Linear Classifiers

Previous lectures introduced the **Bayes Classifier**:

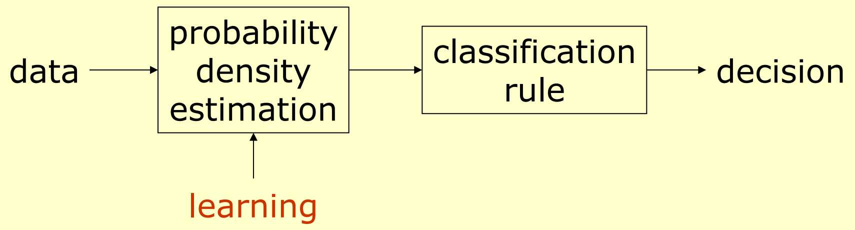
- Optimal accuracy in terms of minimizing the classification error probability.
- If the *probability distribution* is appropriate for the *novel* data.

In real world applications, it is very difficult to obtain the appropriate *probability distribution*.

Therefore, instead of modeling the whole feature space, we often prefer to learn the *discrimination function* directly.

Linear Classifiers

Bayes classifier:



data

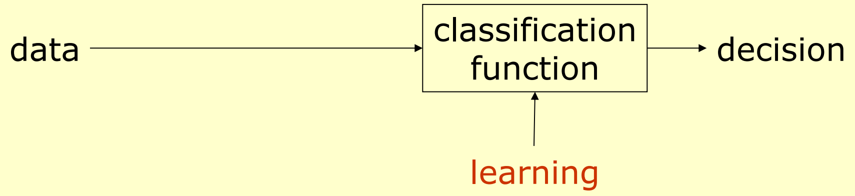
probability density estimation

classification rule

decision

learning

Linear Classifier:



data

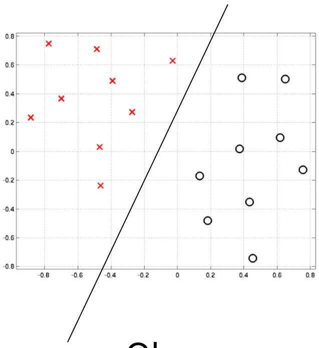
classification function

decision

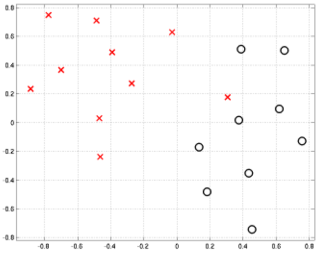
learning

Linear Classifiers

Requirement: The data must be linearly separable.



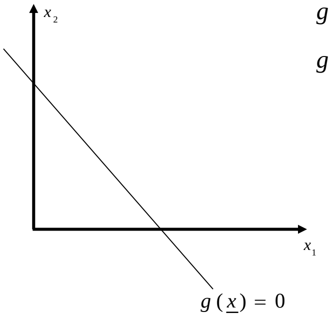
Ok



Not ok!

There is no line that can separate both classes!

Linear Classifiers



$$g(\underline{x}) \equiv \underline{w}^T \underline{x} + w_0 = 0 \qquad g : \mathbb{R}^2 \to \mathbb{R}$$

$$g(\underline{x}) \equiv w_1 x_1 + w_2 x_2 + \dots + w_l x_l + w_0 = 0$$

↑

in l dimensions

If $\underline{x}_1$ and $\underline{x}_2$ are two points on the decision hyperplane:

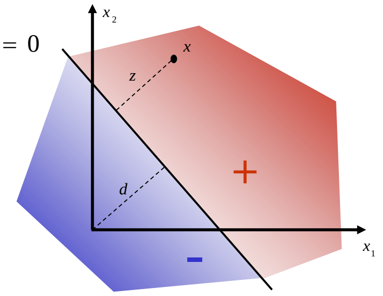
$$\underline{w}^T \underline{x}_1 + w_0 = \underline{w}^T \underline{x}_2 + w_0$$

$$\boxed{\Rightarrow \quad \underline{w}^T (\underline{x}_1 - \underline{x}_2) = 0}$$

hence w is perpendicular to the hyperplane

Linear Classifiers

$$g(\underline{x}) \equiv \underline{w}^T \underline{x} + w_0 = 0$$



$$z = \frac{|g(\underline{x})|}{\|\underline{w}\|}$$

g(x̲) is a measure of the distance from the hyperplane to x̲.

Its sign marks on which side of the hyperplane x̲ is.

$$d = \frac{|w_0|}{\|\underline{w}\|}$$

If there is no axis intercept the hyperplane passes through the origin.

Linear Classifier: Margin Computation

Recall $g(x) \equiv w^T x + w_0 = 0$

The direction normal to the hyperplane is given by: w

Hence,

$$x = x_p + \underset{\uparrow}{d}\frac{w}{\|w\|}$$

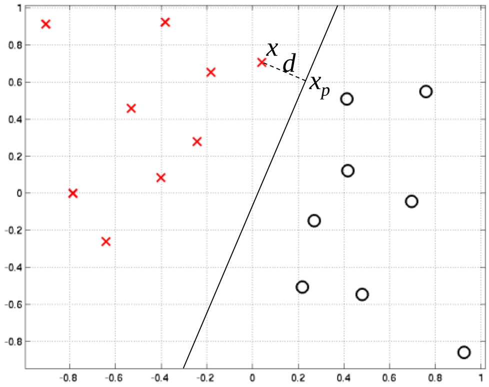
signed distance

$$g(x) = w^T(x_p + d\frac{w}{\|w\|}) + w_0$$

$$\Rightarrow g(x) = w^T x_p + w_0 \quad + d\frac{w^T w}{\|w\|}$$

$$\Rightarrow g(x) = d\frac{w^T w}{\|w\|} = d\|w\|$$

$$\Rightarrow d = \frac{g(x)}{\|w\|}$$



The Perceptron

7

The Perceptron is a learning algorithm that *adjusts the weights* w_i of its weight vector $\underline{w}$ such that for all examples $\underline{x}_i$:

$$\underline{w}^T \underline{x} > 0 \quad \forall \underline{x}_i \in \omega_1$$

$$\underline{w}^T \underline{x} < 0 \quad \forall \underline{x}_i \in \omega_2$$

It is assumed that the problem is linearly separable. Hence this vector $\underline{w}$ exists.

Here, the intercept is included in w : $\underline{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_l \\ w_0 \end{bmatrix} \quad \underline{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_l \\ 1 \end{bmatrix}$

$$\Rightarrow \qquad g(\underline{x}) \equiv \underline{w}^T \underline{x} = 0$$

The Perceptron

8

- $\underline{w}$ must minimize the classification error.
- $\underline{w}$ is found using an *optimization algorithm*.

General steps towards a classifier:

1. Define a cost function to be minimized.
2. Choose an algorithm to minimize it.
3. The minimum corresponds to a solution.

The Perceptron Cost Function

Goal:

$$\underline{w}^T \underline{x} > 0 \quad \forall \underline{x} \in \omega_1$$

$$\underline{w}^T \underline{x} < 0 \quad \forall \underline{x} \in \omega_2$$

Cost function: $J(\underline{w}) = \sum_{x \in Y} \delta_x \underline{w}^T \underline{x}$

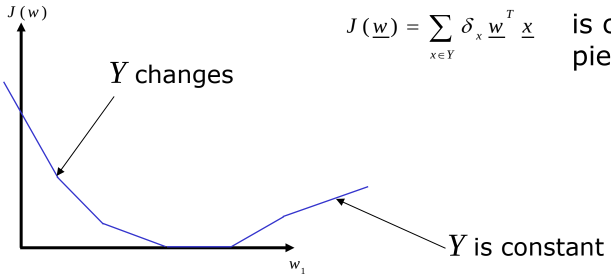
Y: subset of the training vectors which are **misclassified** by the hyperplane defined by w.

$\delta_{x_i} = -1$ if $\underline{x}_i \in \omega_1$ but is classified in ω_2

$\delta_{x_i} = +1$ if $\underline{x}_i \in \omega_2$ but is classified in ω_1

$$\Rightarrow \delta_x \underline{w}^T \underline{x} > 0 \quad \forall \underline{x} \in Y \quad \Rightarrow \quad \begin{array}{ll} J(\underline{w}) > 0 & \forall \underline{w} : Y \neq \varnothing \\ J(\underline{w}) = 0 & \text{if } Y = \varnothing \end{array}$$

The Perceptron Algorithm



$J(\underline{w}) = \sum_{x \in Y} \delta_x \underline{w}^T \underline{x}$ is continuous and piecewise linear.

$J(\underline{w})$ is minimized by *gradient descent:*

(update w by taking steps that are proportional to the negative of the gradient of the cost function J(w))

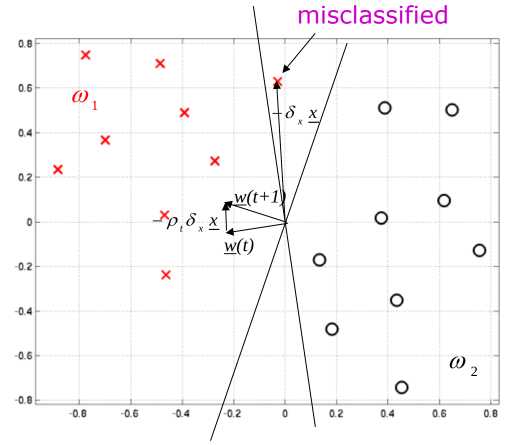
$$\underline{w}(t+1) = \underline{w}(t) + \Delta \underline{w} \quad \Rightarrow \quad \Delta \underline{w} = -\rho_t \frac{\partial J(\underline{w})}{\partial \underline{w}}$$

$$\frac{\partial J(\underline{w})}{\partial \underline{w}} = \frac{\partial}{\partial \underline{w}} \left(\sum_{x \in Y} \delta_x \underline{w}^T \underline{x} \right) = \sum_{x \in Y} \delta_x \underline{x}$$

$$\boxed{\underline{w}(t+1) = \underline{w}(t) - \rho_t \sum_{x \in Y} \delta_x \underline{x}}$$

The Perceptron Algorithm

Example:



$$\underline{w}(t+1) = \underline{w}(t) - \rho_t \sum_{x\in Y} \delta_x \underline{x}$$

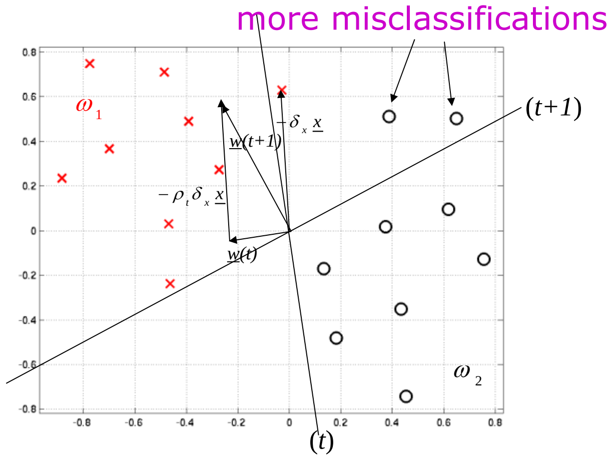
Here, δ_x =-1 because $\underline{x} \in \omega_1$

Here, ρ_t=0.2

The Perceptron Algorithm

Example:

Here, ρ_t=1



Note that ρ_t must be chosen carefully, if it is too large, more errors will occur.

ρ_t is a critical parameter of the algorithm !

The Perceptron Algorithm

13

The perceptron converges in a **finite number** of iterations to a solution if:

$$\lim_{t \to \infty} \sum_{k=0}^{t} \rho_k \to \infty$$

$$\lim_{t \to \infty} \sum_{k=0}^{t} \rho_k^2 < +\infty$$

$$\text{e.g. : } \rho_t = \frac{c}{t}$$

ρ_t is set to be large at the beginning and gets smaller and smaller as the iterations proceed.

The perceptron stops as soon as the last misclassification disappears: Is this optimal?

Perceptron: Online Learning

14

The misclassified training examples can be used *cyclically*, one after the other.

The examples are reused until they are all classified correctly.

$$\underline{w}(t+1) = \underline{w}(t) + \rho_t \underline{x}_t \quad \text{if} \quad \underline{w}(t)^T \underline{x}_t < 0 \quad \text{and} \quad \underline{x}_t \in \omega_1$$

$$\underline{w}(t+1) = \underline{w}(t) - \rho_t \underline{x}_t \quad \text{if} \quad \underline{w}(t)^T \underline{x}_t > 0 \quad \text{and} \quad \underline{x}_t \in \omega_2$$

$$\underline{w}(t+1) = \underline{w}(t) \qquad \text{otherwise}$$

This training of the Perceptron is called "**reward and punishment algorithm**".

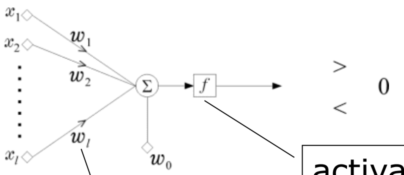
The Perceptron as a Neural Network

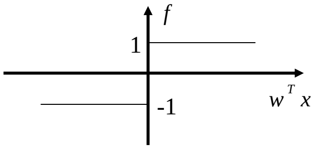
Once the perceptron is trained, it is used to perform the classification:

$$\text{if} \quad w^T x > 0 \quad \text{assign } x \text{ to } \omega_1$$

$$\text{if} \quad w^T x < 0 \quad \text{assign } x \text{ to } \omega_2$$

The perceptron is the simplest form of a "Neural Network":





synaptic weights

activation function

Least Squares Methods

Linear classifiers are attractive because:

- They are simple and
- computationally efficient.

The Perceptron is used in the case where the training examples are linearly separable.

Can we still use a simple linear classifier where the training examples are NOT linearly separable ?

Least Squares Methods

We want that the difference between the output of the linear classifier: $\underline{w}^T \underline{x}$

and the desired outputs (class labels): to be small.

$$y = +1 \text{ if } \underline{x} \in \omega_1$$
$$y = -1 \text{ if } \underline{x} \in \omega_2$$

What does small mean ?

We will describe two criterions:

1. **Mean** square error estimation, and
2. **Sum** of square error estimation.

Mean Square Error

Cost function: $J(\underline{w}) = E\left[\left(y - \underline{w}^{\mathrm{T}}\underline{x}\right)^2\right]$

Find: $\hat{\underline{w}} = \arg\min_{w} J(\underline{w})$

$J(\underline{w})$ is minimum when $\dfrac{\partial J(\underline{w})}{\partial \underline{w}} = 0$

$$\frac{\partial J(\underline{w})}{\partial \underline{w}} = -2E\left[\left(y - \underline{w}^{\mathrm{T}}\underline{x}\right)\underline{x}^{\mathrm{T}}\right]$$

$$= -2E\left[\underline{x}^{\mathrm{T}} y\right] + \underline{w}^{\mathrm{T}} 2E\left[\underline{x}\,\underline{x}^{T}\right]$$

$$\Rightarrow \boxed{\hat{\underline{w}} = E\left[\underline{x}\,\underline{x}^{\mathrm{T}}\right]^{-1} E\left[\underline{x}^{\mathrm{T}} y\right]}$$

$$E[\underline{x}\,\underline{x}^T] = \begin{bmatrix} E[x_1x_1] & E[x_1x_2]\ldots & E[x_1x_l] \\ \ldots\ldots\ldots\ldots & \ldots\ldots\ldots\ldots & \ldots\ldots\ldots\ldots \\ E[x_lx_1] & E[x_lx_2]\ldots & E[x_lx_l] \end{bmatrix} = R_x$$

is the auto-correlation matrix

$$E[\underline{x}^{\mathrm{T}} y] = \begin{bmatrix} E[x_1y] \\ \ldots \\ E[x_ly] \end{bmatrix}$$

is the cross-correlation vector

Mean Square Error

Problem: $E\left[\underline{x}\,\underline{x}^T\right] = ?$ $E\left[\underline{x}\,y\right] = ?$

Computing $E\left[\underline{x}\,\underline{x}^T\right]$ and $E\left[\underline{x}\,y\right]$ requires knowledge of the probability distribution function of the feature vectors.

If the pdf is known or we have a good method to estimate it, we might as well use a Bayesian classifier, which minimizes the classification error !

Here, we want to find a similar result *without* having to know the probability distribution.

This leads us to the minimum *sum* of squares estimation.

Sum of Squares Error

Instead of $J(\underline{w}) = E\left[\left(y - \underline{w}^{\mathrm{T}}\underline{x}\right)^2\right]$ use the following

cost function: $$\boxed{J(\underline{w}) = \sum_{i=1}^{N}\left(y_i - \underline{w}^{\mathrm{T}}\underline{x}_i\right)^2}$$

$J(\underline{w})$ is minimum when $\dfrac{\partial J(\underline{w})}{\partial \underline{w}} = \underline{0}$

$$\frac{\partial J(\underline{w})}{\partial \underline{w}} = -2\sum_{i=1}^{N}\left(y_i - \underline{w}^{\mathrm{T}}\underline{x}_i\right)\underline{x}_i^{\mathrm{T}}$$

$$= -2\sum_{i=1}^{N} y_i\,\underline{x}_i^{\mathrm{T}} + 2\,\underline{w}^{\mathrm{T}}\left(\sum_{i=1}^{N}\underline{x}_i\,\underline{x}_i^{\mathrm{T}}\right)$$

$$= -2X^T\underline{y} + 2X^TX\,\underline{w}$$

$X^T = [\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_N]$

X is a $\mathrm{N}\mathrm{x}l$ matrix, each row is the transpose on one l-dimensional training vector (--> X is $\mathrm{N}\mathrm{x}l$).
X is often referenced as ***Design Matrix***

$\underline{y}^T = [y_1, y_2, \ldots, y_N]$

desired responses column vector.

Sum of Squares Error

$$\frac{\partial J(\underline{w})}{\partial \underline{w}} = 2X^T y - 2X^T X \underline{w}$$

$$\frac{\partial J(\underline{w})}{\partial \underline{w}} = 0 \quad \Rightarrow \quad X^T X \hat{\underline{w}} = X^T y$$

$$\Rightarrow \quad \boxed{\hat{\underline{w}} = \left(X^T X\right)^{-1} X^T y}$$

$$X^+ \equiv \left(X^T X\right)^{-1} X^T$$

X^+ is the lxN Moore-Penrose *Pseudo-inverse* of the Nxl matrix X.

$$\Rightarrow \quad \boxed{\hat{\underline{w}} = X^+ y}$$

If X is a square matrix: $X^+ = X^{-1}$

Sum of Squares Error

Recall that the objective is to solve $Xw = y$.

If $N > l$, which is often the case in Pattern Recognition, then there are more equations than unknowns: the system is <u>over determined</u>.

In general, there is no solution which satisfies all equations.

The solution $\hat{w} = X^+ y$ corresponds to the minimum sum of square solution: $\min \left\| y - X\hat{w} \right\|^2$

Sum of Squares Error - Example

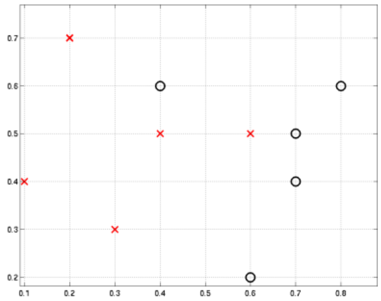
Data:

$$\omega_1 : \begin{bmatrix}0.4\\0.5\end{bmatrix}, \begin{bmatrix}0.6\\0.5\end{bmatrix}, \begin{bmatrix}0.1\\0.4\end{bmatrix}, \begin{bmatrix}0.2\\0.7\end{bmatrix}, \begin{bmatrix}0.3\\0.3\end{bmatrix}$$

$$\omega_2 : \begin{bmatrix}0.4\\0.6\end{bmatrix}, \begin{bmatrix}0.6\\0.2\end{bmatrix}, \begin{bmatrix}0.7\\0.4\end{bmatrix}, \begin{bmatrix}0.8\\0.6\end{bmatrix}, \begin{bmatrix}0.7\\0.5\end{bmatrix}$$

N = 10,

l = 2+1 = 3



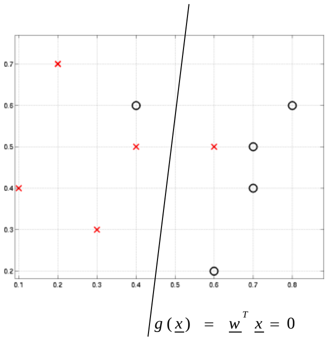
Task: minimize $J(\underline{w}) = \sum_{i=1}^{N} \left(y_i - \underline{w}^T \underline{x}_i \right)^2$

$$\Rightarrow \boxed{\hat{\underline{w}} = \left(X^T X \right)^{-1} X^T y}$$

Sum of Squares Error - Example

$$\boxed{\underline{w} = (X^T X)^{-1} X^T \underline{y} = \ ?}$$

$$X = \begin{bmatrix} 0.4 & 0.5 & 1 \\ 0.6 & 0.5 & 1 \\ 0.1 & 0.4 & 1 \\ 0.2 & 0.7 & 1 \\ 0.3 & 0.3 & 1 \\ 0.4 & 0.6 & 1 \\ 0.6 & 0.2 & 1 \\ 0.7 & 0.4 & 1 \\ 0.8 & 0.6 & 1 \\ 0.7 & 0.5 & 1 \end{bmatrix}, \quad \underline{y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$



$g(\underline{x}) = \underline{w}^T \underline{x} = 0$

$$X^T X = \begin{bmatrix} 2.8 & 2.24 & 4.8 \\ 2.24 & 2.41 & 4.7 \\ 4.8 & 4.7 & 10 \end{bmatrix},$$

$$X^T \underline{y} = \begin{bmatrix} -1.6 \\ 0.1 \\ 0.0 \end{bmatrix}$$

$$\Rightarrow \quad \boxed{\underline{w} = (X^T X)^{-1} X^T \underline{y} = \begin{bmatrix} -3.13 \\ 0.24 \\ 1.34 \end{bmatrix}}$$

The Perceptron Cost Function

Goal:

$$\underline{w}^T \underline{x} > 0 \quad \forall \underline{x} \in \omega_1$$

$$\underline{w}^T \underline{x} < 0 \quad \forall \underline{x} \in \omega_2$$

Cost function: $J(\underline{w}) = \sum_{x \in Y} \delta_x \underline{w}^T \underline{x}$

Y: subset of the training vectors which are **misclassified** by the hyperplane defined by $\underline{w}$.

$$\delta_{x_i} = -1 \quad \text{if } \underline{x}_i \in \omega_1 \text{ but is classified in } \omega_2$$

$$\delta_{x_i} = +1 \quad \text{if } \underline{x}_i \in \omega_2 \text{ but is classified in } \omega_1$$

$$\Rightarrow \delta_x \underline{w}^T \underline{x} > 0 \quad \forall \underline{x} \in Y \quad \Rightarrow \quad \begin{array}{ll} J(\underline{w}) > 0 & \forall \underline{w} : Y \neq \emptyset \\ J(\underline{w}) = 0 & \text{if } Y = \emptyset \end{array}$$

Linear Support Vector Machine

Goal:

$$\underline{w}^T \underline{x} + w_0 > 0 \quad \forall \underline{x} \in \omega_1$$

$$\underline{w}^T \underline{x} + w_0 < 0 \quad \forall \underline{x} \in \omega_2$$

So far, we have seen two classifiers with the same decision function: $g(\underline{x}) \equiv \underline{w}^T \underline{x} + w_0 = 0$

Their difference consisted in the cost function that was optimized to find the weights:

Perceptron: $J(\underline{w}) = \sum_{x \in Y} \delta_x \underline{w}^T \underline{x}$ mit $\begin{array}{ll} \delta_x = -1 & \text{if } \underline{x} \in \omega_1 \\ \delta_x = +1 & \text{if } \underline{x} \in \omega_2 \end{array}$

Sum of Squares: $\min_w \sum_{i=1}^{N} \left(y_i - \underline{w}^T \underline{x}_i - w_0 \right)^2$

Perceptron Problem

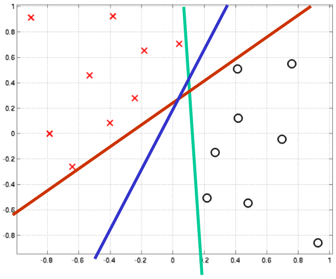
Perceptron: $J(\underline{w}) = \sum_{x \in Y} \delta_x \underline{w}^T \underline{x}$

$\delta_x = -1$ if $\underline{x} \in \omega_1$

$\delta_x = +1$ if $\underline{x} \in \omega_2$

Problem: There is an infinity of classifier that agree with the above criterion.

Example:



The one we want is the one that gives optimal *generalization* performance.

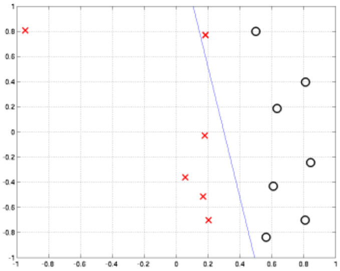
Which one is it ?

Sum of Squares Estimator Problem

Sum of Squares: $\min_{w} \sum_{i=1}^{N} \left(y_i - \underline{w}^T \underline{x}_i - w_0 \right)^2$

Problem: The estimator tries to place the hyperplane so that all the examples have the same distance from it (+1 for ω_1 and −1 for ω_2)

Example:



A single training example can pull the whole decision plane

Even in a linearly separable case, the optimal least squares estimator may get training errors !!!

Linear Support Vector Machine (SVM)

Is it possible to design a linear classifier better than the perceptron and the SSE?

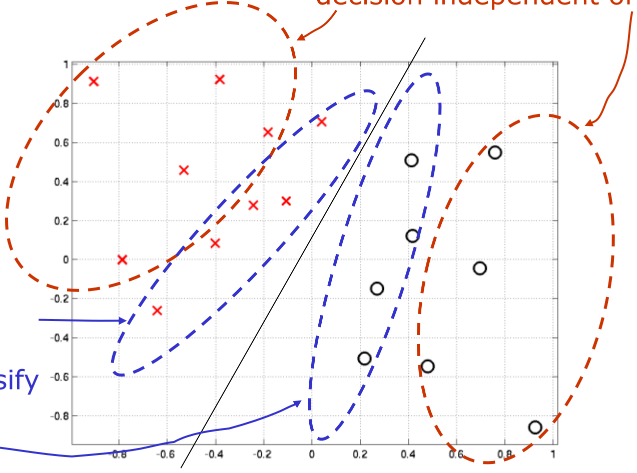
What are the criterions?

1. The decision surface should not be affected by examples far from it.
2. It should minimize the risk of error on unseen data (maximize generalization).
3. It should be unique : Not affected by initial values or optimization parameters (unlike for the perceptron).

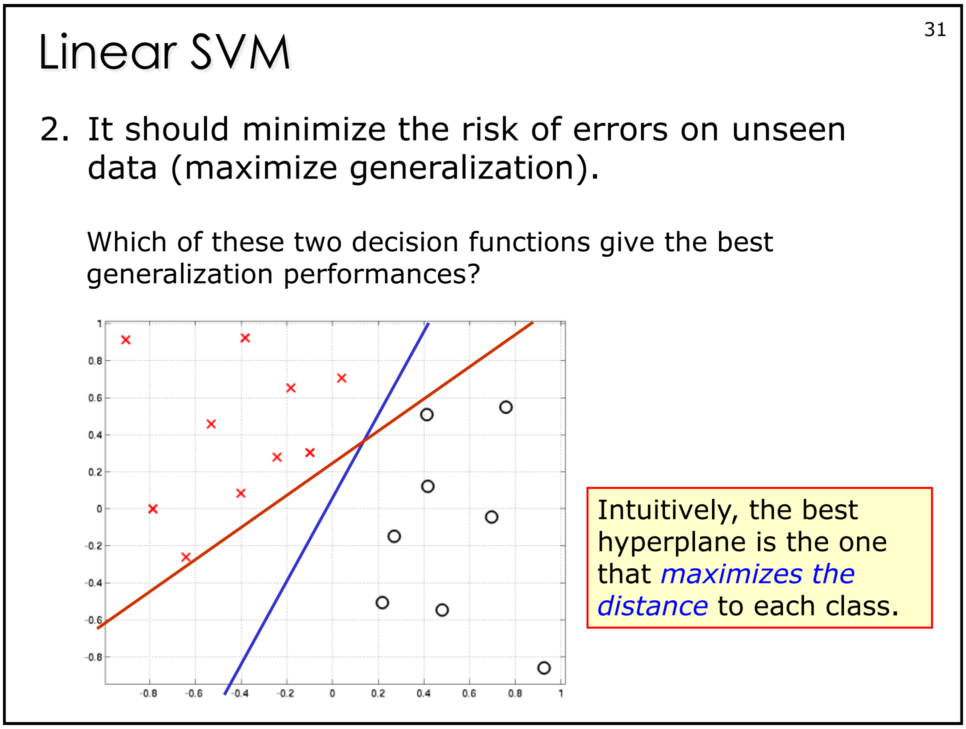
Linear SVM

1. The decision function should not be affected by examples far from it.

decision independent of



decision only dependent on difficult to classify examples



31

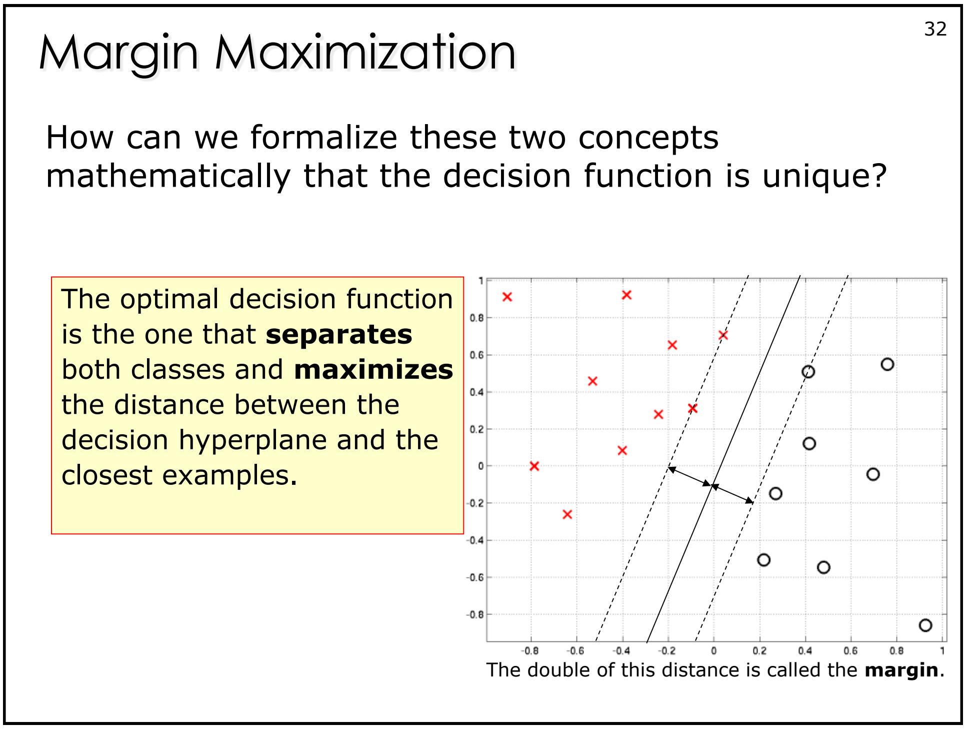
Linear SVM

2. It should minimize the risk of errors on unseen data (maximize generalization).

Which of these two decision functions give the best generalization performances?

1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

Intuitively, the best hyperplane is the one that *maximizes the distance* to each class.



32

Margin Maximization

How can we formalize these two concepts mathematically that the decision function is unique?

The optimal decision function is the one that **separates** both classes and **maximizes** the distance between the decision hyperplane and the closest examples.

1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

The double of this distance is called the **margin**.

Margin Computation

Recall $g(x) \equiv w^T x + w_0 = 0$

The direction normal to the hyperplane is given by: w

Hence,
$$x = x_p + \underset{\uparrow}{d} \frac{w}{\|w\|}$$

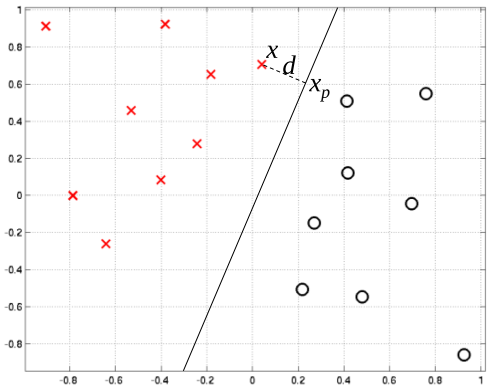
signed distance

$$g(x) = w^T (x_p + d \frac{w}{\|w\|}) + w_0$$

$$\Rightarrow g(x) = w^T x_p + w_0 \quad + d \frac{w^T w}{\|w\|}$$

$$\Rightarrow g(x) = d \frac{w^T w}{\|w\|} = d \|w\|$$

$$\Rightarrow d = \frac{g(x)}{\|w\|}$$



Linear SVM Learning

Now, we want to:

1. find $\underline{w}$ and w_0, such that the margin is maximized. $2|d| = 2\frac{|g(x)|}{\|w\|}$

2. scale $\underline{w}$ and w_0, such that $g(x)= +1$ for the *closest examples* of ω_1 and $g(x)= -1$ for the *closest examples* of ω_2.

   => then the margin is $2|d| = 2/\|w\|$

This is equivalent to:

$$\hat{w} = \min_w \frac{1}{2}\|w\|^2 \text{ subject to } \begin{cases} w^T x + w_0 \geq +1 & \forall x \in \omega_1 \\ w^T x + w_0 \leq -1 & \forall x \in \omega_2 \end{cases}$$

These closest examples, with $|g(x)| = 1$ are called **support vectors**.

Linear SVM

Note that:

1. This formulation provides a **unique** decision function, because there is only one that maximizes the separation between positive and negative examples.

2. This formulation assumes that the training vectors are separable. We will see in the next section how to address the non-separable case.

SVM Learning is a Constrained Optimization

Now, how to compute w and w_0 according to the criterion:

$$\underline{\hat{w}} = \arg\min_{w} \frac{1}{2}\left\|\underline{w}\right\|^2 \text{ subject to } \begin{cases} \underline{w}^T \underline{x} + w_0 \geq +1 & \forall\, \underline{x} \in \omega_1 \\ \underline{w}^T \underline{x} + w_0 \leq -1 & \forall\, \underline{x} \in \omega_2 \end{cases}$$

With labels y_i = +1 for *examples* of ω_1 and y_i = -1 for ω_2

this is equivalent to:

$$\underline{\hat{w}} = \arg\min_{w} \frac{1}{2}\underline{w}^T \underline{w} \text{ subject to } y_i\left(\underline{w}^T \underline{x}_i + w_0\right) \geq 1 \quad i = 1, \cdots, N$$

This is a *constrained* optimization.

Lagrange Multipliers

$$\underline{\hat{w}} = \arg\min_{w} \frac{1}{2}\underline{w}^T\underline{w} \ \text{ subject to } \ y_i\left(\underline{w}^T\underline{x}_i + w_0\right) \ge 1 \quad i = 1,\cdots,N$$

1. The cost function, $J(w) = w^T w$, is convex.
2. The constraints are linear.

- There is a unique solution,
- that can be found using the method of Lagrange Multipliers.

Lagrangian Function:

$$\mathrm{L}(\underline{w}, w_0, \lambda) = \frac{1}{2}\underline{w}^T\underline{w} - \sum_{i=1}^{N}\lambda_i\left[y_i\left(\underline{w}^T\underline{x}_i + w_0\right) - 1\right]$$

Constraint Optimization (insertion)

Problem: Given an objective function $f(x)$ to be optimized and let constraints be given by $h_k(x) = c_k$, moving constants to the left, ==> $h_k(x) - c_k = g_k(x)$. $f(x)$ and $g_k(x)$ must have continuous first partial derivatives

A Solution:

Lagrangian Multipliers $\quad 0 = \nabla_x f(x) + \Sigma \nabla_x \lambda_k g_k(x)$

or starting with the Lagrangian : $\quad L(x,\lambda) = f(x) + \Sigma \lambda_k g_k(x)$.

with $\nabla_x L(x,\lambda) = 0$.

Constrained Optimization in general

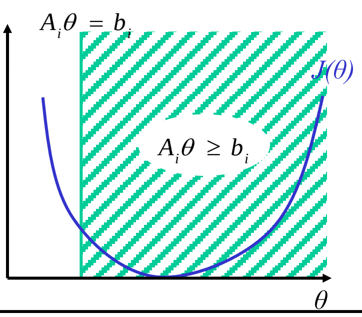
Objective: $\arg\min_{\theta} J(\theta)$ subject to $y_i(\underline{x}_i^T \underline{w} + w_0) \geq 1$ with $\theta = (w_0, \underline{w}^T)^T$

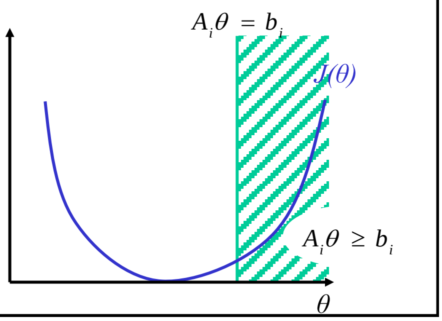
$$\Leftrightarrow A_i\theta \geq b_i$$

Lagrangian: $\mathrm{L}(\theta, \lambda) = J(\theta) - \sum_{i=1}^{N} \lambda_i (A_i\theta - b_i)$

Let us look at an example in 1 dimension.

There are two cases:



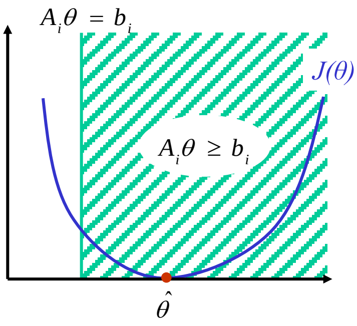


Constrained Optimization: First KKT Condition

Lagrangian: $\mathrm{L}(\theta, \lambda) = J(\theta) - \sum_{i=1}^{N} \lambda_i (A_i\theta - b_i)$

First case:

The minimum of $J(\theta)$ is *inside* the feasible region.



$\Rightarrow$ The constraint is *inactive* and plays no role.

As if it was an *unconstrained* problem.

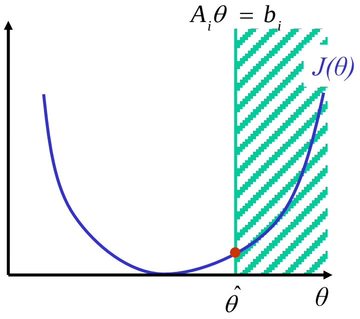
$$\boxed{\Rightarrow \lambda_i = 0}$$

First KKT Condition

Lagrangian: $\mathrm{L}(\theta,\lambda) = J(\theta) - \sum_{i=1}^{N} \lambda_i (A_i\theta - b_i)$

Second case:

The minimum of $J(\theta)$ is *outside* the feasible region.



=> The constraint is *active*.

The constraint minimum is at the *boundary* of the feasible region.

$$\boxed{\Rightarrow\ A_i\hat{\theta} - b_i = 0}$$

First KKT Condition

To summarize both cases, we have $\lambda_i = 0$ or $A_i\theta - b_i = 0$

This can be stated by the single condition:

$$\boxed{\lambda_i (A_i\hat{\theta} - b_i) = 0}$$

At the minimum, either the constraint is active or the Lagrangian multiplier is null.

This is the first *Karush-Kuhn-Tucker* condition.

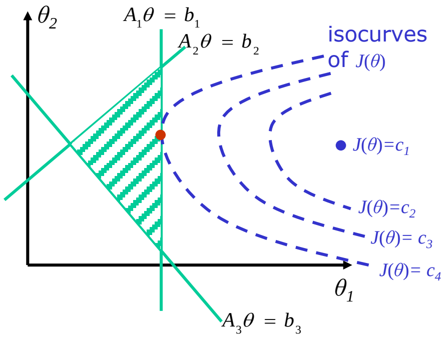
Let‘s now look at the second.

Second KKT Condition

Objective: $\arg\min_{\theta} J(\theta)$ subject to $A_i\theta \geq b_i$

Lagrangian: $\mathrm{L}(\theta,\lambda) = J(\theta) - \sum_{i=1}^{N} \lambda_i (A_i\theta - b_i)$

Let us look at an example in 2 dimensions:



$\min J(\theta)= c_1 < c_2 < c_3 < c_4$

The gradient of $J(\theta)$ is normal to the active constraints at the minimum: $\frac{\partial J(\hat{\theta})}{\partial \theta} = \lambda_1 A_1^T$

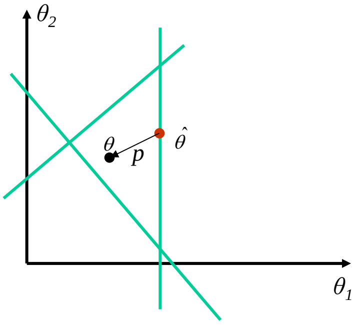
$$\boxed{\Rightarrow \frac{\partial \mathrm{L}(\hat{\theta},\lambda)}{\partial \theta} = 0}$$

Third KKT Condition

Assume a θ in the feasible region

$$\theta = \hat{\theta} + p \Rightarrow Ap = A\theta - A\hat{\theta} = A\theta - b \geq 0$$

Recall that $\frac{\partial J(\hat{\theta})}{\partial \theta} = A^T \lambda$



$$\Rightarrow p^T \frac{\partial J(\hat{\theta})}{\partial \theta} = p^T A^T \lambda$$

$p^T \frac{\partial J(\hat{\theta})}{\partial \theta} \geq 0$ because $\hat{\theta}$ is a minimizer

$$\Rightarrow p^T A^T \lambda \geq 0$$

$\boxed{\Rightarrow \lambda \geq 0}$ Third KKT condition

KKT Conditions

For the **problem** $\arg\min_{\theta} J(\theta)$ subject to $A_i\theta \geq b_i$

The Lagrangian is $\mathrm{L}(\theta,\lambda) = J(\theta) - \sum_{i=1}^{N} \lambda_i (A_i\theta - b_i)$

$\hat{\theta}$ is a minimizer if the three KKT conditions are satisfied:

KKT1: $\lambda_i (A_i\hat{\theta} - b_i) = 0$

KKT2: $\dfrac{\partial \mathrm{L}(\hat{\theta},\lambda)}{\partial \theta} = 0$

KKT3: $\lambda_i \geq 0$

KKT Conditions applied to the SVM

$$\hat{w} = \arg\min_{w} \frac{1}{2} w^T w \quad \text{subject to} \quad y_i\left(w^T x_i + w_0\right) \geq 1 \quad i = 1,\cdots,N$$

$$\Rightarrow \quad \mathrm{L}(w, w_0, \lambda) = \frac{1}{2} w^T w - \sum_{i=1}^{N} \lambda_i \left[y_i\left(w^T x_i + w_0\right) - 1 \right]$$

KKT2: $\dfrac{\partial \mathrm{L}(\hat{w},\hat{w}_0,\lambda)}{\partial w} = 0 \quad \Rightarrow \quad \hat{w} = \sum_{i=1}^{N} \lambda_i y_i x_i$

The hyperplane, defined through *w*, is a linear combination of the examples.

KKT2: $\dfrac{\partial \mathrm{L}(\hat{w},\hat{w}_0,\lambda)}{\partial w_0} = 0 \quad \Rightarrow \quad \sum_{i=1}^{N} \lambda_i y_i = 0$

Can be used to check your implementation.

KKT1: $\lambda_i \left[y_i\left(\hat{w}^T x_i + \hat{w}_0\right) - 1 \right] = 0$

The support vectors, for which $\lambda_i \neq 0$, are those for which the constrain is active, i.e. $y_i\left(\hat{w}^T x_i + \hat{w}_0\right) = 1$

Primal and Dual Problems

The number of support vectors: $N_S \le N$

If the features are discriminative: $N_S \ll N$

$$\min_{w} \frac{1}{2} \underline{w}^T \underline{w} \text{ subject to } y_i \left(\underline{w}^T \underline{x}_i + w_0 \right) \ge 1 \quad i = 1, \cdots, N$$

This is the primal problem, it can be solved efficiently using its dual formulation:

$$\max_{w, w_0, \lambda} \mathrm{L}(\underline{w}, w_0, \underline{\lambda}) \text{ subject to } \left.\begin{aligned} \underline{w} &= \sum_{i=1}^{N} \lambda_i y_i \underline{x}_i \\ \sum_{i=1}^{N} \lambda_i y_i &= 0 \\ \lambda &\ge 0 \end{aligned}\right\} \text{KKT conditions}$$

Learning SVM using the Dual Problem

$$\mathrm{L}(\underline{w}, w_0, \lambda) = \frac{1}{2} \underline{w}^T \underline{w} - \sum_{i=1}^{N} \lambda_i \left[y_i \left(\underline{w}^T \underline{x}_i + w_0 \right) - 1 \right]$$

$$\mathrm{L}(\underline{w}, w_0, \lambda) = \frac{1}{2} \underline{w}^T \underline{w} - \sum_{i=1}^{N} \left[\left(\underline{w}^T \lambda_i y_i \underline{x}_i + \lambda_i y_i w_0 \right) - \lambda_i \right]$$

$$\mathrm{L}(\underline{w}, w_0, \lambda) = \frac{1}{2} \underline{w}^T \underline{w} - \underline{w}^T \sum_{i=1}^{N} \lambda_i y_i \underline{x}_i - w_0 \sum_{i=1}^{N} \lambda_i y_i + \sum_{i=1}^{N} \lambda_i$$

$$\underline{w} = \sum_{i=1}^{N} \lambda_i y_i \underline{x}_i$$

$$\sum_{i=1}^{N} \lambda_i y_i = 0$$

$$\Rightarrow \quad \mathrm{L}(\underline{w}, w_0, \lambda) = -\frac{1}{2} \sum_{i,j}^{N} \lambda_i \lambda_j y_i y_j \underline{x}_i^T \underline{x}_j + \sum_{i}^{N} \lambda_i$$

Learning SVM using the Dual Problem

$$\mathrm{L}(\underline{w}, w_0, \underline{\lambda}) = \frac{1}{2}\underline{w}^T\underline{w} - \sum_{i=1}^{N}\lambda_i\left[y_i\left(\underline{w}^T\underline{x}_i + w_0\right) - 1\right]$$

$$\underline{w} = \sum_{i=1}^{N}\lambda_i y_i \underline{x}_i$$

$$\sum_{i=1}^{N}\lambda_i y_i = 0$$

$$\Rightarrow \mathrm{L}(\cancel{\underline{w}, w_0}, \underline{\lambda}) = -\frac{1}{2}\sum_{i,j}^{N}\lambda_i\lambda_j y_i y_j \underline{x}_i^T\underline{x}_j + \sum_i^N \lambda_i$$

$$\hat{\underline{\lambda}} = \arg\max_{\underline{\lambda}} -\frac{1}{2}\sum_{i,j}^{N}\lambda_i\lambda_j y_i y_j \underline{x}_i^T\underline{x}_j + \sum_i^N \lambda_i \quad \text{subject to} \quad \sum_{i=1}^{N}\lambda_i y_i = 0$$

$$\lambda_i \geq 0$$

We only need to solve with respect to λ !

Learning SVM is a Quad. Prog. Probl.

$$\hat{\lambda} = \arg\max_{\lambda}\left(\sum_i^N \lambda_i - \frac{1}{2}\sum_{i,j}^{N}\lambda_i\lambda_j y_i y_j x_i^T x_j\right) \quad \text{subject to} \quad \sum_{i=1}^{N}\lambda_i y_i = 0 \quad \lambda_i \geq 0$$

This is a standard problem in optimization theory called Convex Quadratic Programming.

Don`t try to program this yourself ;-)

In MATLAB, use `quadprog`, in Scilab, `quapro`

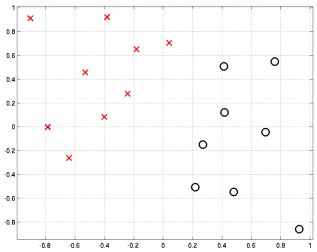
In C++ use the library `OOQP`.

Once $\hat{\lambda}$ is found:

$$\hat{w} = \sum_{i=1}^{N}\hat{\lambda}_i y_i x_i \qquad \rightarrow \hat{w}$$

$$\lambda_i\left[y_i\left(\hat{w}^T x_i + \hat{w}_0\right) - 1\right] = 0 \qquad \rightarrow \hat{w}_0$$

SVM with Non-Separable Classes

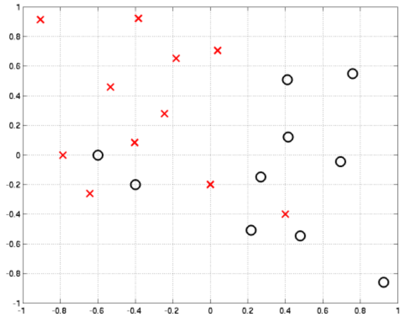


So far, we dealt with the easy case of separable classes.

Now what do we do in this case ?

What‵s the margin here ?

It is impossible to draw a separating hyperplane.



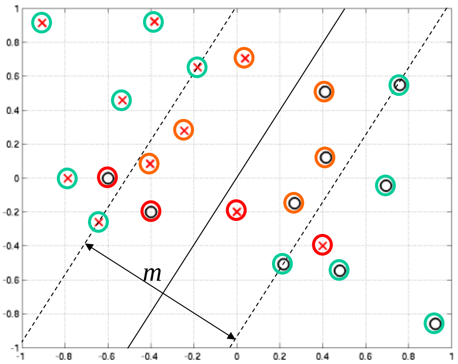
Soft Margin

As before, the margin is the distance between the hyperplanes defined by

$$w^T x + w_0 = \pm 1$$

The margin is soft if one of the points violates

$$y_i \left(w^T x_i + w_0 \right) \geq 1$$



There are 3 types of points:

- ⊖ outside the band and correctly classified $y_i \left(w^T x_i + w_0 \right) \geq 1$
- ⊖ inside the band and correctly classified $0 \leq y_i \left(w^T x_i + w_0 \right) < 1$
- ⊖ misclassified $y_i \left(w^T x_i + w_0 \right) < 0$

Slack Variables

- Outside the band and correctly classified $y_i\left(w^T x_i + w_0\right) \geq 1$
- Inside the band and correctly classified $0 \leq y_i\left(w^T x_i + w_0\right) < 1$
- Misclassified $y_i\left(w^T x_i + w_0\right) < 0$

The 3 cases can be addressed by a single constraint: $y_i\left(w^T x_i + w_0\right) \geq 1 - \xi_i$

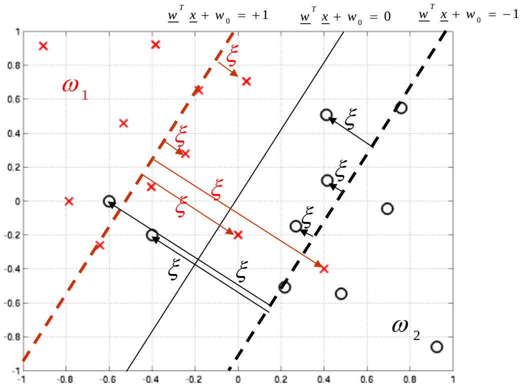
slack variables

- Outside the band and correctly classified $\xi_i = 0$
- Inside the band and correctly classified $0 < \xi_i \leq 1$
- Misclassified $\xi_i > 1$

ξ measures the deviation of a data point from the ideal condition of pattern separability.

Slack Variables

ξ measures the deviation of a data point from the ideal condition of pattern separability.



→ New Goal: $\min \left\|\underline{w}\right\|$ and $\min \#\left[\xi_i > 0\right]$

Non-Separable SVM Objective

→ New Goal: $\min \|\underline{w}\|$ and $\min \#[\xi_i > 0]$

How can we do that mathematically?

Minimize the average training set error:

$$\min \frac{1}{2}\|\underline{w}\|^2 + C\sum_{i=1}^{N} I(\xi_i)$$

Trade off parameter

indicator function

$$I(\xi_i) = \begin{cases} 1 & \xi_i > 0 \\ 0 & \xi_i = 0 \end{cases}$$

Problem: This is a non-convex optimization that is NP hard, i.e. impossible to solve !

Moreover, this doesn't distinguish between disastrous errors and near misses.

Instead we do: $\min \frac{1}{2}\|\underline{w}\|^2 + C\sum_{i=1}^{N} \xi_i$

Non-Separable SVM Dual Problem

Objective:

$$\min_{w,\xi_i} \frac{1}{2}\|\underline{w}\|^2 + C\sum_{i=1}^{N} \xi_i \quad \text{subject to} \quad y_i\left(\underline{w}^T \underline{x}_i + w_0\right) \ge 1 - \xi_i \qquad i = 1, \cdots, N$$

$$\text{and} \quad \xi_i \ge 0$$

As before, this is solved using the Lagrangian and the KKT conditions.

(For a complete derivation of the Lagrangian see e.g. "A Tutorial on Support Vector Machines for Pattern Recognition" by C.J.C Burges)

The dual problem turns out to be:

$$\hat{\lambda} = \arg\max_{\lambda} -\frac{1}{2}\sum_{i,j}^{N} \lambda_i \lambda_j y_i y_j \underline{x}_i^T \underline{x}_j + \sum_{i}^{N} \lambda_i \quad \text{subject to} \quad \sum_{i=1}^{N} \lambda_i y_i = 0$$

$$0 \le \lambda_i \boxed{\le C}$$

Who can spot the difference with the original dual problem? This is a huge difference !

Separable vs Non-Separable SVM

Primal problem:

$$\min_{w,\xi_i} \frac{1}{2}\|\underline{w}\|^2 + C\sum_{i=1}^{N}\xi_i \quad \text{subject to} \quad y_i\left(\underline{w}^T\underline{x}_i + w_0\right) \geq 1 - \xi_i \quad i = 1,\cdots,N$$

$$\text{and} \quad \xi_i \geq 0$$

Dual problem:

$$\hat{\lambda} = \arg\max_{\lambda} -\frac{1}{2}\sum_{i,j}^{N}\lambda_i\lambda_j y_i y_j \underline{x}_i^T\underline{x}_j + \sum_{i}^{N}\lambda_i \quad \text{subject to} \quad \sum_{i=1}^{N}\lambda_i y_i = 0$$

$$0 \leq \lambda_i \leq C$$

The separable case is a special case of this case. What should be done to get back to the separable case?

If $C = \infty$, we get back to the separable case.

Influence of the Parameter C

$$\min_{w,\xi_i} \underbrace{\frac{1}{2}\|\underline{w}\|^2}_{\text{improves generalization}} + C\underbrace{\sum_{i=1}^{N}\xi_i}_{\text{reduce \textit{training} errors}}$$

If C is high ... ?

fewer training errors,
lower generalization performance,
less support vectors.

If C is low ... ?

the opposite !

C is generally adjusted by trial/error on a validation set.

Non-Separable SVM

As before, once $\hat{\lambda}$ is found:

$$\hat{w} = \sum_{i=1}^{N} \hat{\lambda}_i y_i x_i \qquad \rightarrow \hat{w}$$

$$\lambda_i \left[y_i \left(\hat{w}^T x_i + \hat{w}_0 \right) - (1 - \xi_i) \right] = 0 \qquad \rightarrow \hat{w}_0$$

The support vectors are those for which $\hat{\lambda}_i \neq 0$!
But what are the values of ξ_i ?

From the KKT-conditions of the full Lagrangian for the non-separable SVM follows:

$$\forall i \ \text{ with } \ \hat{\lambda}_i < C \quad \rightarrow \quad \xi_i = 0$$

$$\rightarrow \quad \lambda_i \left[y_i \left(\hat{w}^T x_i + \hat{w}_0 \right) - 1 \right] = 0 \qquad \rightarrow \hat{w}_0$$

Applications

Linear classifiers are best applied to ...
... linear problems !

However, in practice, it is difficult to find linear problems. But even if the problem is not linearly separable, Sum of Square Classifier and Non-Separable Linear SVM may be applied.

Though, due to the simplicity of the classifier, we expect sub-optimal results.

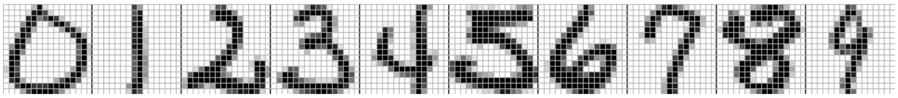
Zip Code Recognition

Example of application: Zip Code Recognition

A Standardized set of normalized digit data is available at:
http://www-stat-class.stanford.edu/~tibs/ElemStatLearn/

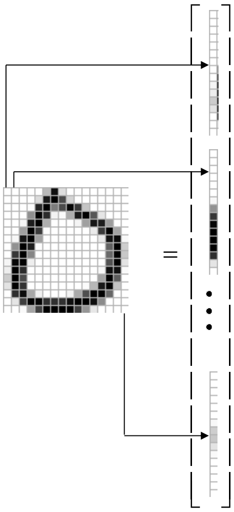
- 7291 digits used for training
- 2007 digits used for testing
- 1 digit = 16x16 grey level value

Example:



Zip Code Feature

Using as feature vector, the simplest of its features: the pixel intensities



$$= \begin{bmatrix}0\\0\\0\\15\\24\\43\\0\\\vdots\\12\\0\end{bmatrix} = x_i$$

$i = 1,\cdots, N = 7291$

$\underline{x}_i \in \mathbb{R}^{256}$

$w \in \mathbb{R}^{256}$

$w_0 \in \mathbb{R}$

Multiple Classes

In this example, there are 10 classes, but all the linear classifiers that we have reviewed can only discriminate between 2 classes.

So what can we do ?

We use the one against all strategy:
We build 10 classifiers:

$$g^{0}(x) \equiv x^{T} w^{0} + w_{0}^{0} \begin{cases} > 0, \text{ x is the digit 0} \\ < 0, \text{ x is any other digit} \end{cases}$$

$$\vdots$$

$$g^{9}(x) \equiv x^{T} w^{9} + w_{0}^{9} \begin{cases} > 0, \text{ x is the digit 9} \\ < 0, \text{ x is any other digit} \end{cases}$$

Zip Code Sum of Squares Classifiers

Example 1: Sum of Squares classifier (0 versus rest)

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,256} & 1 \end{bmatrix}$$

X is the 7291x257 data matrix.

$$y = \begin{bmatrix} +1 \\ -1 \\ \vdots \\ -1 \end{bmatrix}$$

e.g. $\underline{x}_1$ represents the digit 0

y is the 7291x1 column vector representing class belonging.

+1 for the digit 0
-1 for any digit in [1, 9]

$$\begin{bmatrix} \underline{w}^{0} \\ w_{0}^{0} \end{bmatrix} = \left(X^{T} X \right)^{-1} X^{T} y$$

Optimal sum of squares classifier

Zip Code Linear SVM Classifier

Example 2: Linear SVM Classifier

Training:

$$\hat{\lambda} = \arg\max_{\lambda} -\frac{1}{2}\sum_{i,j}^{N} \lambda_i \lambda_j y_i y_j \underline{x}_i^T \underline{x}_j + \sum_{i}^{N} \lambda_i \quad \text{subject to} \quad \sum_{i=1}^{N} \lambda_i y_i = 0$$

$$0 \le \lambda_i \le C$$

$$\underline{w}^0 = \sum_{i=1}^{N_S} \hat{\lambda}_i y_i \underline{x}_i \qquad \rightarrow \underline{w}^0$$

$$\lambda_i \left[y_i \left(\underline{x}_i^T \underline{w}^0 + w_0^0 \right) - 1 \right] = 0 \qquad \rightarrow w_0^0$$

Classifying:

$$\underline{x}^T \underline{w}^0 + w_0^0 \begin{cases} > 0, & \underline{x} \text{ is the digit 0} \\ < 0, & \underline{x} \text{ is any other digit} \end{cases}$$

Conclusion

Linear classifiers are:

- Efficient,
- Simple and easy to train and classify.

However, they do not attain the best performance when the features are not linearly separable. This is because the model is too simplistic: The number of degrees of freedom is just 1+dimensionality of the feature space.