

# Soft-margin SVM, SMO Algorithm, Decision Trees

Milan Straka

 November 25, 2019



Charles University in Prague  
Faculty of Mathematics and Physics  
Institute of Formal and Applied Linguistics



unless otherwise stated

# Kernel Linear Regression

When dimensionality of input is  $D$ , one step of SGD takes  $\mathcal{O}(D^3)$ .

Surprisingly, we can do better under some circumstances. We start by noting that we can write the parameters  $\mathbf{w}$  as a linear combination of the input features  $\varphi(\mathbf{x}_i)$ .

By induction,  $\mathbf{w} = \mathbf{0} = \sum_i 0 \cdot \varphi(\mathbf{x}_i)$ , and assuming  $\mathbf{w} = \sum_i \beta_i \cdot \varphi(\mathbf{x}_i)$ , after a SGD update we get

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} + \alpha \sum_i (t_i - \mathbf{w}^T \varphi(\mathbf{x}_i)) \varphi(\mathbf{x}_i) \\ &= \sum_i \left( \beta_i + \alpha (t_i - \mathbf{w}^T \varphi(\mathbf{x}_i)) \right) \varphi(\mathbf{x}_i). \end{aligned}$$

A individual update is  $\beta_i \leftarrow \beta_i + \alpha (t_i - \mathbf{w}^T \varphi(\mathbf{x}_i))$ , and substituting for  $\mathbf{w}$  we get

$$\beta_i \leftarrow \beta_i + \alpha \left( t_i - \sum_j \beta_j \varphi(\mathbf{x}_j)^T \varphi(\mathbf{x}_i) \right).$$

We can formulate the alternative linear regression algorithm (it would be called a *dual formulation*):

**Input:** Dataset ( $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\} \in \mathbb{R}^{N \times D}$ ,  $\mathbf{t} \in \mathbb{R}^N$ ), learning rate  $\alpha \in \mathbb{R}^+$ .

- Set  $\beta_i \leftarrow 0$
- Compute all values  $K(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_j)$
- Repeat
  - Update the coordinates, either according to a full gradient update:
    - $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \alpha(\mathbf{t} - \mathbf{K}\boldsymbol{\beta})$
  - or alternatively use single-batch SGD, arriving at:
    - for  $i$  in random permutation of  $\{1, \dots, N\}$ :
      - $\beta_i \leftarrow \beta_i + \alpha \left( t_i - \sum_j \beta_j K(\mathbf{x}_i, \mathbf{x}_j) \right)$

In vector notation, we can write  $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \alpha(\mathbf{t} - \mathbf{K}\boldsymbol{\beta})$ .

The predictions are then performed by computing  $y(\mathbf{x}) = \mathbf{w}^T \varphi(\mathbf{x}) = \sum_i \beta_i \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x})$ .

We define a *kernel* corresponding to a feature map  $\varphi$  as a function

$$K(\mathbf{x}, \mathbf{z}) \stackrel{\text{def}}{=} \varphi(\mathbf{x})^t \varphi(\mathbf{z}).$$

There is quite a lot of theory behind kernel construction. The most often used kernels are:

- polynomial kernel or degree  $d$

$$K(\mathbf{x}, \mathbf{z}) = (\gamma \mathbf{x}^T \mathbf{z} + 1)^d,$$

which corresponds to a feature map generating all combinations of up to  $d$  input features;

- Gaussian (or RBF) kernel

$$K(\mathbf{x}, \mathbf{z}) = e^{-\gamma \|\mathbf{x} - \mathbf{z}\|^2},$$

corresponding to a scalar product in an infinite-dimensional space (it is in a sense a combination of polynomial kernels of all degrees).

# Support Vector Machines

Assume we have a dataset  $\mathbf{X} \in \mathbb{R}^{N \times D}$ ,  $\mathbf{t} \in \{-1, 1\}^N$ , feature map  $\varphi$  and model

$$y(\mathbf{x}) \stackrel{\text{def}}{=} \varphi(\mathbf{x})^T \mathbf{w} + b.$$

We already know that the distance of a point  $\mathbf{x}_i$  to the decision boundary is

$$\frac{|y(\mathbf{x}_i)|}{\|\mathbf{w}\|} = \frac{t_i y(\mathbf{x}_i)}{\|\mathbf{w}\|}.$$

We therefore want to maximize

$$\arg \max_{\mathbf{w}, b} \frac{1}{\|\mathbf{w}\|} \min_i [t_i (\varphi(\mathbf{x})^T \mathbf{w} + b)].$$

However, this problem is difficult to optimize directly.

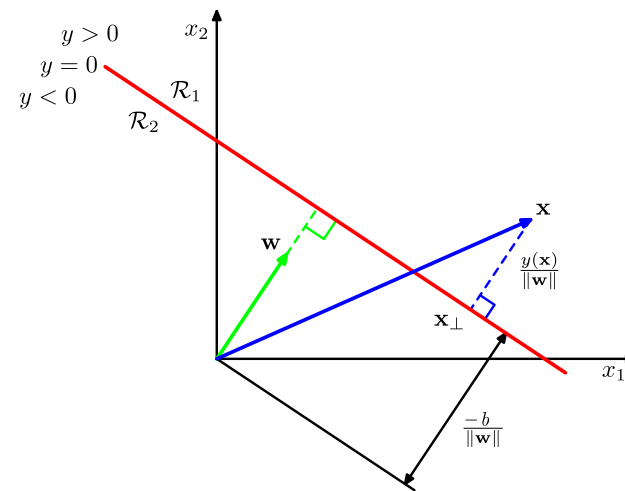


Figure 4.1 of Pattern Recognition and Machine Learning.

Because the model is invariant to multiplying  $\mathbf{w}$  and  $b$  by a constant, we can say that for the points closest to the decision boundary, it will hold that

$$t_i y(\mathbf{x}_i) = 1.$$

Then for all the points we will have  $t_i y(\mathbf{x}_i) \geq 1$  and we can simplify

$$\arg \max_{w,b} \frac{1}{\|\mathbf{w}\|} \min_i [t_i (\boldsymbol{\varphi}(\mathbf{x})^T \mathbf{w} + b)]$$

to

$$\arg \min_{w,b} \frac{1}{2} \|\mathbf{w}\|^2 \text{ given that } t_i y(\mathbf{x}_i) \geq 1.$$

In order to solve the constrained problem of

$$\arg \min_{w,b} \frac{1}{2} \|\mathbf{w}\|^2 \text{ given that } t_i y(\mathbf{x}_i) \geq 1,$$

we write the Lagrangian with multipliers  $\mathbf{a} = (a_1, \dots, a_N)$  as

$$L = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_i a_i [t_i y(\mathbf{x}_i) - 1].$$

Setting the derivatives with respect to  $\mathbf{w}$  and  $b$  to zero, we get

$$\begin{aligned} \mathbf{w} &= \sum_i a_i t_i \varphi(\mathbf{x}_i) \\ 0 &= \sum_i a_i t_i \end{aligned}$$

Substituting these to the Lagrangian, we get

$$L = \sum_i a_i - \frac{1}{2} \sum_i \sum_j a_i a_j t_i t_j K(\mathbf{x}_i, \mathbf{x}_j)$$

with respect to the constraints  $\forall_i : a_i \geq 0$ ,  $\sum_i a_i t_i = 0$  and kernel  $K(\mathbf{x}, \mathbf{z}) = \varphi(\mathbf{x})^T \varphi(\mathbf{z})$ .

The solution of this Lagrangian will fulfil the KKT conditions, meaning that

$$\begin{aligned} a_i &\geq 0 \\ t_i y(\mathbf{x}_i) - 1 &\geq 0 \\ a_i (t_i y(\mathbf{x}_i) - 1) &= 0. \end{aligned}$$

Therefore, either a point is on a boundary, or  $a_i = 0$ . Given that the predictions for point  $\mathbf{x}$  are given by  $y(\mathbf{x}) = \sum a_i t_i K(\mathbf{x}, \mathbf{x}_i) + b$ , we need to keep only the points on the boundary, the so-called **support vectors**.



The dual formulation allows us to use non-linear kernels.

**Figure 7.2** Example of synthetic data from two classes in two dimensions showing contours of constant  $y(\mathbf{x})$  obtained from a support vector machine having a Gaussian kernel function. Also shown are the decision boundary, the margin boundaries, and the support vectors.

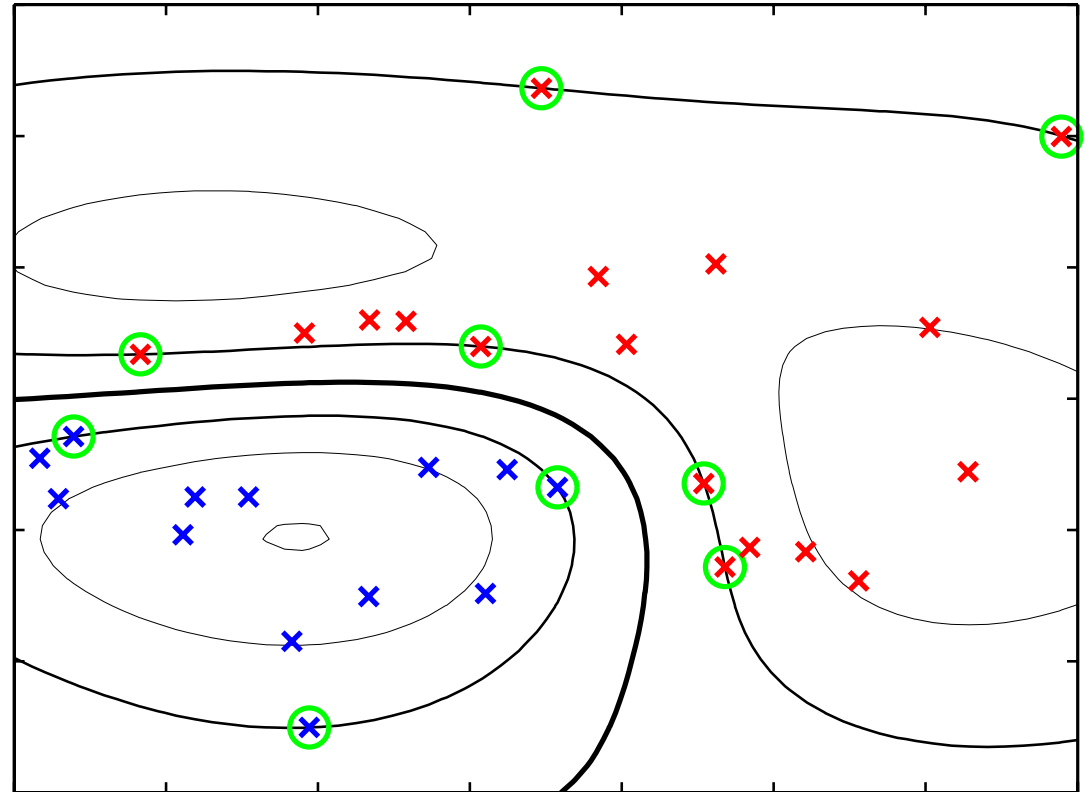


Figure 7.2 of Pattern Recognition and Machine Learning.

# Support Vector Machines for Non-linearly Separable Data

Until now, we assumed the data to be linearly separable – the **hard-margin SVM** variant. We now relax this condition to arrive at **soft-margin SVM**. The idea is to allow points to be in the margin or even on the *wrong side* of the decision boundary. We introduce *slack variables*  $\xi_i \geq 0$ , one for each training instance, defined as

$$\xi_i = \begin{cases} 0 & \text{for points fulfilling } t_i y(\mathbf{x}_i) \geq 1, \\ |t_i - y(\mathbf{x}_i)| & \text{otherwise.} \end{cases}$$

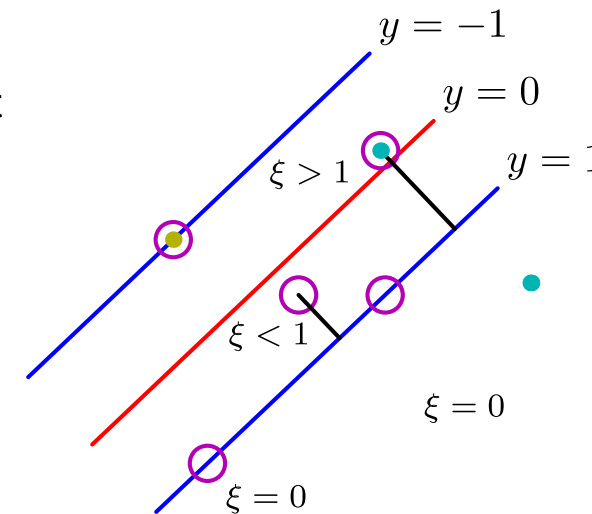


Figure 7.3 of Pattern Recognition and Machine Learning.

Therefore,  $\xi_i = 0$  signifies a point outside of margin,  $0 < \xi_i < 1$  denotes a point inside the margin,  $\xi_i = 1$  is a point on the decision boundary and  $\xi_i > 1$  indicates the point is on the opposite side of the separating hyperplane.

Therefore, we want to optimize

$$\arg \min_{w,b} C \sum_i \xi_i + \frac{1}{2} \|\mathbf{w}\|^2 \text{ given that } t_i y(\mathbf{x}_i) \geq 1 - \xi_i \text{ and } \xi_i \geq 0.$$

# Support Vector Machines for Non-linearly Separable Data

We again create a Lagrangian, this time with multipliers  $\mathbf{a} = (a_1, \dots, a_N)$  and also  $\boldsymbol{\mu} = (\mu_1, \dots, \mu_N)$ :

$$L = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \xi_i - \sum_i a_i [t_i y(\mathbf{x}_i) - 1 + \xi_i] - \sum_i \mu_i \xi_i.$$

Solving for the critical points and substituting for  $\mathbf{w}$ ,  $b$  and  $\boldsymbol{\xi}$  (obtaining an additional constraint  $\mu_i = C - a_i$  compared to the previous case), we obtain the Lagrangian in the form

$$L = \sum_i a_i - \frac{1}{2} \sum_i \sum_j a_i a_j t_i t_j K(\mathbf{x}_i, \mathbf{x}_j),$$

which is identical to the previous case, but the constraints are a bit different:

$$\forall_i : C \geq a_i \geq 0 \text{ and } \sum_i a_i t_i = 0.$$

# Support Vector Machines for Non-linearly Separable Data

Using KKT conditions, we can see that the support vectors (examples with  $a_i > 0$ ) are the ones with  $t_i y(\mathbf{x}_i) = 1 - \xi_i$ , i.e., the examples on the margin boundary, inside the margin and on the opposite side of the decision boundary.

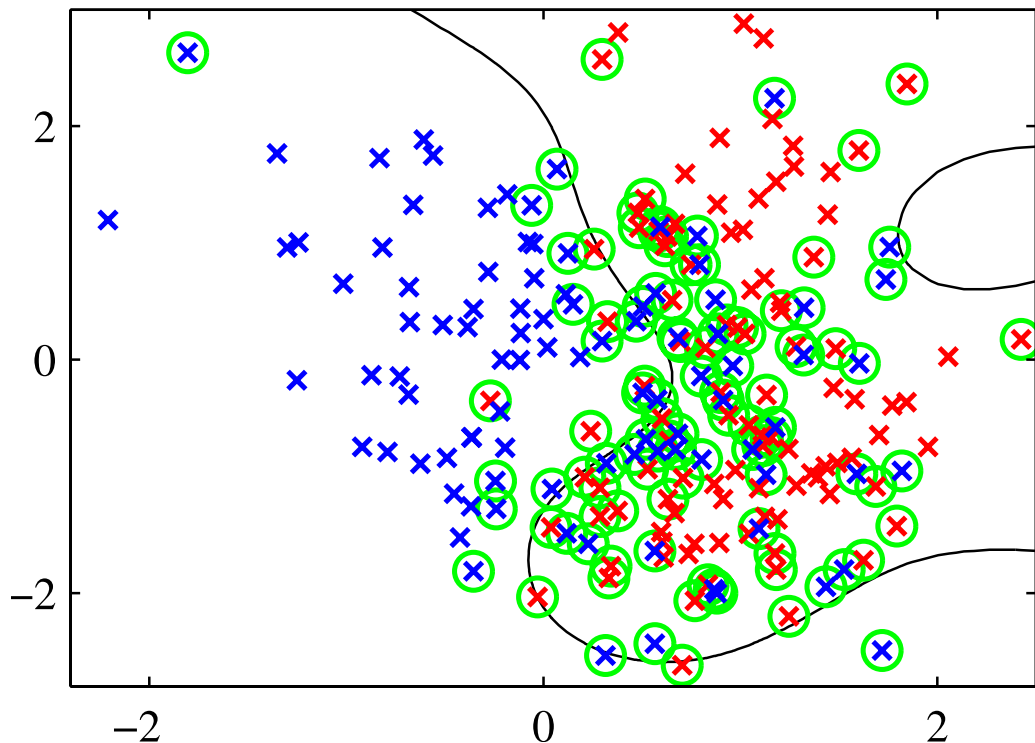


Figure 7.4 of *Pattern Recognition and Machine Learning*.

Note that the slack variables can be written as

$$\xi_i = \max(0, 1 - t_i y(\mathbf{x}_i)),$$

so we can reformulate the soft-margin SVM objective using **hinge loss**

$$\mathcal{L}_{\text{hinge}}(t, y) \stackrel{\text{def}}{=} \max(0, 1 - ty)$$

to

$$\arg \min_{w, b} C \sum_i \mathcal{L}_{\text{hinge}}(t_i, y(\mathbf{x}_i)) + \frac{1}{2} \|\mathbf{w}\|^2.$$

Such formulation is analogous to a regularized loss, where  $C$  is an *inverse* regularization strength, so  $C = \infty$  implies no regularization and  $C = 0$  ignores the data entirely.

# Comparison of Linear and Logistic Regression and SVM

For  $f(\mathbf{x}; \mathbf{w}, b) \stackrel{\text{def}}{=} \boldsymbol{\varphi}(\mathbf{x})^T \mathbf{w} + b$ , we have seen the following losses:

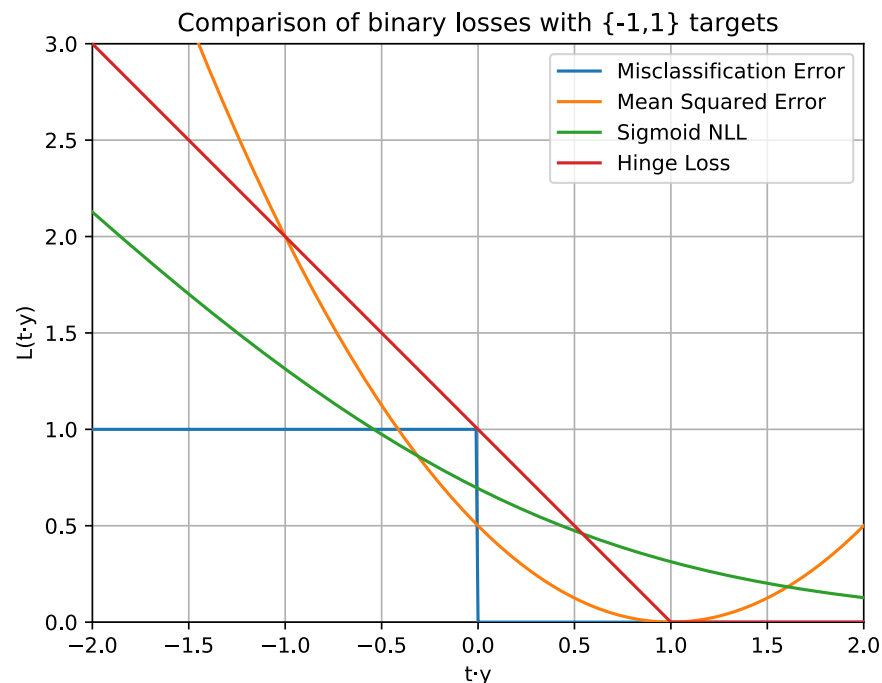
Model	Objective	Per-Instance Loss
Linear Regression	$\arg \min_{\mathbf{w}, b} \sum_i \mathcal{L}_{\text{MSE}}(t_i, f(\mathbf{x}_i)) + \frac{1}{2} \lambda \ \mathbf{w}\ ^2$	$\mathcal{L}_{\text{MSE}}(t, y) = \frac{1}{2} (t - y)^2$
Logistic regression	$\arg \min_{\mathbf{w}, b} \sum_i \mathcal{L}_{\sigma\text{-NLL}}(t_i, f(\mathbf{x}_i)) + \frac{1}{2} \lambda \ \mathbf{w}\ ^2$	$\mathcal{L}_{\sigma\text{-NLL}}(t, y) = -\log \left( \frac{\sigma(y)^t + (1 - \sigma(y))^{1-t}}{(1 - \sigma(y))^{1-t}} \right)$
Softmax regression	$\arg \min_{\mathbf{W}, b} \sum_i \mathcal{L}_{\text{s-NLL}}(t_i, f(\mathbf{x}_i)) + \frac{1}{2} \lambda \ \mathbf{w}\ ^2$	$\mathcal{L}_{\text{s-NLL}}(t, \mathbf{y}) = -\log \text{softmax}(\mathbf{y})_t$
SVM	$\arg \min_{\mathbf{w}, b} C \sum_i \mathcal{L}_{\text{hinge}}(t_i, f(\mathbf{x}_i)) + \frac{1}{2} \ \mathbf{w}\ ^2$	$\mathcal{L}_{\text{hinge}}(t, y) = \max(0, 1 - ty)$

Note that  $\mathcal{L}_{\text{MSE}}(t, y) \propto -\log(\mathcal{N}(t; \mu = y, \sigma^2 = 1))$  and that  $\mathcal{L}_{\sigma\text{-NLL}}(t, y) = \mathcal{L}_{\text{s-NLL}}(t, [y, 0])$ .

# Binary Classification Loss Functions Comparison

To compare various functions for binary classification, we need to formulate them all in the same settings, with  $t \in \{-1, 1\}$ .

- MSE:  $(ty - 1)^2$ , because it is  $(y - 1)^2$  for  $t = 1$  and  $(-y - t)^2$  for  $t = -1$
- LR:  $\sigma(ty)$ , because it is  $\sigma(y)$  for  $t = 1$  and  $1 - \sigma(y) = \sigma(-y)$  for  $t = -1$
- SVM:  $\max(0, 1 - ty)$



To solve the dual formulation of a SVM, usually Sequential Minimal Optimization (SMO; John Platt, 1998) algorithm is used.

Before we introduce it, we start by introducing **coordinate descent** optimization algorithm.

Consider solving unconstrained optimization problem

$$\arg \min_w L(w_1, w_2, \dots, w_D).$$

Instead of the usual SGD approach, we could optimize the weights one by one, using the following algorithm

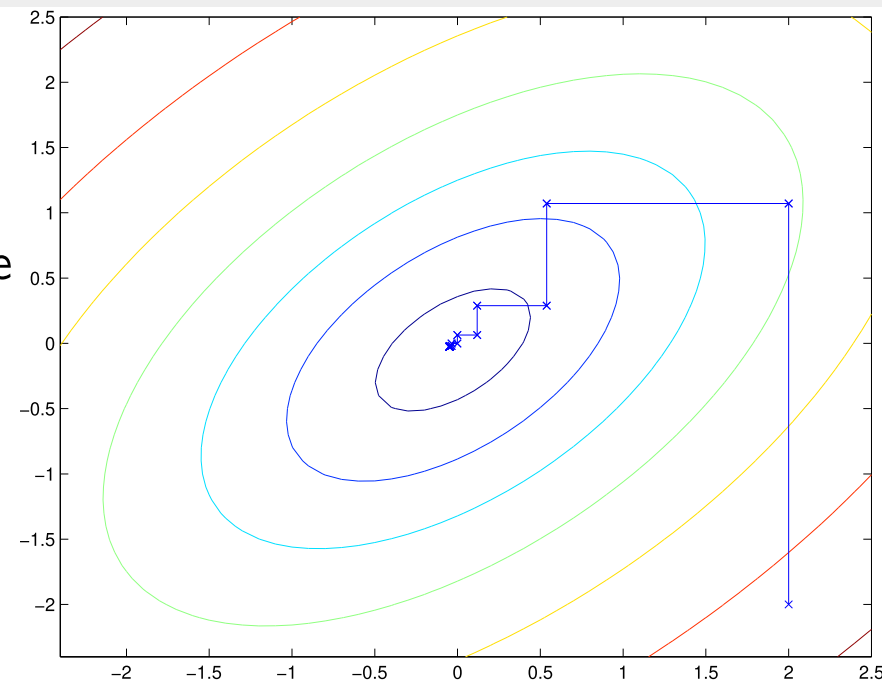
- loop until convergence
  - for  $i$  in  $\{1, 2, \dots, D\}$ :
    - $w_i \leftarrow \arg \min_{w_i} L(w_1, w_2, \dots, w_D)$



- loop until convergence
- for  $i$  in  $\{1, 2, \dots, D\}$ :
  - $w_i \leftarrow \arg \min_{w_i} L(w_1, w_2, \dots, w_D)$

If the inner  $\arg \min$  can be performed efficiently, the coordinate descent can be fairly efficient.

Note that we might want to choose  $w_i$  in different order, for example by trying to choose  $w_i$  providing the largest decrease of  $L$ .



CS229 Lecture 3 Notes, <http://cs229.stanford.edu/notes/cs229-notes3.pdf>

In soft-margin SVM, we try to minimize

$$L = \sum_i a_i - \frac{1}{2} \sum_i \sum_j a_i a_j t_i t_j K(\mathbf{x}_i, \mathbf{x}_j),$$

such that

$$\forall_i : C \geq a_i \geq 0 \text{ and } \sum_i a_i t_i = 0.$$

The KKT conditions for the solution can be reformulated (while staying equivalent) as

$$a_i > 0 \Rightarrow t_i y(\mathbf{x}_i) \leq 1, \text{ because } a_i > 0 \Rightarrow t_i y(\mathbf{x}_i) = 1 - \xi_i \text{ and we have } \xi_i \geq 0,$$

$$a_i < C \Rightarrow t_i y(\mathbf{x}_i) \geq 1, \text{ because } a_i < C \Rightarrow \mu_i > 0 \Rightarrow \xi_i = 0 \text{ and } t_i y(\mathbf{x}_i) \geq 1 - \xi_i,$$

$$0 < a_i < C \Rightarrow t_i y(\mathbf{x}_i) = 1, \text{ a combination of both.}$$

# Sequential Minimal Optimization Algorithm

At its core, the SMO algorithm is just a coordinate descent.

It tries to find such  $\alpha_i$  fulfilling the KKT conditions – for soft-margin SVM, KKT conditions are sufficient conditions for optimality (the loss is convex and inequality constraints affine).

However, note that because of the  $\sum a_i t_i = 0$  constraint we cannot optimize just one  $a_i$ , because a single  $a_i$  is determined from the others.

Therefore, in each step we pick two  $a_i, a_j$  coefficients and try to minimize the loss while fulfilling the constraints.

- loop until convergence (until  $\forall i : a_i < C \Rightarrow t_i y(\mathbf{x}_i) \geq 1$  and  $a_i > 0 \Rightarrow t_i y(\mathbf{x}_i) \leq 1$ )
  - for  $i$  in  $\{1, 2, \dots, D\}$ , for  $j \neq i$  in  $\{1, 2, \dots, D\}$ :
    - $a_i, a_j \leftarrow \arg \min_{a_i, a_j} L(a_1, a_2, \dots, a_D)$  such that  $C \geq a_i \geq 0, \sum_i a_i t_i = 0$

The SMO is an efficient algorithm, because we can compute the update to  $a_i, a_j$  efficiently, because there exists an closed form solution.

Assume that we are updating  $a_i$  and  $a_j$ . Then from the  $\sum_k a_k t_k = 0$  condition we can write  $a_i t_i = -\sum_{k \neq i} a_k t_k$ . Given that  $t_i^2 = 1$  and denoting  $\zeta = -\sum_{k \neq i, k \neq j} a_k t_k$ , we get

$$a_i = t_i(\zeta - a_j t_j).$$

Minimizing  $L(\mathbf{a})$  with respect to  $a_i$  and  $a_j$  then amounts to minimizing a quadratic function of  $a_j$ , which has an analytical solution.

Note that the real SMO algorithm has several heuristics for choosing  $a_i, a_j$  such that the  $L$  can be minimized the most.

**Input:** Dataset ( $\mathbf{X} \in \mathbb{R}^{N \times D}$ ,  $\mathbf{t} \in \{-1, 1\}^N$ ), kernel  $\mathbf{K}$ , regularization parameter  $C$ , tolerance  $tol$ ,  $max\_passes\_without\_a\_changing$  value

- Initialize  $a_i \leftarrow 0$ ,  $b \leftarrow 0$ ,  $passes \leftarrow 0$
- **while**  $passes < max\_passes\_without\_a\_changing$ :
  - $changed\_as \leftarrow 0$
  - **for**  $i$  in  $1, 2, \dots, N$ :
    - $E_i \leftarrow y(\mathbf{x}_i) - t_i$
    - **if** ( $a_i < C$  and  $t_i E_i < -tol$ ) or ( $a_i > 0$  and  $t_i E_i > tol$ ):
      - Choose  $j \neq i$  randomly
      - Update  $a_i$ ,  $a_j$  and  $b$
      - $changed\_as \leftarrow changed\_as + 1$
  - **if**  $changed\_as = 0$ :  $passes \leftarrow passes + 1$
  - **else**:  $passes \leftarrow 0$

**Input:** Dataset ( $\mathbf{X} \in \mathbb{R}^{N \times D}$ ,  $\mathbf{t} \in \{-1, 1\}^N$ ), kernel  $\mathbf{K}$ , regularization parameter  $C$ , tolerance  $tol$ , *max\_passes\_without\_a\_changing* value

- Update  $a_i$ ,  $a_j$ ,  $b$ :
  - Express  $a_i$  using  $a_j$
  - Find  $a_j$  optimizing the loss  $L$  quadratic with respect to  $a_j$
  - Clip  $a_j$  so that  $0 \leq a_i, a_j \leq C$
  - Compute corresponding  $a_i$
  - Compute  $b$  matching to updated  $a_i$ ,  $a_j$

# Primal versus Dual Formulation

Assume we have a dataset with  $N$  training examples, each with  $D$  features. Also assume the used feature map  $\varphi$  generates  $F$  features.

Property	Primal Formulation	Dual Formulation
Parameters	$F$	$N$
Model size	$F$	$s \cdot D$ for $s$ support vectors
Usual training time	$c \cdot N \cdot F$ for $c$ iterations	between $\Omega(ND)$ and $\mathcal{O}(N^2D)$
Inference time	$\Theta(F)$	$\Theta(s \cdot D)$ for $s$ support vectors

# Decision Trees

The idea of decision trees is to partition the input space into usually cuboid regions and solving each region with a simpler model.

We focus on **Classification and Regression Trees** (CART; Breiman et al., 1984), but there are additional variants like ID3, C4.5, ...

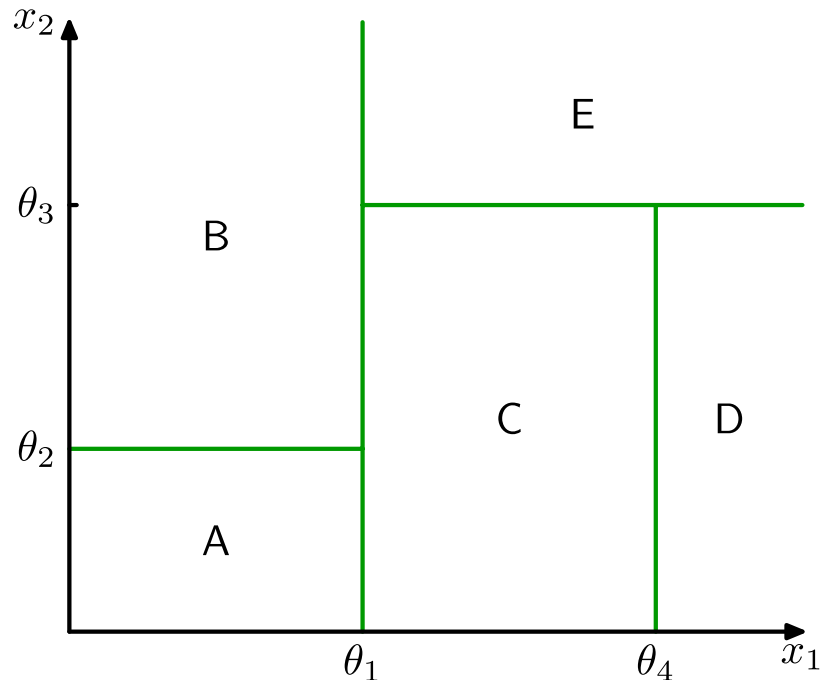


Figure 14.6 of Pattern Recognition and Machine Learning.

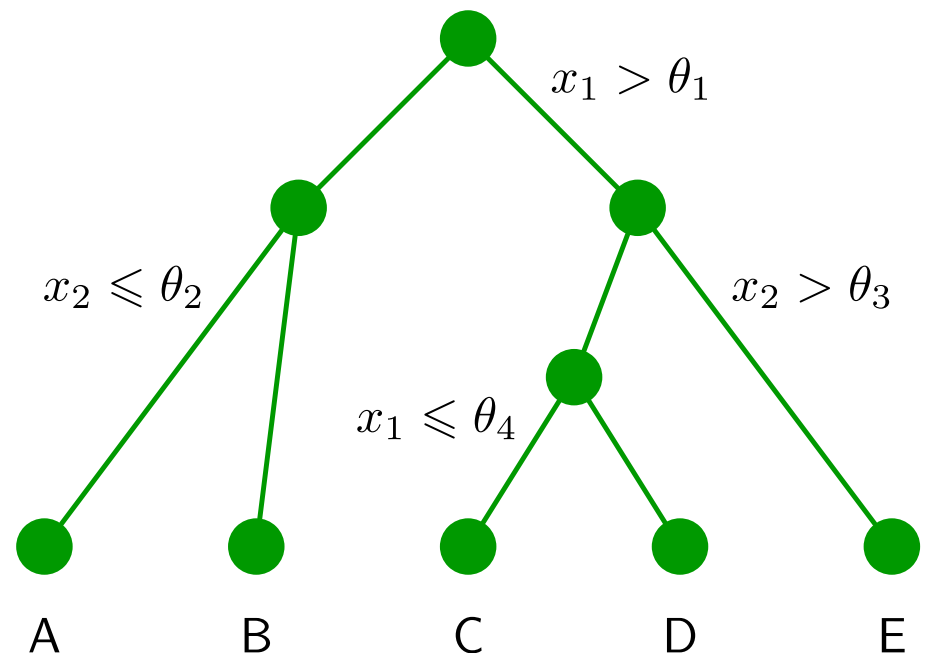


Figure 14.5 of Pattern Recognition and Machine Learning.



# Regression Decision Trees

Assume we have an input dataset  $\mathbf{X} \in \mathbb{R}^{N \times D}$ ,  $\mathbf{t} \in \mathbb{R}^N$ . At the beginning, the decision tree is just a single node and all input examples belong to this node. We denote  $I_{\mathcal{T}}$  the set of training example indices belonging to a leaf node  $\mathcal{T}$ .

For each leaf, our model will predict the average of the training examples belonging to that leaf,  $\hat{t}_{\mathcal{T}} = \frac{1}{|I_{\mathcal{T}}|} \sum_{i \in I_{\mathcal{T}}} t_i$ .

We will use a *criterion*  $c_{\mathcal{T}}$  telling us how *uniform* or *homogeneous* are the training examples belonging to a leaf node  $\mathcal{T}$  – for regression, we will employ the sum of squares error between the examples belonging to the node and the predicted value in that node; this is proportional to variance of the training examples belonging to the leaf node  $\mathcal{T}$ , multiplied by the number of the examples. Note that even if it not *mean* squared error, it is sometimes denoted as MSE.

$$c_{\text{SE}}(\mathcal{T}) \stackrel{\text{def}}{=} \sum_{i \in I_{\mathcal{T}}} (t_i - \hat{t}_{\mathcal{T}})^2, \text{ where } \hat{t}_{\mathcal{T}} = \frac{1}{|I_{\mathcal{T}}|} \sum_{i \in I_{\mathcal{T}}} t_i.$$

To split a node, the goal is to find a feature and its value such that when splitting a node  $\mathcal{T}$  into  $\mathcal{T}_L$  and  $\mathcal{T}_R$ , the resulting regions decrease the overall criterion value the most, i.e., the difference  $c_{\mathcal{T}_L} + c_{\mathcal{T}_R} - c_{\mathcal{T}}$  is the lowest.

Usually we have several constraints, we mention on the most common ones:

- **maximum tree depth:** we do not split nodes with this depth;
- **minimum examples to split:** we only split nodes with this many training examples;
- **maximum number of leaf nodes**

The tree is usually built in one of two ways:

- if the number of leaf nodes is unlimited, we usually build the tree in a depth-first manner, recursively splitting every leaf until some above constraint is invalidated;
- if the maximum number of leaf nodes is give, we usually split such leaf  $\mathcal{T}$  where the criterion difference  $c_{\mathcal{T}_L} + c_{\mathcal{T}_R} - c_{\mathcal{T}}$  is the lowest.

# Classification Decision Trees

For multi-class classification, we predict such class most frequent in the training examples belonging to a leaf  $\mathcal{T}$ .

To define the criterions, let us denote the average probability for class  $k$  in a region  $\mathcal{T}$  at  $p_{\mathcal{T}}(k)$ .

For classification trees, one of the following two criterions is usually used:

- **Gini index:**

$$c_{\text{Gini}}(\mathcal{T}) \stackrel{\text{def}}{=} |I_{\mathcal{T}}| \sum_k p_{\mathcal{T}}(k)(1 - p_{\mathcal{T}}(k))$$

- **Entropy Criterion**

$$c_{\text{entropy}}(\mathcal{T}) \stackrel{\text{def}}{=} |I_{\mathcal{T}}| H(p_{\mathcal{T}}) = -|I_{\mathcal{T}}| \sum_k p_{\mathcal{T}}(k) \log p_{\mathcal{T}}(k)$$