Sapienza University of Rome

Master in Artificial Intelligence and Robotics
Master in Engineering in Computer Science

Machine Learning

A.Y. 2020/2021

Prof. L. Iocchi, F. Patrizi

Sapienza University of Rome, Italy - Machine Learning (2020/2021)

7. Linear models for classification

L. Iocchi, F. Patrizi

Overview

- Linearly separable data
- Linear models
- Least squares
- Perceptron
- Fisher's linear discriminant
- Support Vector Machines

References

C. Bishop. Pattern Recognition and Machine Learning. Sect. 4.1, 7.1

T. Mitchell. Machine Learning. Section 4.4

Linear Models for Classification

Learning a function $f : X \rightarrow Y$, with ...

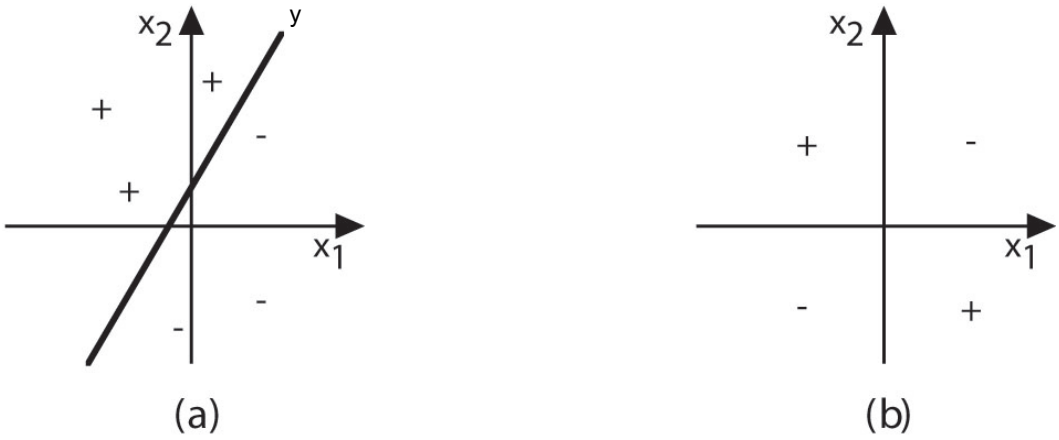
- $X \subseteq \Re^d$
- $Y = \{C_1, \ldots, C_k\}$

assuming *linearly separable* data.

Linearly separable data

data are linearly separable only when I can partition the classes with a line. If they are simply separable I could use also a curve, but in this case it would not be a linearly separable set

Instances in a data set are *linearly separable* iff there exists a hyperplane that separates the instance space into two regions, such that differently classified instances are separated



Linear discriminant functions

Linear discriminant function

$$y : X \to \{C_1, \ldots, C_K\}$$

Two classes:

weight vector, bias

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

The negative of the bias is sometimes called a threshold

K-class:

we need to consider k linear models

$$y_1(\mathbf{x}) = \mathbf{w}_1^T\mathbf{x} + w_{10}$$
$$\ldots$$
$$y_K(\mathbf{x}) = \mathbf{w}_K^T\mathbf{x} + w_{K0}$$

Compact notation

Two classes:

$$y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 = \tilde{\mathbf{w}}^T\tilde{\mathbf{x}}, \text{ with:}$$

$$\tilde{\mathbf{w}} = \begin{pmatrix} w_0 \\ \mathbf{w} \end{pmatrix}, \tilde{\mathbf{x}} = \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix}$$

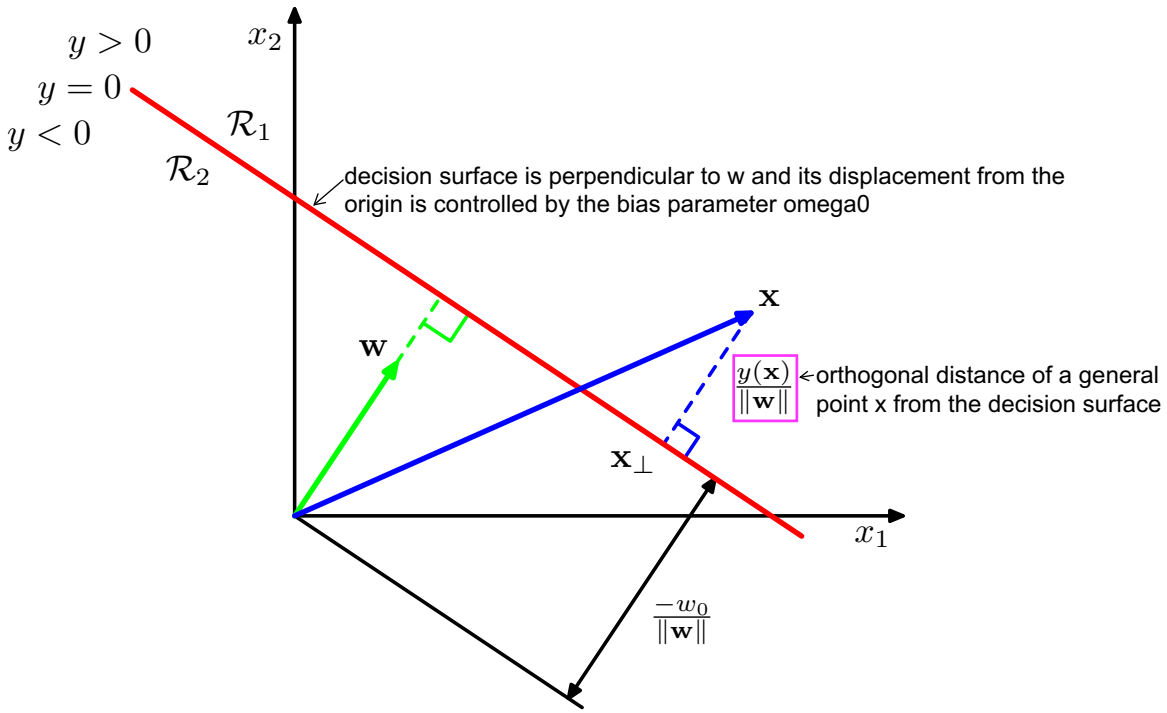
K classes:

$$\mathbf{y}(\mathbf{x}) = \begin{pmatrix} y_1(\mathbf{x}) \\ \cdots \\ y_K(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} \mathbf{w}_1^T\mathbf{x} + w_{10} \\ \cdots \\ \mathbf{w}_K^T\mathbf{x} + w_{K0} \end{pmatrix} = \begin{pmatrix} \tilde{\mathbf{w}}_1^T \\ \cdots \\ \tilde{\mathbf{w}}_K^T \end{pmatrix} \tilde{\mathbf{x}} = \tilde{\mathbf{W}}^T\tilde{\mathbf{x}}, \text{ with:}$$

$$\tilde{\mathbf{W}}^T = \begin{pmatrix} \tilde{\mathbf{w}}_1^T \\ \cdots \\ \tilde{\mathbf{w}}_K^T \end{pmatrix}, \text{ i.e.: } \tilde{\mathbf{W}} = (\tilde{\mathbf{w}}_1, \cdots, \tilde{\mathbf{w}}_K)$$

Linear discriminant functions

An input vector x is assigned to class C1 if y(x)>=0 and to class C2 otherwise. The corresponding decision boundary is therefore defined by the relation y(x)=0, which corresponds to a (D − 1)-dimensional hyperplane within the D-dimensional input space.

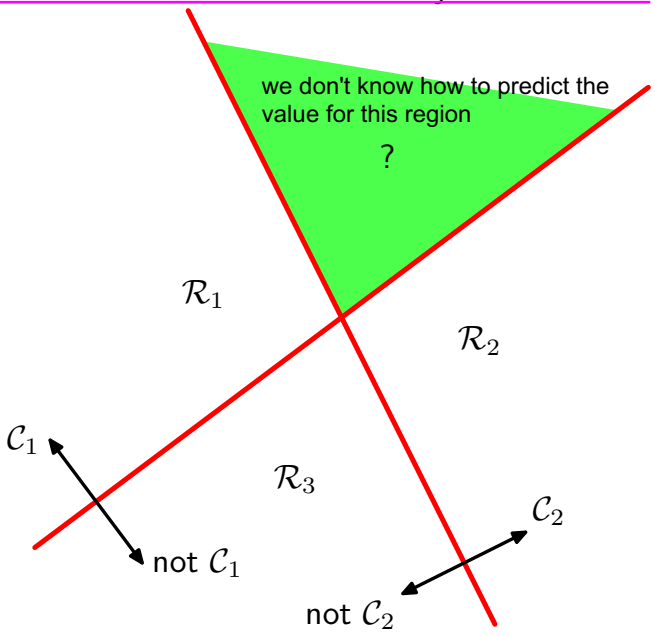


Multiple classes

Consider the use of K−1 classifiers each of which solves a two-class problem of separating points in a particular class Ck from points not in that class

Cannot use combinations of binary linear models.

One-versus-the-rest classifiers: $K-1$ binary classifiers: C_k vs. not-C_k

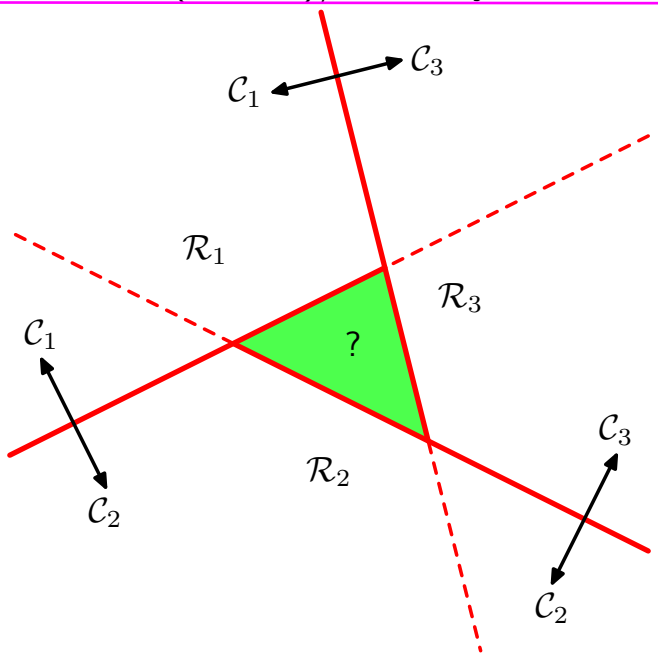


Multiple classes

Cannot use combinations of binary linear models.

Each point is then classified according to a majority vote amongst the discriminant functions.

One-versus-one classifiers: $K(K-1)/2$ binary classifiers: C_k vs. C_j



Multiple classes

K-class discriminant comprising K linear functions ($\mathbf{x}$ not in dataset)

$$\mathbf{y}(\mathbf{x}) = \begin{pmatrix} y_1(\mathbf{x}) \\ \cdots \\ y_K(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} \tilde{\mathbf{w}}_1^T \tilde{\mathbf{x}} \\ \cdots \\ \tilde{\mathbf{w}}_K^T \tilde{\mathbf{x}} \end{pmatrix} = \tilde{\mathbf{W}}^T \tilde{\mathbf{x}}$$

Classify $\mathbf{x}$ as C_k if $y_k(\mathbf{x}) > y_j(\mathbf{x})$ for all $j \neq k$ $(j, k = 1, \ldots, K)$

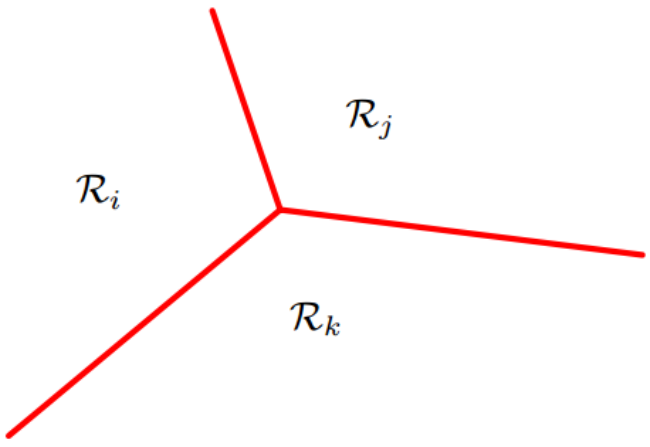
Decision boundary between C_k and C_j (hyperplane in $\Re^{D-1}$):

we need a set of decision boundaries one for each 2 classes

$$(\tilde{\mathbf{w}}_k - \tilde{\mathbf{w}}_j)^T \tilde{\mathbf{x}} = 0$$

Multiple classes

Example of K-class discriminant



Learning linear discriminants

Given a multi-class classification problem and data set D with linearly separable data,

determine $\tilde{\mathbf{W}}$ such that $\mathbf{y}(\mathbf{x}) = \tilde{\mathbf{W}}^T \tilde{\mathbf{x}}$ is the K-class discriminant.

Approaches to learn linear discriminants

- Least squares 4.1.3
- Perceptron
- Fisher's linear discriminant
- Support Vector Machines

Least squares

Consider a general classification problem with K classes , with a 1-of-K binary coding scheme for the target vector t. One justification for using least squares in such a context is that it approximates the conditional expectation E[t|x] of the target values Given the input vector.

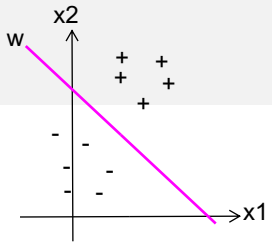
Given $D = \{(\mathbf{x}_n, \mathbf{t}_n)_{n=1}^N\}$, find the linear discriminant

$$\mathbf{y}(\mathbf{x}) = \tilde{\mathbf{W}}^T \tilde{\mathbf{x}}$$

1-of-K coding scheme for $\mathbf{t}$: $\mathbf{x} \in C_k \rightarrow t_k = 1, t_j = 0$ for all $j \neq k$.
E.g., $\mathbf{t}_n = (0, \ldots, 1, \ldots, 0)^T$

$$\tilde{\mathbf{X}} = \begin{pmatrix} \tilde{\mathbf{x}}_1^T \\ \cdots \\ \tilde{\mathbf{x}}_N^T \end{pmatrix} \qquad \mathbf{T} = \begin{pmatrix} \mathbf{t}_1^T \\ \cdots \\ \mathbf{t}_N^T \end{pmatrix}$$

Least squares



Minimize sum-of-squares error function

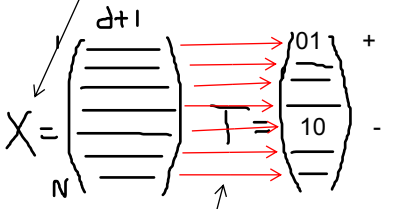
the trace is the sum of the components on the diagonal

$$E(\tilde{\mathbf{W}}) = \frac{1}{2} Tr\left\{(\tilde{\mathbf{X}}\tilde{\mathbf{W}} - \mathbf{T})^T (\tilde{\mathbf{X}}\tilde{\mathbf{W}} - \mathbf{T})\right\}$$

X = (all the points here, one on each row)

Closed-form solution:

$$\tilde{\mathbf{W}} = \underbrace{(\tilde{\mathbf{X}}^T\tilde{\mathbf{X}})^{-1}\tilde{\mathbf{X}}^T}_{\tilde{\mathbf{X}}^\dagger}\mathbf{T}$$



correspondence between the point and the label

discriminant function $\longrightarrow$ $\mathbf{y}(\mathbf{X}) = \tilde{\mathbf{W}}^T\tilde{\mathbf{X}} = \mathbf{T}^T(\tilde{\mathbf{X}}^\dagger)^T\tilde{\mathbf{X}}$

```
W = pinv(X) * T
```

Least squares

Classification of new instance $\mathbf{x}$ not in dataset:

Use learnt $\tilde{\mathbf{W}}$ to compute:

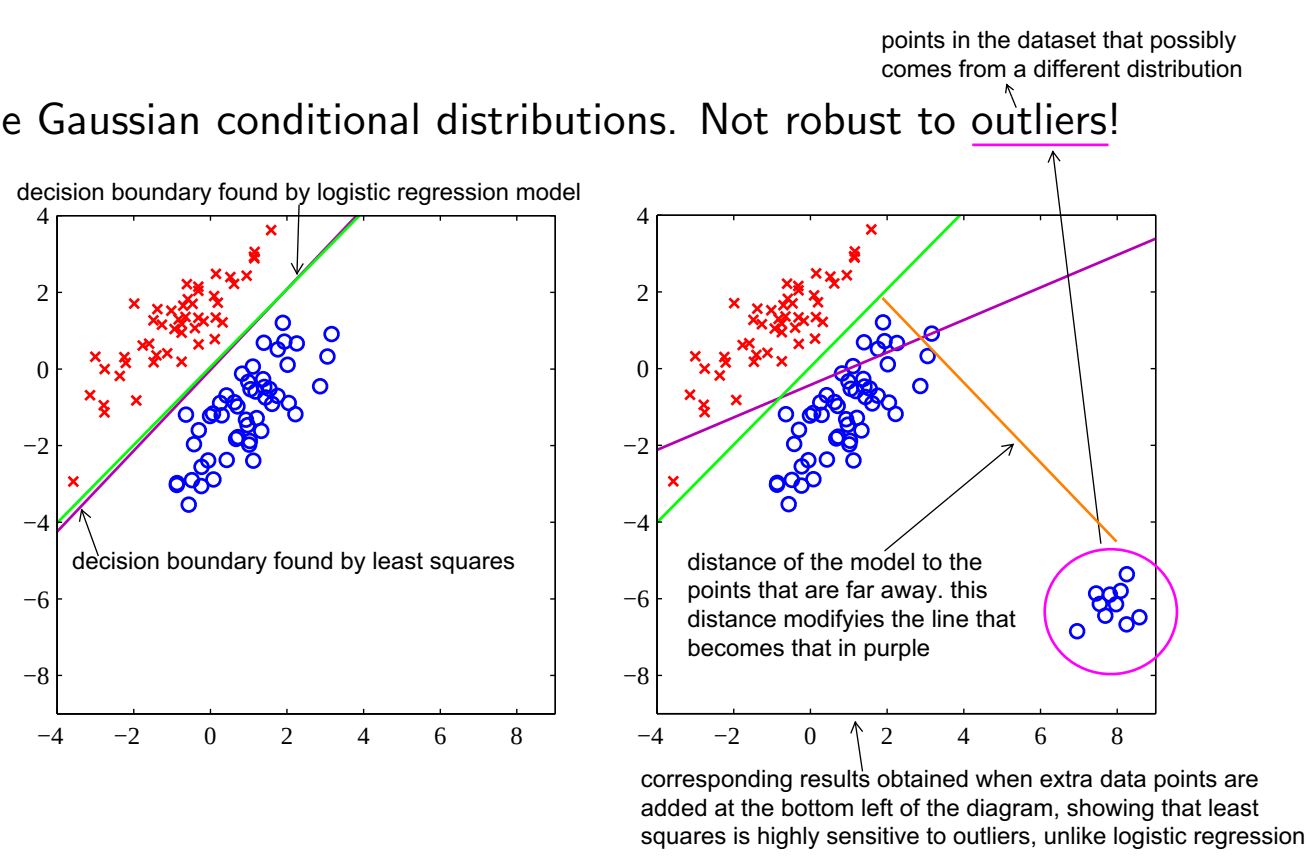
$$\mathbf{y}(\mathbf{x}) = \tilde{\mathbf{W}}^T \tilde{\mathbf{x}} = \begin{pmatrix} y_1(\mathbf{x}) \\ \cdots \\ y_K(\mathbf{x}) \end{pmatrix}$$

Assign class C_k to $\mathbf{x}$, where:

$$k = \underset{i \in \{1,\ldots,k\}}{\operatorname{argmax}} \{y_i(\mathbf{x})\}$$

Issues with least squares

Assume Gaussian conditional distributions. Not robust to outliers!

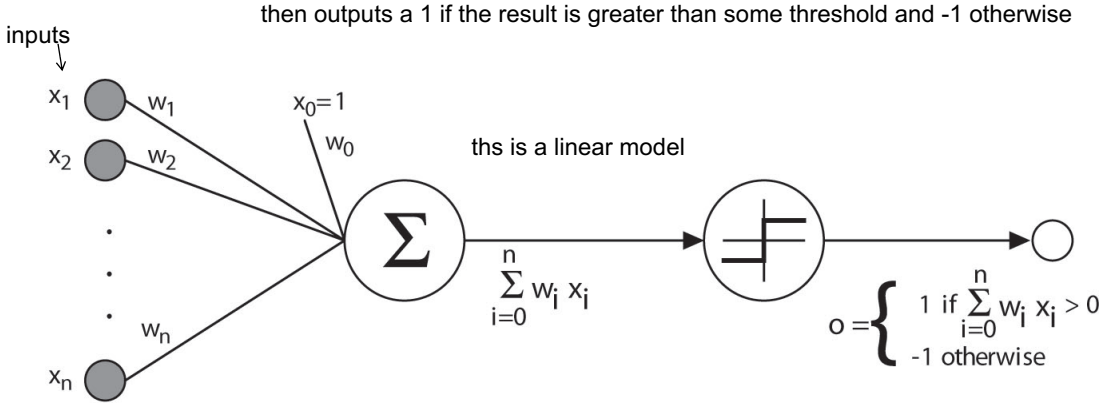


A single perceptron can be used to represent many boolean functions

Perceptron

is a model of the first simple block of a NN

A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise



-w0 is a threshold that the weighted combination of inputs w1*x1 +. . .+ wn*xn must surpass in order for the perceptron to output a 1.

$$o(x_1, \ldots, x_d) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_d x_d > 0 \\ -1 & \text{otherwise.} \end{cases}$$

$$o(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^T \mathbf{x} > 0 \\ -1 & \text{otherwise.} \end{cases} = sign(\mathbf{w}^T \mathbf{x})$$

sign function

The perceptron outputs a 1 for instances lying on one side of the hyperplane and outputs a -1 for instances lying on the other side, as illustrated in Figure 4.3. The equation for this decision hyperplane is w*x= 0

Learning a perceptron involves choosing values for the weights w0, .. .,wn. Therefore, the space H of candidate hypotheses considered in perceptron Learning is the set of all possible real-valued weight vectors.

Perceptron training rule

Although we are interested in learning networks of many interconnected units, let us begin by understanding how to learn the weights for a single perceptron. Here the precise learning problem is to determine a weight vector that causes the perceptron to produce the correct +-1 output for each of the given training examples. Several algorithms are known to solve this learning problem. Here we consider two: the perceptron rule and the delta rule. One way to learn an acceptable weight vector is to begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example. Weights are modified at each step according to the perceptron training rule

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta (t - o) x_i$$

learning rate

The role of the learning rate is to moderate the degree to which weights are changed at each step

Consider the *unthresholded linear unit*, where

$$o = w_0 + w_1 x_1 + \cdots + w_d x_d = \mathbf{w}^T \mathbf{x}$$

Let's learn w_i from training examples $D = \{(\mathbf{x}_n, t_n)_{n=1}^N\}$ that minimize the squared error (*loss function*)

$$E(\mathbf{w}) \equiv \frac{1}{2} \sum_{n=1}^{N} (t_n - o_n)^2 = \frac{1}{2} \sum_{n=1}^{N} (t_n - \mathbf{w}^T \mathbf{x}_n)^2$$

prediction that the model does on xn

E(w)=0 if tn=wTxn

Perceptron training rule

we do the derivative and we obtain what we see here

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{n=1}^{N}(t_n - \mathbf{w}_n^T\mathbf{x}_n)^2 = \frac{1}{2}\sum_{n=1}^{N}\frac{\partial}{\partial w_i}(t_n - \mathbf{w}^T\mathbf{x}_n)^2$$

$$= \frac{1}{2}\sum_{n=1}^{N} 2(t_n - \mathbf{w}^T\mathbf{x}_n)\frac{\partial}{\partial w_i}(t_n - \mathbf{w}^T\mathbf{x}_n)$$

$$= \sum_{n=1}^{N}(t_n - \mathbf{w}^T\mathbf{x}_n)\frac{\partial}{\partial w_i}(t_n - \mathbf{w}^T\mathbf{x}_n)$$

$$= \sum_{n=1}^{N}(t_n - \mathbf{w}^T\mathbf{x}_n)(-x_{i,n})$$

we can use this derivative (the gradient) to implement an iterative approach to find the minimum of a function

it comes from the developement of the derivative

xn is a sample that is a vector of dimension d, and the indices i indicates which component of xn we are considering

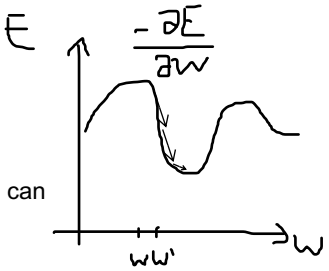
the negative derivative says us how much we can descend the curve of the function

Although the perceptron rule finds a successful weight vector when the training examples are linearly separable, it can fail to converge if the examples are not linearly separable. A second training rule, called the delta rule, is designed to overcome this difficulty.

Perceptron training rule

The delta training rule is best understood by considering the task of training an unthresholded perceptron

The key idea behind the delta rule is to use gradient descent to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples. This rule is important because gradient descent provides the basis for the BACKPROPAGATION algorithm, which can learn networks with many interconnected units.



Unthresholded unit:

the negative derivative says us how much we can descend the curve of the function

Update of weights **w**

the arrow means un update of the value

how much I move

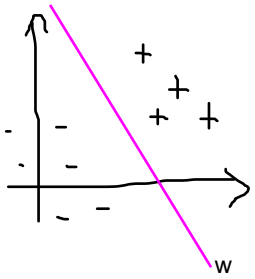
$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = -\eta\frac{\partial E}{\partial w_i} = \eta\sum_{n=1}^{N}(t_n - \mathbf{w}^T\mathbf{x}_n)\,x_{i,n}$$

η is a small constant (e.g., 0.05) called *learning rate*

Perceptron training rule

in this case if we apply the algo, no update is possible and the error will be zero



Thresholded unit:

Update of weights **w**

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = \eta \sum_{n=1}^{N} (t_n - sign(\mathbf{w}^T \mathbf{x}_n)) x_{i,n}$$

we simply add the sign function

they both can be +1 or -1

we use this function because it is not derivative. With the derivative we can block in a local minimum

Perceptron algorithm

Given perceptron model $o(\mathbf{x}) = sign(\mathbf{w}^T \mathbf{x})$ and data set D, determine weights **w**.

1. Initialize $\hat{\mathbf{w}}$ (e.g. small random values)
2. Repeat until termination condition
   - $\hat{w}_i \leftarrow \hat{w}_i + \Delta w_i$
3. Output $\hat{\mathbf{w}}$

Perceptron algorithm

consider all the examples is veri heavy

Batch mode: Consider all dataset D

$$\Delta w_i = \eta \sum_{(\mathbf{x},t)\in D} (t - o(\mathbf{x}))\, x_i$$

Mini-Batch mode: Choose a small subset $S \subset D$

this is the best mode, becuase is a middle via

$$\Delta w_i = \eta \sum_{(\mathbf{x},t)\in S} (t - o(\mathbf{x}))\, x_i$$

Incremental mode: Choose one sample $(\mathbf{x}, t) \in D$

$$\Delta w_i = \eta\, (t - o(\mathbf{x}))\, x_i$$

$o(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ for unthresholded, $o(\mathbf{x}) = sign(\mathbf{w}^T\mathbf{x})$ for thresholded
Incremental and mini-batch modes speed up convergence and are less sensitive to local minima.

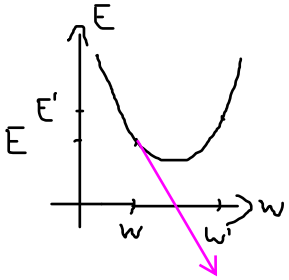
Perceptron algorithm

this method returns a line that separate so it converges when these two conditions are verified

Termination conditions

- Predefined number of iterations
- Threshold on changes in the loss function $E(\mathbf{w})$

Moreover the learning rate should be sufficiently small -> this condition is very important



if I choose a large learning rate, the resulting value of E' is bigger than E

Perceptron training rule

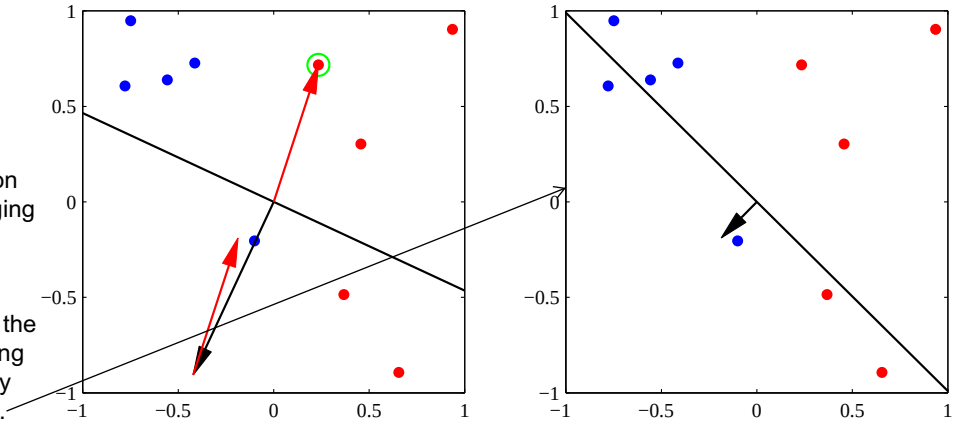
Example:
$\eta = 0.1$, $x_i = 0.8$

- if $t = 1$ and $o = -1$ then $\Delta w_i = 0.16$
- if $t = -1$ and $o = 1$ then $\Delta w_i = -0.16$

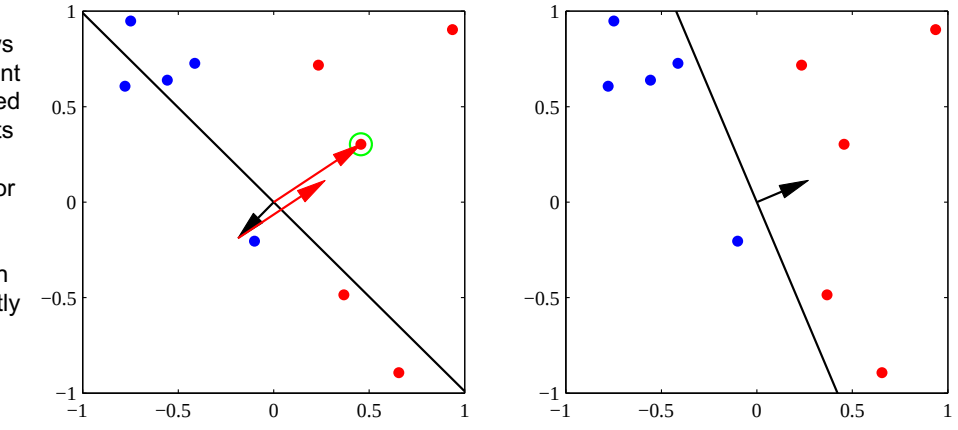
Perceptron training rule

Illustration of the convergence of the perceptron learning algorithm

initial parameter vector 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.



Perceptron training rule

Can prove it will converge:

- if training data is linearly separable
- and η sufficiently small

Small $\eta \rightarrow$ slow convergence.

Perceptron: Prediction

Classification of new instance $\mathbf{x}$ not in dataset:

Classify $\mathbf{x}$ as C_k, for $k = sign(\mathbf{w}^T\mathbf{x})$, using learnt $\mathbf{w}$

Fisher's linear discriminant

One way to view a linear classification model is in terms of dimensionality reduction. Consider first the case of two classes, and suppose we take the D- dimensional input vector x and project it down to one dimension using

Consider two classes case.

Determine $y = \mathbf{w}^T\mathbf{x}$
and classify $\mathbf{x} \in C_1$ if $y \geq -w_0$, $\mathbf{x} \in C_2$ otherwise.

Corresponding to the projection on a line determined by $\mathbf{w}$.

Fisher's linear discriminant

Adjusting $\mathbf{w}$ to find a direction that maximizes class separation.

Consider a data set with N_1 points in C_1 and N_2 points in C_2

mean vectors of the 2 classes

$$\mathbf{m}_1 = \frac{1}{N_1}\sum_{n \in C_1} \mathbf{x}_n \qquad \mathbf{m}_2 = \frac{1}{N_2}\sum_{n \in C_2} \mathbf{x}_n$$

The simplest measure of the separation of the classes, when projected onto w, is the separation of the projected class means. This suggests that we might choose w so as to maximize

Choose $\mathbf{w}$ that maximizes $J(\mathbf{w}) = \mathbf{w}^T(\mathbf{m}_2 - \mathbf{m}_1)$, subject to $||\mathbf{w}|| = 1$.

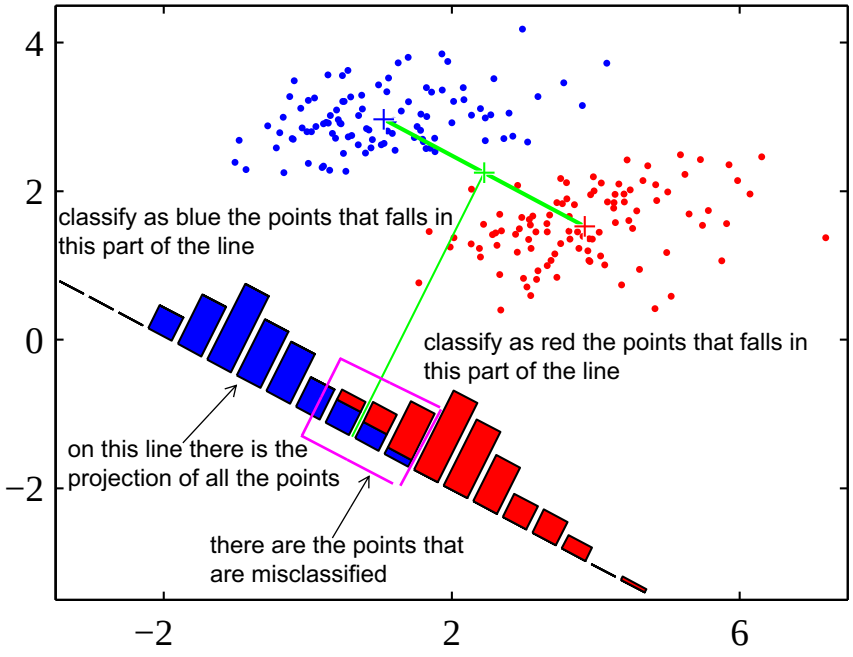
this expression can be made arbitrarily large simply by increasing the magnitude of w. To solve this problem, we could constrain w to have unit length

Fisher's linear discriminant

means of these distributions

Using a Lagrange multiplier to perform the constrained maximization, we then find this $\longrightarrow \mathbf{w} \propto (\mathbf{m}_2 - \mathbf{m}_1)$

The left plot shows samples from two classes (depicted in red and blue) along with the histograms resulting from projection onto the line joining the class means. Note that there is considerable class overlap in the projected space.



There is still a problem with this approach. The plot shows two classes that are well separated in the original two dimensional space (x1, x2) but that have considerable overlap when projected onto the line joining their means. This difficulty arises from the strongly non diagonal covariances of the class distributions. The idea proposed by Fisher is to maximize a function that will give a large separation between the projected class means while also giving a small variance within each class, so minimizing the class overlap. Result in SLIDE 36

Fisher's linear discriminant

Fisher criterion

$$J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}$$

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

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

Choose $\mathbf{w}$ that maximizes $J(\mathbf{w})$.

Fisher's linear discriminant

Find $\mathbf{w}$ that maximizes

$$J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}$$

by solving

$$\frac{d}{d\mathbf{w}} J(\mathbf{w}) = 0$$

Fisher's linear discriminant

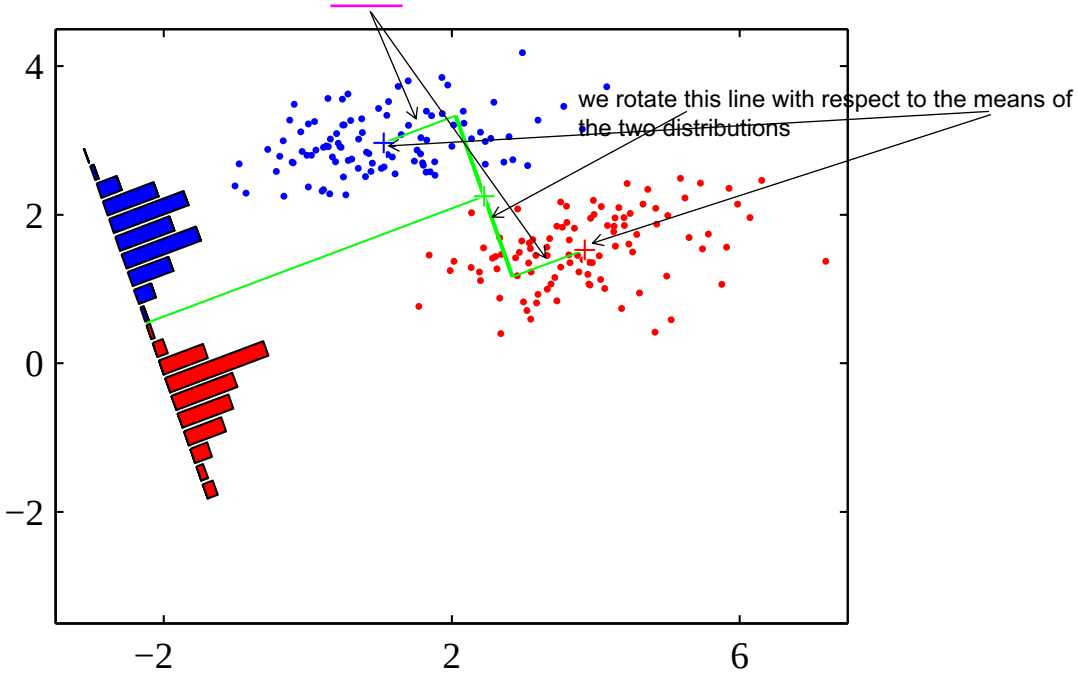
$$\Rightarrow \mathbf{w}^* \propto \mathbf{S}_W^{-1}(\mathbf{m}_2 - \mathbf{m}_1)$$

Fisher's linear discriminant

we add a moltiplication by a rotation matrix to solve the problem of misclassified points

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

the plot shows the corresponding projection based on the Fisher linear discriminant, showing the greatly improved class separation.



Fisher's linear discriminant

Summarizing, given a two classes classification problem, Fisher's linear discriminant is given by the function $y = \mathbf{w}^T\mathbf{x}$ and the classification of new instances is given by $y \geq -w_0$
where

$$\mathbf{w} = \mathbf{S}_W^{-1}(\mathbf{m}_2 - \mathbf{m}_1)$$

$$w_0 = \mathbf{w}^T\mathbf{m}$$

$\mathbf{m}$ is the global mean of all the data set.

Fisher's linear discriminant

Multiple classes.

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

Maximizing

$$J(\mathbf{W}) = Tr\left\{(\mathbf{W}\mathbf{S}_W\mathbf{W}^T)^{-1}(\mathbf{W}\mathbf{S}_B\mathbf{W}^T)\right\}$$

...

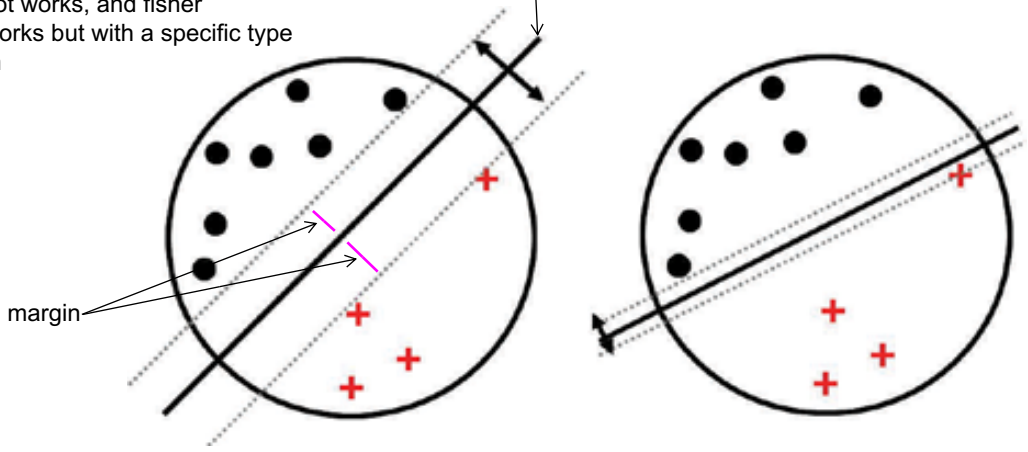
Support Vector Machines

the real goal of ML is that once we have computed the model (w) we have to seen what is the model that better classify samples that are not in the dataset

Support Vector Machines (SVM) for Classification aims at maximum margin providing for better accuracy.

it is also robust to outliers

perceptron not works, and fisher sometimes works but with a specific type of distribution



Support Vector Machines

Let's consider binary classification $y : X \to \{+1, -1\}$ with data set $D = \{(\mathbf{x}_n, t_n)_{n=1}^N\}$, $t_n \in \{+1, -1\}$ and a linear model

using linear models of the form

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

Assume D is linearly separable

so there exists an hyperplane such that

$$\exists\, \mathbf{w}, w_0 \; s.t. \quad \begin{array}{l} y(\mathbf{x}_n) > 0, \; if \; t_n = +1 \\ y(\mathbf{x}_n) < 0, \; if \; t_n = -1 \end{array}$$

$$t_n\, y(\mathbf{x}_n) > 0 \; \forall n = 1, \ldots N$$

We shall assume for the moment that the training data set is linearly separable in feature space, so that by definition there exists at least one choice of the parameters w and b such that a function of the form y(xn)>0 for points having tn = +1 and y(xn) < 0 for points having tn = −1, so that tny(xn)>0 for all training data points.

Support Vector Machines

we want to maximize the margins of w

If there are multiple solutions all of which classify the training data set exactly, then we should try to find the one that will give the smallest generalization error. The support vector machine approaches this problem through the concept of the margin, which is defined to be the smallest distance between the decision boundary and any of the samples

Let $\mathbf{x}_k$ be the closest point of the data set D to the hyperplane $\bar{h} : \bar{\mathbf{w}}^T \mathbf{x} + \bar{w}_0 = 0$

the *margin* (smallest distance between $\mathbf{x}_k$ and $\bar{h}$) is $\frac{|y(\mathbf{x}_k)|}{||\mathbf{w}||}$

In support vector machines the decision boundary is chosen to be the one for which the margin is maximized.

Given data set D and hyperplane $\bar{h}$, the margin is computed as

$$\min_{n=1,\dots,N} \frac{|y(\mathbf{x}_n)|}{||\mathbf{w}||} = \dots = \frac{1}{||\mathbf{w}||} \min_{n=1,\dots,N} [t_n(\bar{\mathbf{w}}^T \mathbf{x}_n + \bar{w}_0)]$$

using the property $|y(\mathbf{x}_n)| = t_n\, y(\mathbf{x}_n)$

Support Vector Machines

instead of using this optimal boundary, they determine the best hyperplane by minimizing the probability of error relative to the learned density model

Given data set D, the hyperplane $h^* : \mathbf{w}^{*T} \mathbf{x} + w_0^* = 0$ with maximum margin is computed as

The margin is given by the perpendicular distance to the closest point xn from the data set, and we wish to optimize the parameters w and b in order to maximize this distance. Thus the maximum margin solution is found by solving

$$\mathbf{w}^*, w_0^* = \underset{\mathbf{w}, w_0}{\operatorname{argmax}} \frac{1}{||\mathbf{w}||} \min_{n=1,\dots,N} [t_n(\mathbf{w}^T \mathbf{x}_n + w_0)]$$

we have taken this factor outside the optimization over n because w does not depend on n

Direct solution of this optimization problem would be very complex, and so we shall convert it into an equivalent problem that is much easier to solve.

Support Vector Machines

Rescaling all the points does not affect the solution.

Rescale in such a way that for the closet point $\mathbf{x}_k$ we have

$$t_k(\mathbf{w}^T\mathbf{x}_k + w_0) = 1$$

Canonical representation: of the decision hyperplane

$$t_n(\mathbf{w}^T\mathbf{x}_n + w_0) \geq 1 \quad \forall n = 1, \ldots, N$$

Support Vector Machines

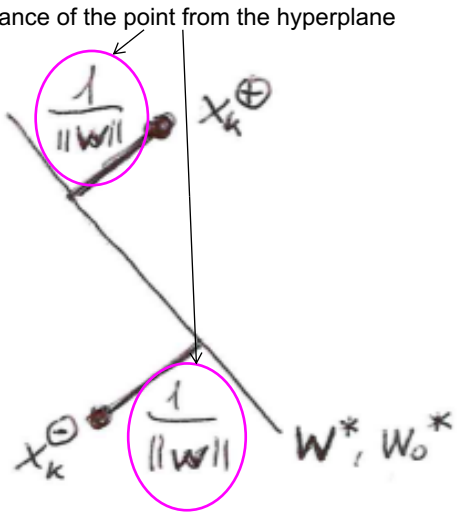
when we find the optimal hyperplane there exists at least 2 points, one for each class, that are closest

When the maximum margin hyperplane $\mathbf{w}^*, w_0^*$ is found, there will be at least 2 closest points $\mathbf{x}_k^{\oplus}$ and $\mathbf{x}_k^{\ominus}$ (one for each class).

for the 2 points these equalities are true

$$\mathbf{w}^{*T}\mathbf{x}_k^{\oplus} + w_0^* = +1$$
$$\mathbf{w}^{*T}\mathbf{x}_k^{\ominus} + w_0^* = -1$$

distance of the point from the hyperplane



Support Vector Machines

In the canonical representation of the problem the maximum margin hyperplane can be found by solving the optimization problem

$$\mathbf{w}^*, w_0^* = \operatorname{argmax} \frac{1}{||\mathbf{w}||} = \operatorname{argmin} \frac{1}{2}||\mathbf{w}||^2$$

from this we can go to the lagrangian optimizer

subject to

$$t_n(\mathbf{w}^T\mathbf{x}_n + w_0) \geq 1 \quad \forall n = 1, \ldots, N$$

This is an example of a quadratic programming problem in which we are trying to minimize a quadratic function subject to a set of linear inequality constraints.

Quadratic programming problem solved with Lagrangian method.

Support Vector Machines

Solution

$$\mathbf{w}^* = \sum_{n=1}^{N} a_n^* \, t_n \, \mathbf{x}_n$$

optimal solution of the optimization problem

a_i^* (Lagrange multipliers): results of the Lagrangian optimization problem

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

subject to

$$a_n \geq 0 \quad \forall n = 1, \ldots, N$$

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

Support Vector Machines

Note:
samples $\mathbf{x}_n$ for which $a_n^* = 0$ do not contribute to the solution

a constrained optimization of this form satisfies the Karush-Kuhn-Tucker (KKT) conditions

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

Karush-Kuhn-Tucker (KKT) condition:

for each $\mathbf{x}_n \in D$, *either* $a_n^* = 0$ *or* $t_n\, y(\mathbf{x}_n) = 1$

thus $t_n\, y(\mathbf{x}_n) > 1$ implies $a_n^* = 0$

The remaining data points are called support vectors, and because they satisfy tny(xn) = 1, they correspond to points that lie on the maximum margin hyperplanes in feature space

Support vectors: x_k such that $t_k\, y(\mathbf{x}_k) = 1$ and $a_k^* > 0$

samples that are closest to the hyperplane

$$SV \equiv \{\mathbf{x}_k \in D \mid t_k\, y(\mathbf{x}_k) = 1\}$$

Once the model is trained, a significant proportion of the data points can be discarded and only the support vectors retained.

Support Vector Machines

Hyperplanes expressed with support vectors

$$y(\mathbf{x}) = \sum_{\mathbf{x}_j \in SV} a_j^*\, t_j\, \mathbf{x}^T \mathbf{x}_j + w_0^* = 0$$

Note: other vectors $\mathbf{x}_n \notin SV$ do not contribute ($a_n^* = 0$)

Support Vector Machines

To compute w_0^*:

Support vector $\mathbf{x}_k \in SV$ satisfies $t_k\, y(\mathbf{x}_k) = 1$

$$t_k \left(\sum_{\mathbf{x}_j \in SV} a_j^* t_j \mathbf{x}_k^T \mathbf{x}_j + w_0^* \right) = 1$$

Multiplying by t_k and using $t_k^2 = 1$

$$w_0^* = t_k - \sum_{\mathbf{x}_j \in SV} a_j^* t_j \mathbf{x}_k^T \mathbf{x}_j$$

Support Vector Machines

Instead of using one particular support vector $\mathbf{x}_k$ to determine w_0

$$w_0^* = t_k - \sum_{\mathbf{x}_j \in SV} a_j^* t_j \mathbf{x}_k^T \mathbf{x}_j$$

a more stable solution is obtained by averaging over all the support vectors

$$w_0^* = \frac{1}{|SV|} \sum_{\mathbf{x}_k \in SV} \left(t_k - \sum_{\mathbf{x}_j \in S} a_j^* t_j \mathbf{x}_k^T \mathbf{x}_j \right)$$

Support Vector Machines

Given the maximum margin hyperplane determined by a_k^*, w_0^*

once we have the maximum margin we classify a new instance x'

Classification of a new instance $\mathbf{x}'$

$$sign(y(\mathbf{x}')) = sign\left(\sum_{\mathbf{x}_k \in SV} a_k^* \, t_k \, \mathbf{x}'^T \mathbf{x}_k + w_0^*\right)$$

Support Vector Machines

CONCLUSIONS

Optimization problem for determining **w** (dimension $|X|$) transformed in an optimization problem for determining **a** (dimension $|D|$)

Efficient when $|X| << |D|$ (most of a_i will be zero).
Very useful when $|X|$ is large or infinite.

7.1.1

Support Vector Machines with soft margin constraints

We now modify this approach so that data points are allowed to be on the 'wrong side' of the margin boundary, but with a penalty that increases with the distance from that boundary. For the subsequent optimization problem, it is convenient to make this penalty a linear function of this distance. To do this, we introduce slack variables

so we have some noise

What if data are "almost" linearly separable (e.g., a few points are on the "wrong side")

used to attenuate the constraint

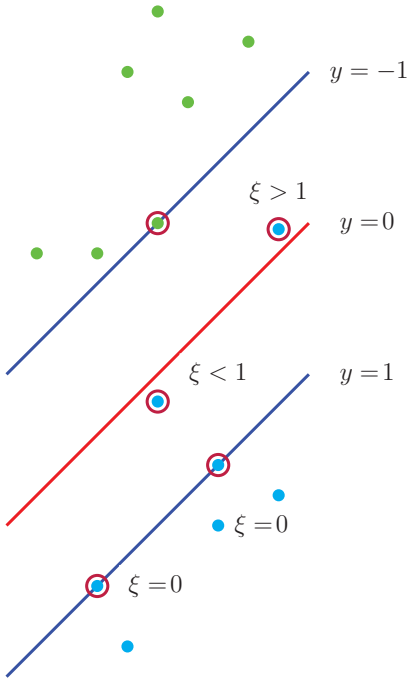
Let us introduce *slack variables* $\xi_n \geq 0$ $n = 1, \ldots, N$

one slack variable for each training data point

when the data are not linearly separable we can find a curve that can separate the data

Support Vector Machines with soft margin constraints

- $\xi_n = 0$ if point on or inside the correct margin boundary
- $0 < \xi_n \leq 1$ if point inside the margin but correct side
- $\xi_n > 1$ if point on wrong side of boundary



when $\xi_n = 1$, the sample lies on the decision boundary $y(\mathbf{x}_n) = 0$
when $\xi_n > 1$, the sample will be mis-classified

Support Vector Machines with soft margin constraints

N.B: Note that while slack variables allow for overlapping class distributions, this framework is still sensitive to outliers because the penalty for misclassification increases linearly with ξ.

Soft margin constraint

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

if chi_n = 0 the constraint is the same as before

Optimization problem with soft margin constraints

C controls the trade-off between the slack variable penalty and the margin

it is the argmin of everything

$$\mathbf{w}^*, w_0^* = \text{argmin}\ \frac{1}{2}||\mathbf{w}||^2 + C\sum_{n=1}^{N} \xi_n$$

sum of all the slack variables

subject to

if we add the constraint 0<chi_n<=1, the method doesn't work with the almost linearly separable dataset

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

$$\xi_n \geq 0\,, \quad n = 1, \ldots, N$$

C is a constant (inverse of a regularization coefficient)

Support Vector Machines with soft margin constraints

Solution similar to the case of linearly separable data.

$$\mathbf{w}^* = \sum_{n=1}^{N} a_n^*\, t_n\, \mathbf{x}_n$$

this is still a quadratic programming problem that can be solved with the lagrangian optimization

$$w_0^* = \ldots.$$

with a_n^* computed as solution of a Lagrangian optimization problem.

Basis functions

when a datset is not linearly separable like in SLIDE 58 we can make it linearly separable

So far we considered models working directly on $\mathbf{x}$.

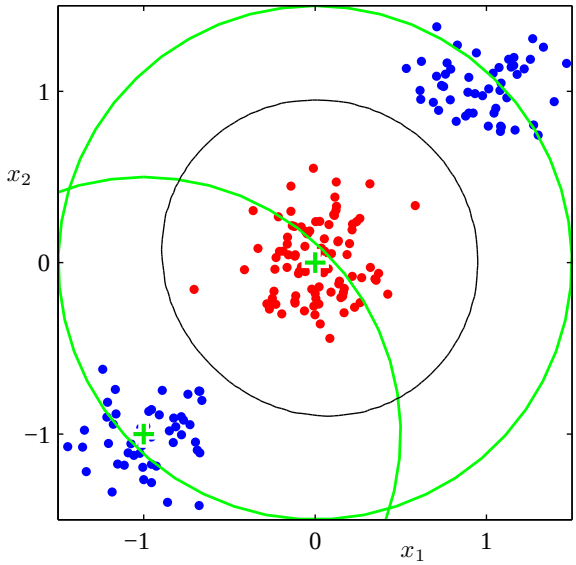
All the results hold if we consider a non-linear transformation of the inputs $\phi(\mathbf{x})$ (*basis functions*).

Decision boundaries will be linear in the feature space ϕ and non-linear in the original space $\mathbf{x}$

Classes that are linearly separable in the feature space ϕ may not be separable in the input space $\mathbf{x}$.

Basis functions example

non linearly separable dataset



we made the dataset linearly separable

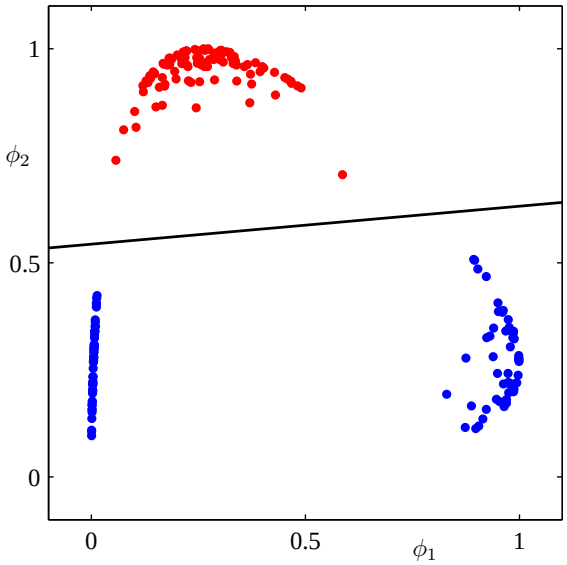


Illustration of the role of nonlinear basis functions in linear classification models. it shows the original input space (x1, x2) together with data points from two classes labelled red and blue. Two 'Gaussian' basis functions ϕ1(x) and ϕ2(x) are defined in this space with centres shown by the green crosses and with contours shown by the green circles.

it shows the corresponding feature space (ϕ1, ϕ2) together with the linear decision boundary obtained given by a logistic regression model. This corresponds to a nonlinear decision boundary in the original input space, shown by the black curve in the left-hand plot.

Basis functions examples

- Linear
- Polynomial
- Radial Basis Function (RBF)
- Sigmoid
- ...

Linear models for non-linear functions

this method can be expanded to multiple classes

Learning non-linear function

$$y : X \rightarrow \{C_1, \ldots, C_K\}$$

from data set D non-linearly separable.

Find a non-linear transformation ϕ and learn a linear model

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + w_0 \ \textit{(two classes)}$$

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \phi(\mathbf{x}) + w_{k0} \ \textit{(multiple classes)}$$

Summary

- Basic methods for learning linear classification functions
- Based on solution of an optimization problem
- Closed form vs. iterative solutions
- Sensitivity to outliers
- Learning non-linear functions with linear models using basis functions
- Further developed as kernel methods