a) xor truth table. (b) Fitting a linear logistic regression classifier using degree 10 polynomial expansion. (c) Same model, but using an RBF kernel with centroids specified by the 4 black crosses

# L1VMs, and other sparse vector machines

- The main issue with kernel machines is: how do we choose the centroids $\boldsymbol{\mu}_k$?
- If the input is low-dimensional Euclidean space, we can uniformly tile the space occupied by the data with prototypes
- However, this approach breaks down in higher numbers of dimensions because of the curse of dimensionality
- A simpler approach is to make each example $\mathbf{x}_i$ be a prototype, so we get

$$\phi(\mathbf{x}) = [\kappa(\mathbf{x}, \mathbf{x}_1), \ldots, \kappa(\mathbf{x}, \mathbf{x}_N)]$$

# L1VMs, and other sparse vector machines

- Now $D = N$, we have as many parameters as data points
- However, we can use any of the sparsity- promoting priors for $\boldsymbol{w}$ to efficiently select a subset of the training exemplars. We call this a **sparse vector machine**
- Most natural choice is to use $\ell_1$ regularization resulting in **L1VM** or "$\ell_1$-regularised vector machine"
- By analogy, we define the use of an $\ell_2$ regularizer to be a **L2VM** or "$\ell_2$- regularized vector machine"
- Another very popular approach to creating a sparse kernel machine is to use a **support vector machine** or **SVM**

# The kernel trick

- Rather than defining our feature vector in terms of kernels, $\phi(\mathbf{x}) = [\kappa(\mathbf{x}, \mathbf{x}_1), \ldots, \kappa(\mathbf{x}, \mathbf{x}_N)]$, we can work with the original feature vectors $\mathbf{x}$, but modify the algorithm so that it replaces all inner products of the form $< \mathbf{x}, \mathbf{x}' >$ with a call to the kernel function, $\kappa(\mathbf{x}, \mathbf{x}')$

- This is called the kernel trick.

# Support vector machines (SVMs)

- Consider the $\ell_2$ regularized empirical risk function

$$J(\mathbf{w}, \lambda) = \sum_{i=1}^{N} L(y_i, \hat{y}_i) + \lambda ||\mathbf{w}||^2 \qquad \hat{y}_i = \mathbf{w}^T \mathbf{x}_i + w_0$$

- If $L$ is quadratic loss, this is equivalent to ridge regression
- We can rewrite these equations in a way that only involves inner products of the form $\mathbf{x}^T \mathbf{x}$, which we can replace by calls to a kernel function, $\kappa(\mathbf{x}, \mathbf{x})$
- This is kernelized, but not sparse
- If we replace the quadratic loss with some other loss function, we can ensure that the solution is sparse, so that predictions only depend on a subset of the training data, known as **support vectors**
- This combination of the kernel trick plus a modified loss function is known as a **support vector machine** or **SVM**

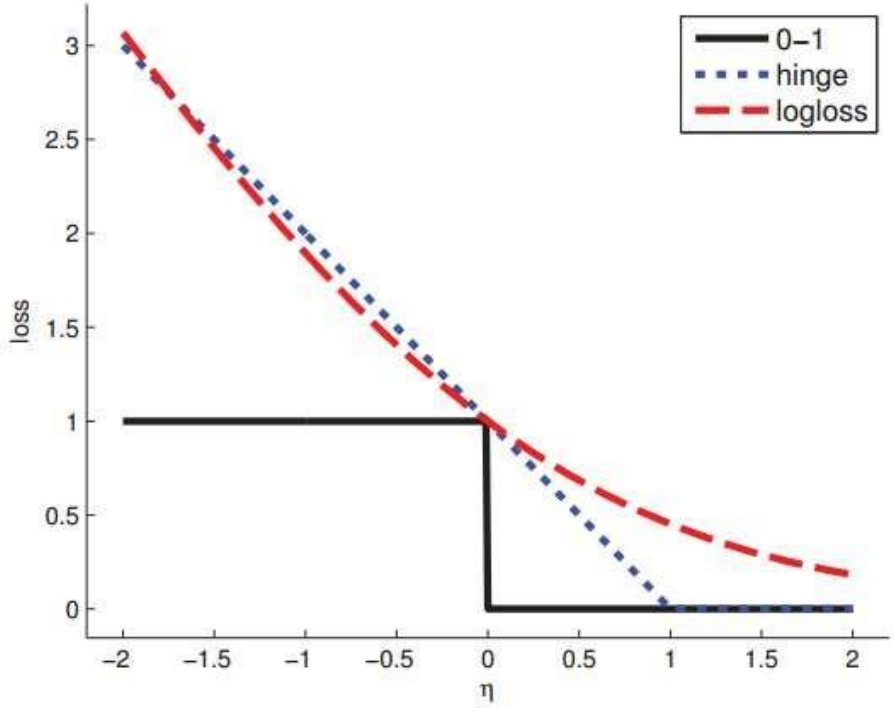
# SVMs for classification

- The Hinge loss is defined as:

$$L_{\text{hinge}}(y, \eta) = \max(0, 1 - y\eta) = (1 - y\eta)_+$$

- We have assumed the labels are $y \in \{1, -1\}$, $\eta = f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$ is our "confidence" in choosing label $y = 1$; however, it need not have any probabilistic semantics

Illustration of various loss functions for binary classification. The horizontal axis is the margin $\eta$, the vertical axis is the loss



# SVMs for classification

- The overall objective has the form

$$\min_{\mathbf{w},w_0} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N}(1 - y_i f(\mathbf{x}_i))_+$$

- This is non-differentiable, because of the max term. However, by introducing slack variables $\xi_i$, one can show that this is equivalent to solving

$$\min_{\mathbf{w},w_0,\boldsymbol{\xi}} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N}\xi_i \quad \text{s.t.} \quad \xi_i \geq 0,\ y_i(\mathbf{x}_i^T\mathbf{w} + w_0) \geq 1 - \xi_i, i = 1:N$$

- This is a quadratic program in $N + D + 1$ variables, subject to $O(N)$ constraints. We can eliminate the primal variables $\mathbf{w}$, $w_0$ and $\xi_i$, and just solve the $N$ dual variables, which correspond to the Lagrange multipliers for the constraints. Standard solvers take $O(N^3)$ time

# SVMs for classification

- The solution involves constructing a dual problem where a Lagrange multiplier $\lambda_i$ is associated with every constraint in the primary problem
- One can show that the solution has the form

$$\hat{\mathbf{w}} = \sum_i \alpha_i \mathbf{x}_i$$

- $\alpha_i = \lambda_i y_i$ and where $\boldsymbol{\alpha}$ is sparse (because of the hinge loss)
- The $\mathbf{x}_i$ for which $\alpha_i > 0$ are called support vectors; these are points which are either incorrectly classified, or are classified correctly but are on or inside the margin

# SVMs for classification

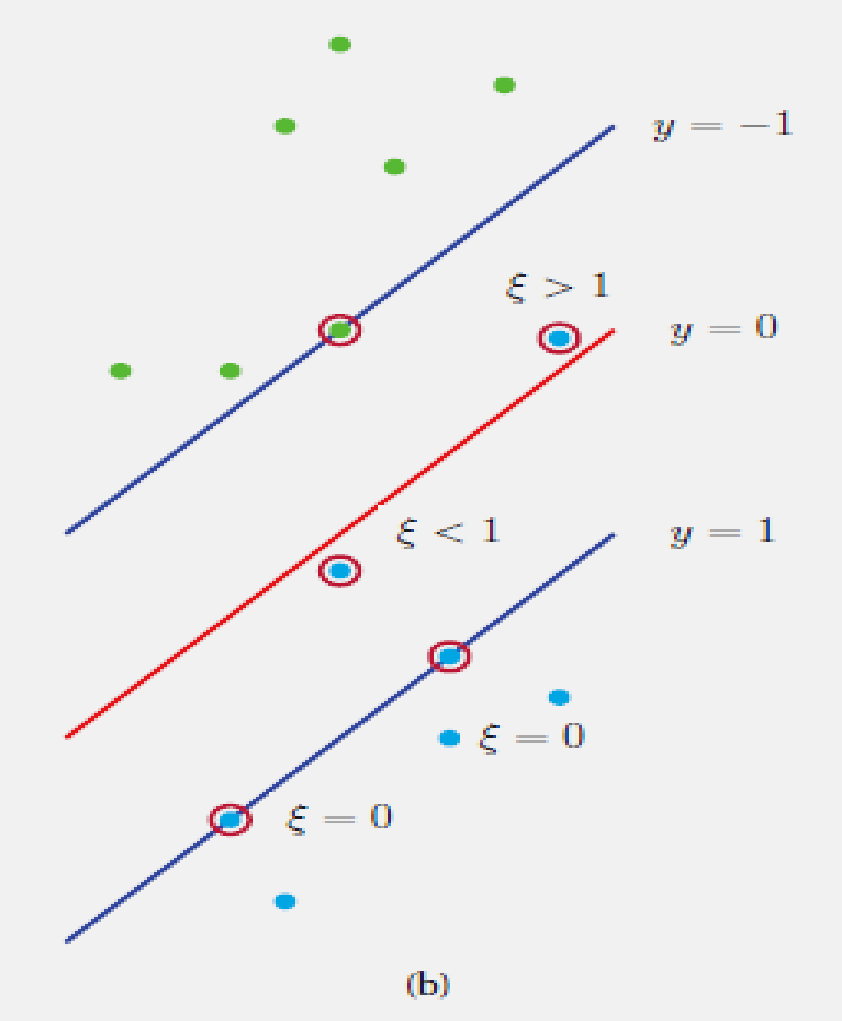



Illustration of the soft margin principle. Points with circles around them are support vectors. We also indicate the value of the corresponding slack variables.

# SVMs for classification

- At test time, prediction is done using

$$\hat{y}(\mathbf{x}) = \text{sgn}(f(\mathbf{x})) = \text{sgn}\left(\hat{w}_0 + \hat{\mathbf{w}}^T\mathbf{x}\right)$$

- Using the kernel trick we have

$$\hat{y}(\mathbf{x}) = \text{sgn}\left(\hat{w}_0 + \sum_{i=1}^{N} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x})\right)$$

This takes $O(sD)$ time to compute, where $s \leq N$ is the number of support vectors. This depends on the sparsity level, and hence on the regularizer $C$

# The large margin principle

- In this section, we derive the Equation form a completely different perspective.

$$\mathbf{x} = \mathbf{x}_\perp + r\frac{\mathbf{w}}{||\mathbf{w}||}$$

- where $r$ is the distance of $\mathbf{x}$ from the decision boundary whose normal vector is $\mathbf{w}$, and $\mathbf{x}_\perp$ is the orthogonal projection of $\mathbf{x}$ onto this boundary

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 = (\mathbf{w}^T\mathbf{x}_\perp + w_0) + r\frac{\mathbf{w}^T\mathbf{w}}{||\mathbf{w}||}$$

Now $f(\mathbf{x}_\perp) = 0$ so $0 = \mathbf{w}^T\mathbf{x}_\perp + w_0$. Hence $f(\mathbf{x}) = r\frac{\mathbf{w}^T\mathbf{w}}{\sqrt{\mathbf{w}^T\mathbf{w}}}$, and $r = \frac{f(\mathbf{x})}{||\mathbf{w}||}$

# The large margin principle

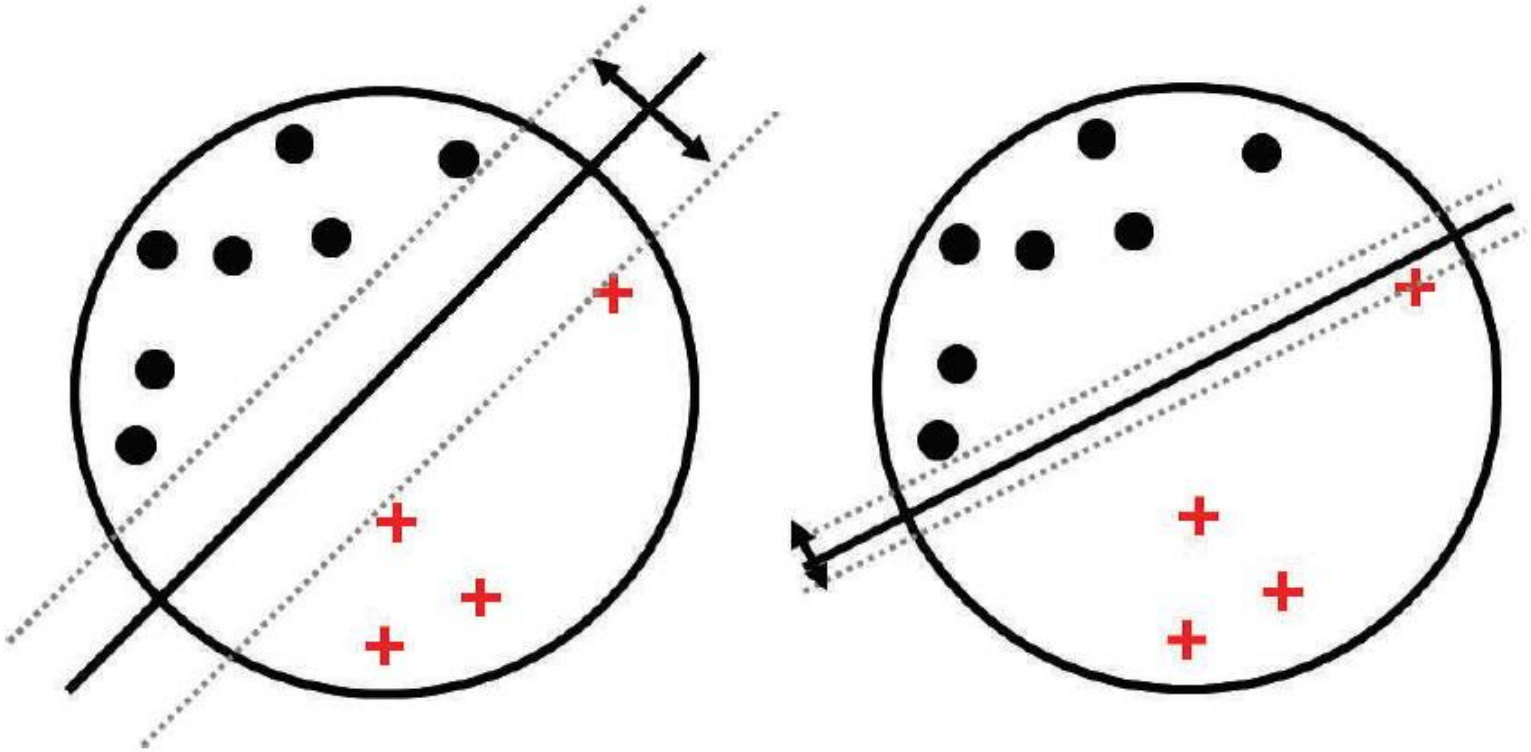
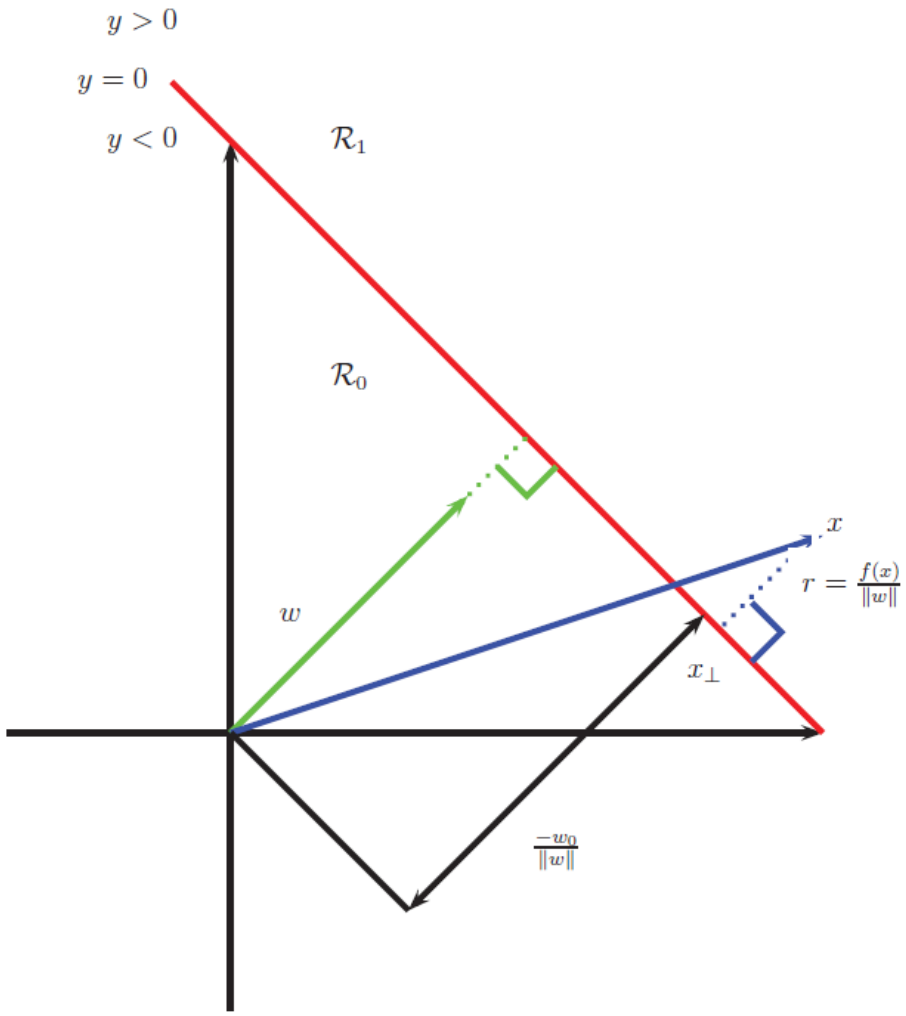


Illustration of the large margin principle
Left: a separating hyper-plane with large margin
Right: a separating hyper-plane with small margin

# The large margin principle

Illustration of the geometry of a linear decision boundary in 2d. A point $\mathbf{x}$ is classified as belonging in decision region $R_1$ if $f(\mathbf{x}) > 0$, otherwise it belongs in decision region $R_0$; here $f(\mathbf{x})$ is known as a **discriminant function**. The decision boundary is the set of points such that $f(\mathbf{x}) = 0$. $\mathbf{w}$ is a vector which is perpendicular to the decision boundary. The term $w_0$ controls the distance of the decision boundary from the origin. The signed distance of $\mathbf{x}$ from its orthogonal projection onto the decision boundary, $\boldsymbol{x}_\perp$, is given by $f(\mathbf{x})/||\mathbf{w}||$.



# The large margin principle

- We would like to make this distance $r = f(\mathbf{x})/||\mathbf{w}||$ as large as possible
- Intuitively, the best one to pick is the one that maximizes the margin, i.e., the perpendicular distance to the closest point
- In addition, we want to ensure each point is on the correct side of the boundary, hence we want $f(\mathbf{x}_i)\, y_i > 0$.
- So our objective becomes

$$\max_{\mathbf{w}, w_0} \min_{i=1}^{N} \frac{y_i(\mathbf{w}^T \mathbf{x}_i + w_0)}{||\mathbf{w}||}$$

# The large margin principle

- Note that by rescaling the parameters using $\mathbf{w} \rightarrow k\mathbf{w}$ and $w_0 \rightarrow kw_0$, we do not change the distance of any point to the boundary, since the $k$ factor cancels out when we divide by $||\mathbf{w}||$.
- Therefore let us define the scale factor such that $y_i f_i = 1$ for the point that is closest to the decision boundary
- We therefore want to optimize

$$\min_{\mathbf{w},w_0} \frac{1}{2}||\mathbf{w}||^2 \quad \text{s.t.} \quad y_i(\mathbf{w}^T\mathbf{x}_i + w_0) \geq 1, i = 1:N$$

- The constraint says that we want all points to be on the correct side of the decision boundary with a margin of at least 1