Soft-margin SVM, SMO

Milan Straka

November 15, 2021

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

Support Vector Machines

In order to solve the constrained problem of

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

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

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

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

$$\boldsymbol{w} = \sum_i a_i t_i \varphi(\boldsymbol{x}_i),$$

$$0 = \sum_i a_i t_i.$$

Substituting these to the Lagrangian, we want to maximize

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

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

The solution will fulfill the KKT conditions, meaning that

$$a_i \geq 0, \qquad t_i y(\boldsymbol{x}_i) - 1 \geq 0, \qquad a_i \big(t_i y(\boldsymbol{x}_i) - 1\big) = 0.$$

Therefore, either a point $\boldsymbol{x}_i$ is on a boundary, or $a_i = 0$. Given that the prediction for $\boldsymbol{x}$ is $y(\boldsymbol{x}) = \sum_i a_i t_i K(\boldsymbol{x}, \boldsymbol{x}_i) + b$, we only need to keep the training points $\boldsymbol{x}_i$ that are on the boundary, the so-called **support vectors**. Therefore, even though SVM is a nonparametric model, it needs to store only a subset of the training data.

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.

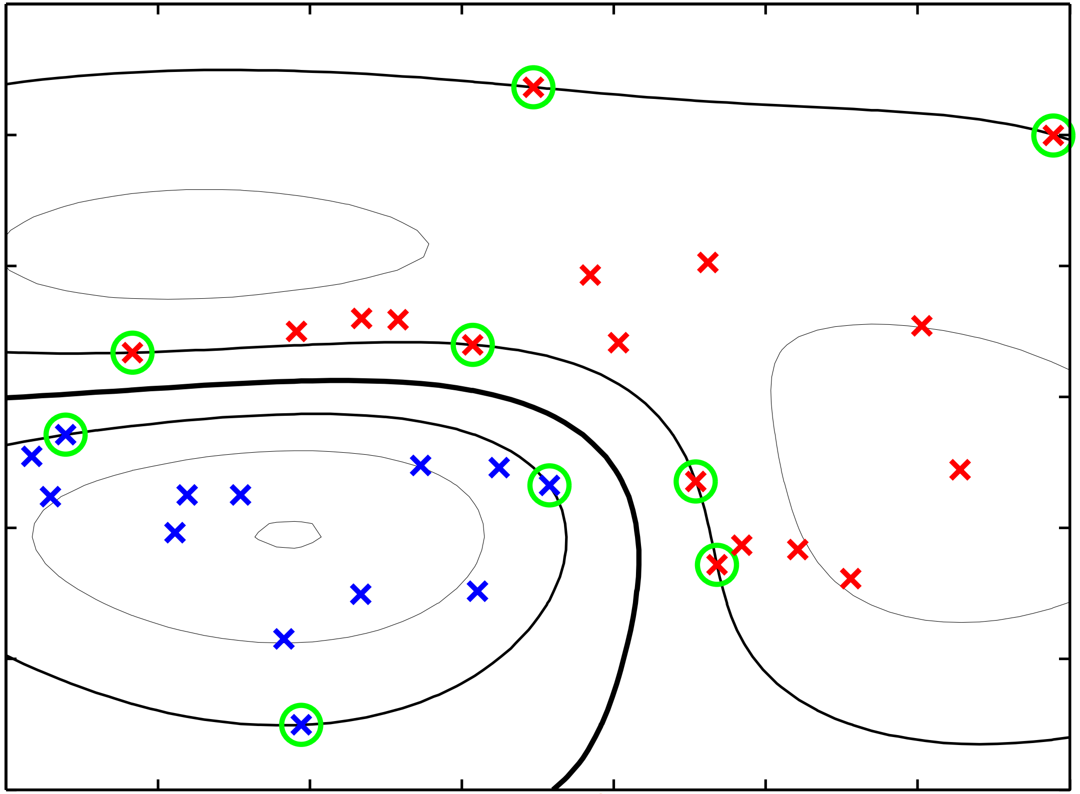


Figure 7.2 of Pattern Recognition and Machine Learning.

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

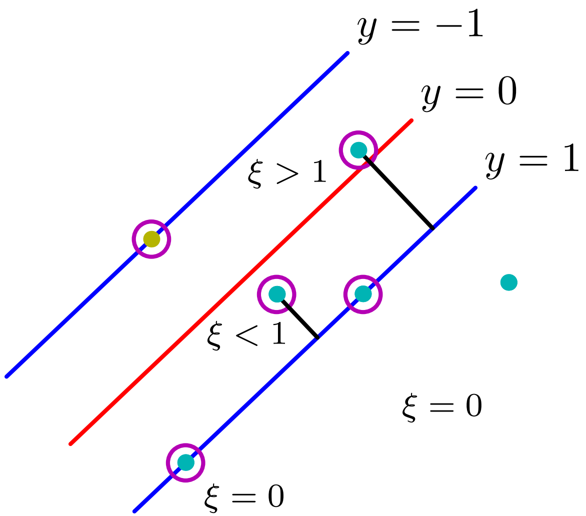


Figure 7.3 of Pattern Recognition and Machine Learning.

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

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

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

Support Vector Machines for Non-linearly Separable Data

To solve the soft-margin variant, we again create a Lagrangian, this time with two sets of multipliers $\boldsymbol{a} = (a_1, \ldots, a_N)$ and $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_N)$:

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

Solving for the critical points and substituting for $\boldsymbol{w}$, 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

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

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

$$\forall\, i : C \geq a_i \geq 0 \;\text{ and }\; \sum_i a_i t_i = 0.$$

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

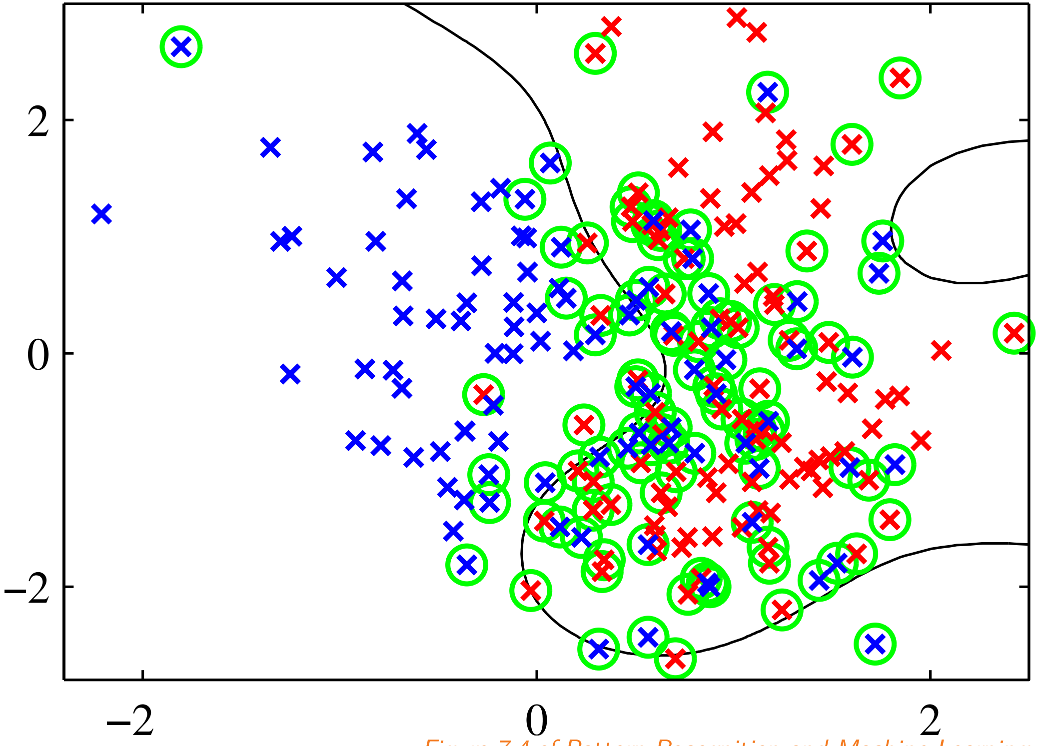


Figure 7.4 of Pattern Recognition and Machine Learning.

SGD-like Formulation of Soft-Margin SVM

Note that the slack variables can be written as

$$\xi_i = \max\big(0, 1 - t_i y(\boldsymbol x_i)\big),$$

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

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

to

$$\operatorname*{arg\,min}_{\boldsymbol w, b} C \sum_i \mathcal{L}_\textrm{hinge}\big(t_i, y(\boldsymbol x_i)\big) + \frac{1}{2}\|\boldsymbol w\|^2.$$

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

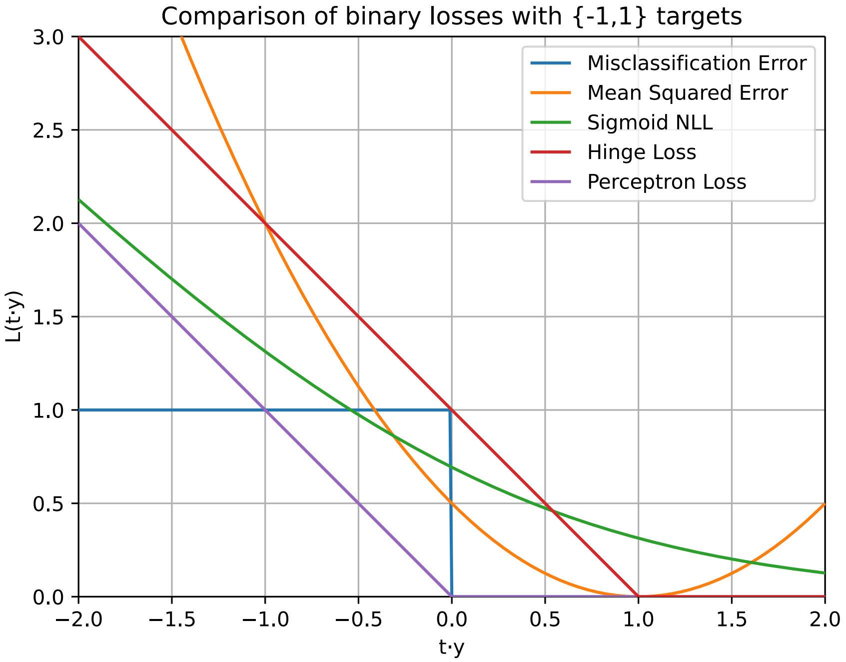
Model	**Objective**	**Per-Instance Loss**
Linear Regression	$\arg\min_{\boldsymbol{w},b} \sum_i \mathcal{L}_{\text{MSE}}\big(t_i, y(\boldsymbol{x}_i)\big) + \frac{1}{2}\lambda\|\boldsymbol{w}\|^2$	$\mathcal{L}_{\text{MSE}}(t, y) = \frac{1}{2}(t - y)^2$
Logistic regression	$\arg\min_{\boldsymbol{w},b}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

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

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 $(y + 1)^2 = (-y - 1)^2$ for $t = -1$,
- LR: $-\log \sigma(ty)$, 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 the Sequential Minimal Optimization (SMO; John Platt, 1998) algorithm is used.

Before we introduce it, we start with the **coordinate descent** optimization algorithm.

Consider solving unconstrained optimization problem

$$\arg\min_{\boldsymbol{w}} L(w_1, w_2, \ldots, w_D).$$

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

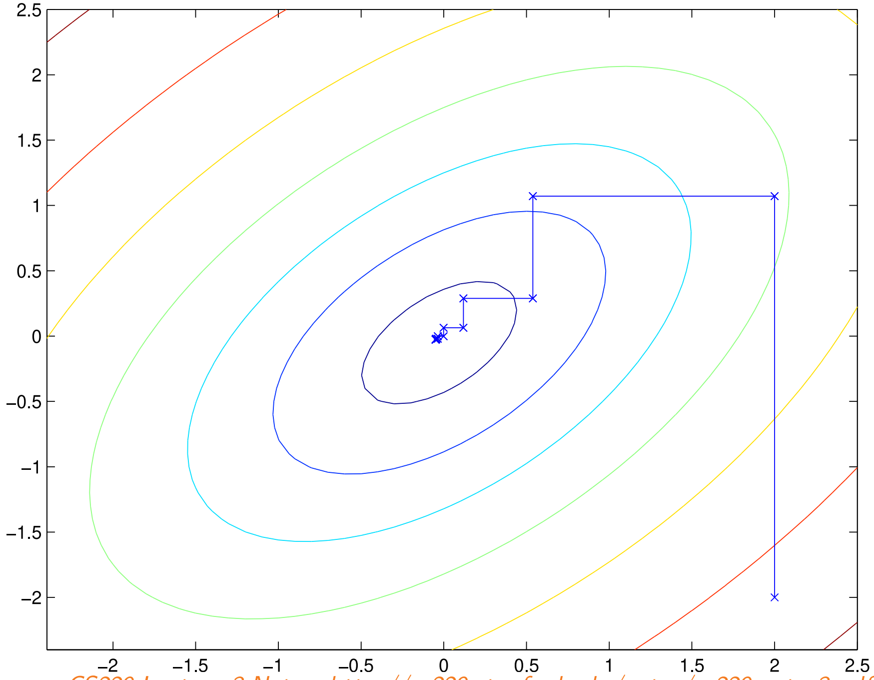
- loop until convergence
  - for i in $\{1, 2, \ldots, D\}$:
    - $w_i \leftarrow \arg\min_{w_i} L(w_1, w_2, \ldots, w_D)$

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

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 a different order, for example by trying to choose w_i providing the largest decrease of L.

The Kernel linear regression dual formulation with single-example batches was in fact trained by a coordinate descent – updating a single β_i corresponds to updating weights for a single example.



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

In soft-margin SVM, we try to maximize

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

with respect to a_i, 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

$$\begin{aligned} a_i > 0 &\Rightarrow t_i y(\boldsymbol{x}_i) \leq 1, && \text{because } a_i > 0 \Rightarrow t_i y(\boldsymbol{x}_i) = 1 - \xi_i \text{ and we have } \xi_i \geq 0, \\ a_i < C &\Rightarrow t_i y(\boldsymbol{x}_i) \geq 1, && \text{because } a_i < C \Rightarrow \mu_i > 0 \Rightarrow \xi_i = 0 \text{ and } t_i y(\boldsymbol{x}_i) \geq 1 - \xi_i, \\ 0 < a_i < C &\Rightarrow t_i y(\boldsymbol{x}_i) = 1, && \text{a combination of both.} \end{aligned}$$

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

It tries to find a_i maximizing $\mathcal{L}$ while fulfilling the KKT conditions – once found, an optimum has been reached, given that for soft-margin SVM the KKT conditions are sufficient conditions for optimality (for soft-margin SVM, the loss is convex and the inequality constraints are not only convex but even 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 maximize the loss while fulfilling the constraints.

- loop until convergence (until $\forall\, i : a_i < C \Rightarrow t_i y(\boldsymbol{x}_i) \geq 1$ and $a_i > 0 \Rightarrow t_i y(\boldsymbol{x}_i) \leq 1$)
  - for i in $\{1, 2, \ldots, N\}$:
    - choose $j \neq i$ in $\{1, 2, \ldots, N\}$
    - $a_i, a_j \leftarrow \arg\max_{a_i, a_j} \mathcal{L}(a_1, a_2, \ldots, a_N)$, while respecting the constraints:
      - $0 \leq a_i \leq C$, $0 \leq a_j \leq C$, $\sum_i a_i t_i = 0$

Sequential Minimal Optimization Algorithm

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

Assume that we are updating a_i and a_j. Then using the condition $\sum_k a_k t_k = 0$ 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

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

Maximizing $\mathcal{L}(\boldsymbol{a})$ with respect to a_i and a_j then amounts to maximizing a quadratic function of a_j, which has an analytical solution.

Note that the real SMO algorithm employs several heuristics for choosing a_i, a_j such that the $\mathcal{L}$ can be maximized the most.

Sequential Minimal Optimization Algorithm Part I

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

- Initialize $a_i \leftarrow 0$, $b \leftarrow 0$, $passes \leftarrow 0$
- **while** $passes < max\_passes\_without\_as\_changing$ (or we run out of patience):
  - $changed\_as \leftarrow 0$
  - **for** i in $1, 2, \ldots, N$:
    - $E_i \leftarrow y(\boldsymbol{x}_i) - t_i$
    - **if** ($a_i < C - tol$ **and** $t_i E_i < -tol$) **or** ($a_i > tol$ **and** $t_i E_i > tol$):
      - choose $j \neq i$ randomly
      - try updating a_i, a_j to maximize $\mathcal{L}(a_1, a_2, \ldots, a_N)$ such that $0 \leq a_k \leq C$ and $\sum_i a_i t_i = 0$; if successful, set b fo fulfill the KKT conditions and set $changed\_as \leftarrow changed\_as + 1$
  - $passes \leftarrow 0$ **if** $changed\_as$ **else** $passes + 1$

Sequential Minimal Optimization Update Rules

We already know that $a_i = t_i(\zeta - a_j t_j)$.

To find a_j maximizing $\mathcal{L}$, we use the formula for locating a vertex of a parabola

$$a_j^{\mathrm{new}} \leftarrow a_j - \frac{\partial \mathcal{L}/\partial a_j}{\partial^2 \mathcal{L}/\partial a_j^2},$$

which is in fact one Newton-Raphson iteration step.

Denoting $E_j \stackrel{\text{def}}{=} y(\boldsymbol{x}_j) - t_j$, we can compute the first derivative as

$$\frac{\partial \mathcal{L}}{\partial a_j} = t_j(E_i - E_j),$$

and the second derivative as

$$\frac{\partial^2 \mathcal{L}}{\partial a_j^2} = 2K(\boldsymbol{x}_i, \boldsymbol{x}_j) - K(\boldsymbol{x}_i, \boldsymbol{x}_i) - K(\boldsymbol{x}_j, \boldsymbol{x}_j) = -\left\|\varphi(\boldsymbol{x}_i) - \varphi(\boldsymbol{x}_j)\right\|^2 \leq 0.$$

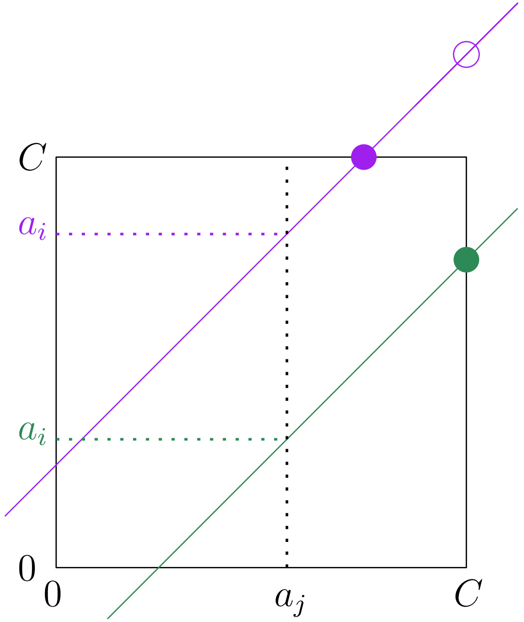
If the second derivative is strictly negative, we know that the vertex is really a maximum, in which case we get

$$a_j^{\text{new}} \leftarrow a_j - t_j \frac{E_i - E_j}{2K(\boldsymbol{x}_i, \boldsymbol{x}_j) - K(\boldsymbol{x}_i, \boldsymbol{x}_i) - K(\boldsymbol{x}_j, \boldsymbol{x}_j)}.$$

However, our maximization is constrained – it must hold that $0 \leq a_i \leq C$ and $0 \leq a_j \leq C$.

Recalling that $a_i = -t_i t_j a_j + \text{const}$, we can plot the dependence of a_i and a_j. If for example $-t_i t_j = 1$ and $a_j^{\text{new}} > C$, we need to find the "right-most" solution fulfilling both $a_i \leq C$ and $a_j \leq C$. Such a solution is either:

- when a_j^{new} is clipped to C, as in the green case in the example,
- when a_j^{new} is clipped so that $a_i^{\text{new}} = C$ (the purple case in the example), in which case $a_j^{\text{new}} = a_j + (C - a_i)$.



Sequential Minimal Optimization Update Rules

If we consider both $t_i t_j = \pm 1$ and $a_j^{\text{new}} < 0$, $a_j^{\text{new}} > C$, we get that the value maximizing the Lagrangian is a_j^{new} clipped to range $[L, H]$, where

$$\begin{aligned}
t_i = t_j &\Rightarrow L = \max(0, a_i + a_j - C), H = \min(C, a_i + a_j) \\
t_i \neq t_j &\Rightarrow L = \max(0, a_j - a_i), H = \min(C, C + a_j - a_i).
\end{aligned}$$

After obtaining a_j^{new} we can compute a_i^{new}. Remembering that $a_i = -t_i t_j a_j + \text{const}$, we can compute it efficiently as

$$a_i^{\text{new}} \leftarrow a_i - t_i t_j \big(a_j^{\text{new}} - a_j\big).$$

Sequential Minimal Optimization Update Rules

To arrive at the bias update, we consider the KKT condition that for $0 < a_j^\textrm{new} < C$, it must hold that $1 = t_j y(\boldsymbol x_j) = t_j \big[\big(\sum_l a_l^\textrm{new} t_l K(\boldsymbol x_j, \boldsymbol x_l)\big) + b^\textrm{new}\big]$. Combining it with the fact that $\big(\sum_l a_l t_l K(\boldsymbol x_j, \boldsymbol x_l)\big) + b = E_j + t_j$, we obtain

$$b_j^\textrm{new} = b - E_j - t_i(a_i^\textrm{new} - a_i)K(\boldsymbol x_i, \boldsymbol x_j) - t_j(a_j^\textrm{new} - a_j)K(\boldsymbol x_j, \boldsymbol x_j).$$

Analogously for $0 < a_i^\textrm{new} < C$ we get

$$b_i^\textrm{new} = b - E_i - t_i(a_i^\textrm{new} - a_i)K(\boldsymbol x_i, \boldsymbol x_i) - t_j(a_j^\textrm{new} - a_j)K(\boldsymbol x_j, \boldsymbol x_i).$$

Finally, if $a_j^\textrm{new}, a_i^\textrm{new} \in \{0, C\}$, we know that all values between $b_i^\textrm{new}$ and $b_j^\textrm{new}$ fulfill the KKT conditions. We therefore arrive at the following update for bias:

$$b^\textrm{new} = \begin{cases} b_i^\textrm{new} & \textrm{if}~~0 < a_i^\textrm{new} < C, \\ b_j^\textrm{new} & \textrm{if}~~0 < a_j^\textrm{new} < C, \\ (b_i^\textrm{new} + b_j^\textrm{new})/2 & \textrm{otherwise.} \end{cases}$$

Sequential Minimal Optimization Algorithm Part II

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

- Try updating a_i, a_j and b to fulfill the KKT conditions:
  - Find a_j maximizing $\mathcal{L}$, in which we express a_i using a_j.
    - Such $\mathcal{L}$ is a quadratic function of a_j.
    - If the second derivative of $\mathcal{L}$ is not negative, stop.
  - Clip a_j so that $0 \leq a_i \leq C$ and $0 \leq a_j \leq C$.
    - If we did not make enough progress (the new a_j is very similar), revert the value of a_j and stop.
  - Compute corresponding a_i.
  - Compute b appropriate to the 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 φ 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 $\mathcal{O}(N^2 D)$
Inference time	$\Theta(F)$	$\Theta(s \cdot D)$ for s support vectors

The SVM algorithm with RBF kernel implements a better variant of the k-NN algorithm, weighting “evidence” of training data points according to their distance.

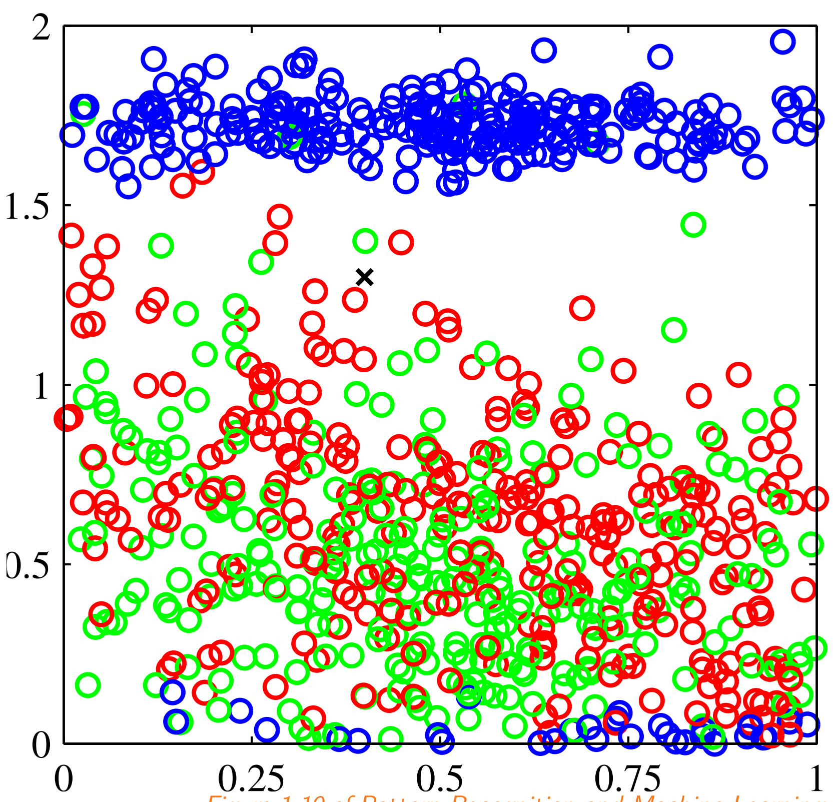


Figure 1.19 of Pattern Recognition and Machine Learning.

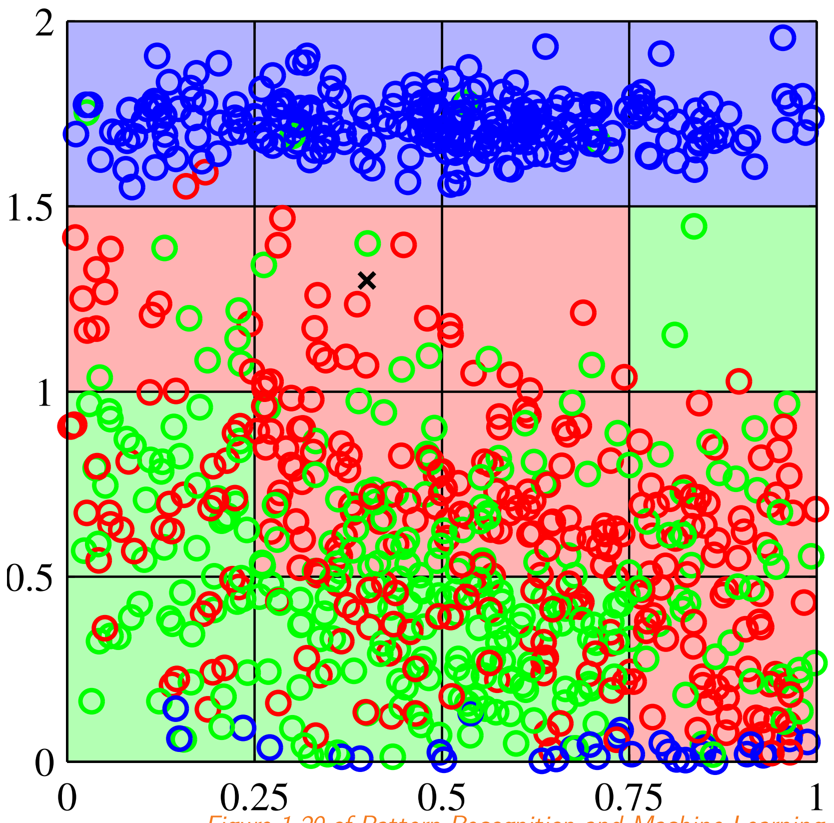


Figure 1.20 of Pattern Recognition and Machine Learning.

Multiclass SVM

There are two general approaches for building a K-class classifier by combining several binary classifiers:

- **one-versus-rest** scheme: K binary classifiers are constructed, the i-th separating instances of class i from all others; during prediction, the one with the highest probability is chosen
  - the binary classifiers need to return calibrated probabilities (not SVM)
- **one-versus-one** scheme: $\binom{K}{2}$ binary classifiers are constructed, one for each (i, j) pair of class indices; during prediction, the class with the majority of votes wins (used by SVM)

However, voting suffers from serious difficulties, because when the binary classifiers are trained independently, usually large regions in the feature space recieve tied votes (and such regions are then ambiguous).

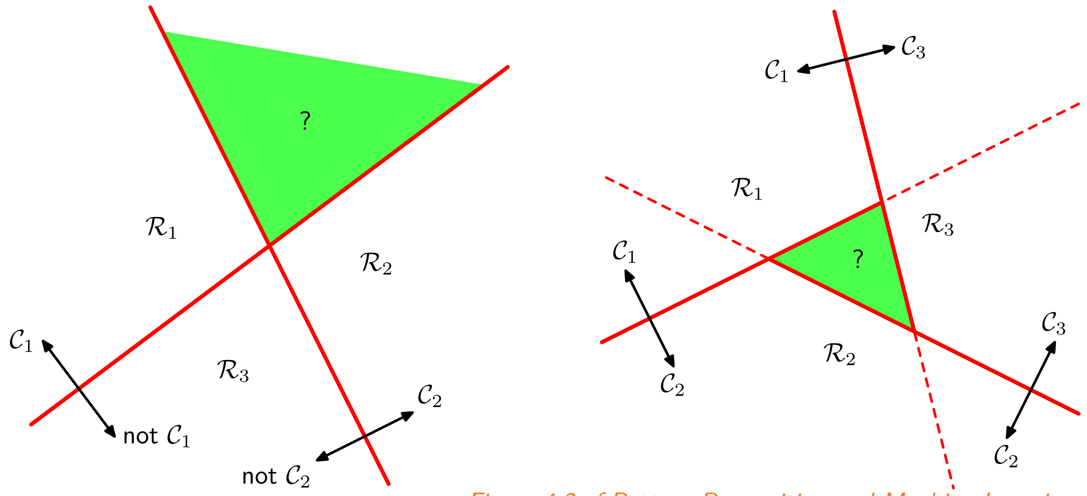
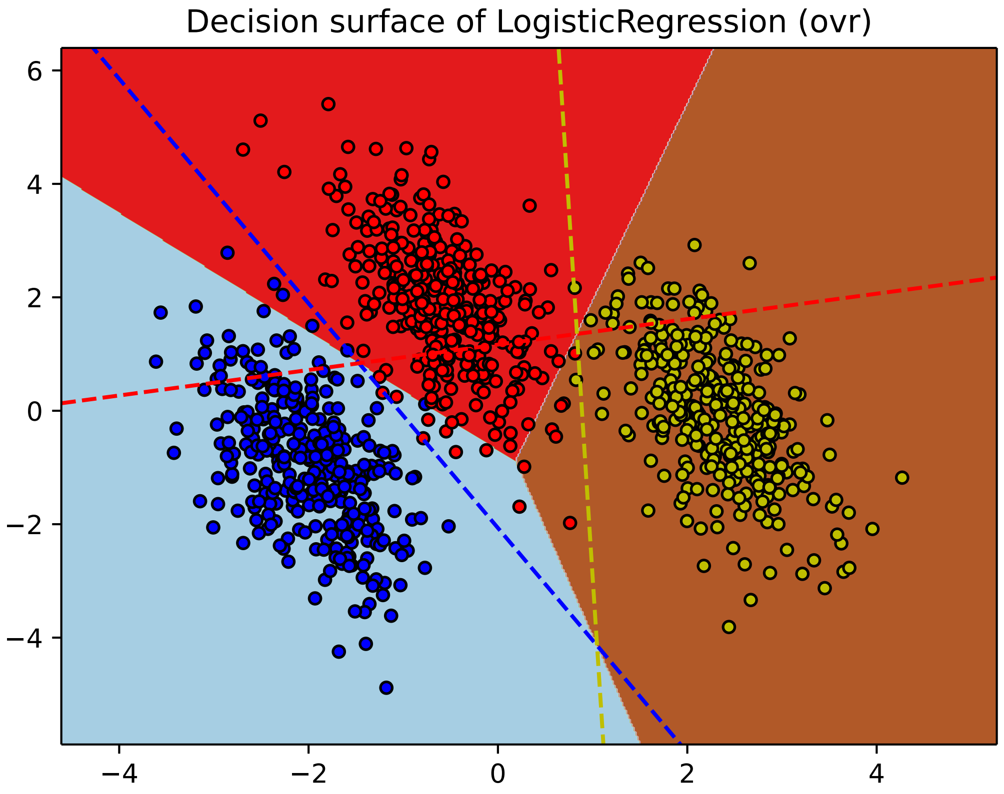
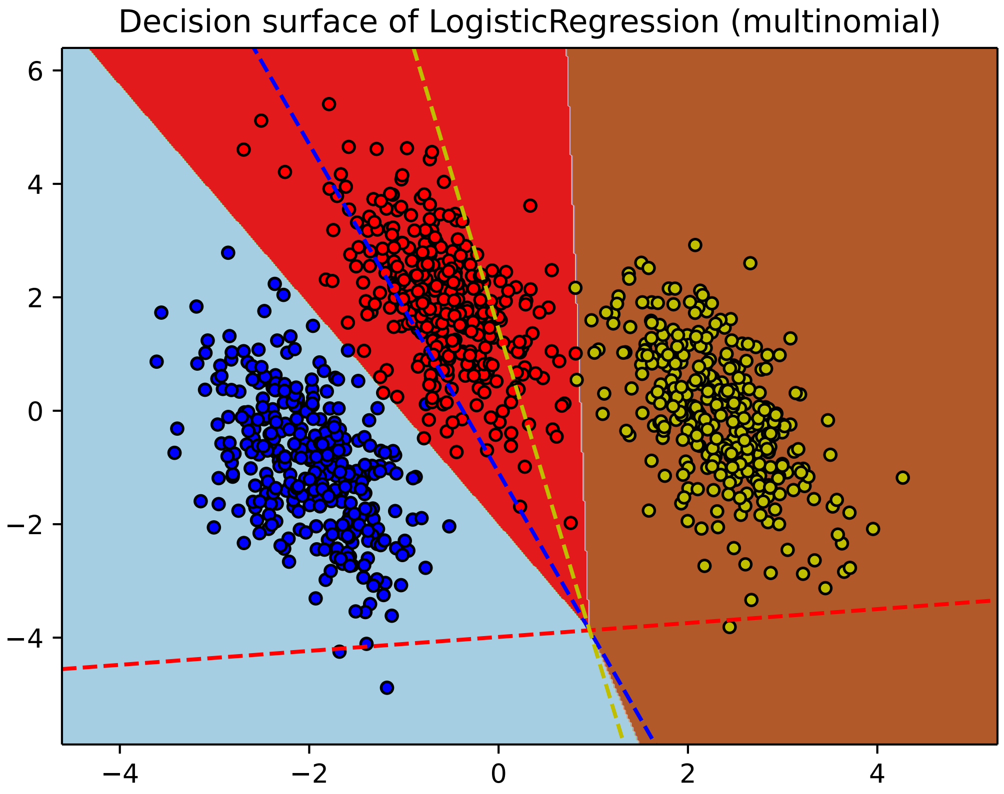


Figure 4.2 of Pattern Recognition and Machine Learning.

One-Versus-Rest Compared to Softmax Classification



https://scikit-learn.org/stable/auto_examples/linear_model/plot_logistic_multinomial.html



https://scikit-learn.org/stable/auto_examples/linear_model/plot_logistic_multinomial.html

The idea of SVM for regression is to use an ε-insensitive error function

$$\mathcal{L}_\varepsilon\big(t, y(\boldsymbol{x})\big) = \max\big(0, |y(\boldsymbol{x}) - t| - \varepsilon\big).$$

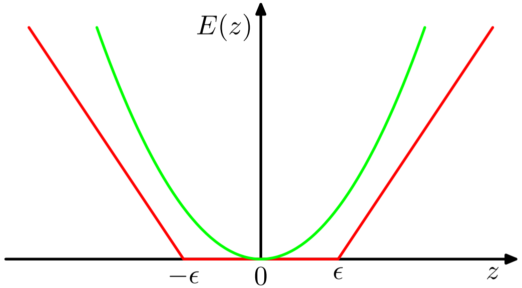


Figure 7.6 of Pattern Recognition and Machine Learning.

The primary formulation of the loss is then

$$C \sum_i \mathcal{L}_\varepsilon\big(t_i, y(\boldsymbol{x}_i)\big) + \frac{1}{2}\|\boldsymbol{w}\|^2.$$

In the dual formulation, we require every training example to be within ε of its target, but introduce two slack variables $\boldsymbol{\xi}^-$, $\boldsymbol{\xi}^+$ to allow outliers. We therefore minimize the loss

$$C \sum_i (\xi_i^- + \xi_i^+) + \tfrac{1}{2}\|\boldsymbol{w}\|^2$$

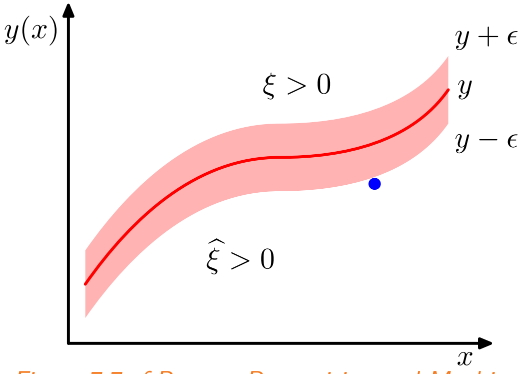


Figure 7.7 of Pattern Recognition and Machine Learning.

while requiring for every example $t_i - \varepsilon - \xi_i^- \leq y(\boldsymbol{x}_i) \leq t_i + \varepsilon + \xi_i^+$ for $\xi_i^- \geq 0, \xi_i^+ \geq 0$.

The Lagrangian after substituting for $\boldsymbol w$, b, $\boldsymbol\xi^-$ and $\boldsymbol\xi^+$ is

$$\mathcal{L} = \sum_i (a_i^+ - a_i^-) t_i - \varepsilon \sum_i (a_i^+ + a_i^-) - \frac{1}{2} \sum_i \sum_j (a_i^+ - a_i^-)(a_j^+ - a_j^-) K(\boldsymbol x_i, \boldsymbol x_j)$$

subject to

$$0 \le a_i^+, a_i^- \le C,$$
$$\sum_i (a_i^+ - a_i^-) = 0.$$

The prediction is then given by

$$y(\boldsymbol z) = \sum_i (a_i^+ - a_i^-) K(\boldsymbol z, \boldsymbol x_i) + b.$$

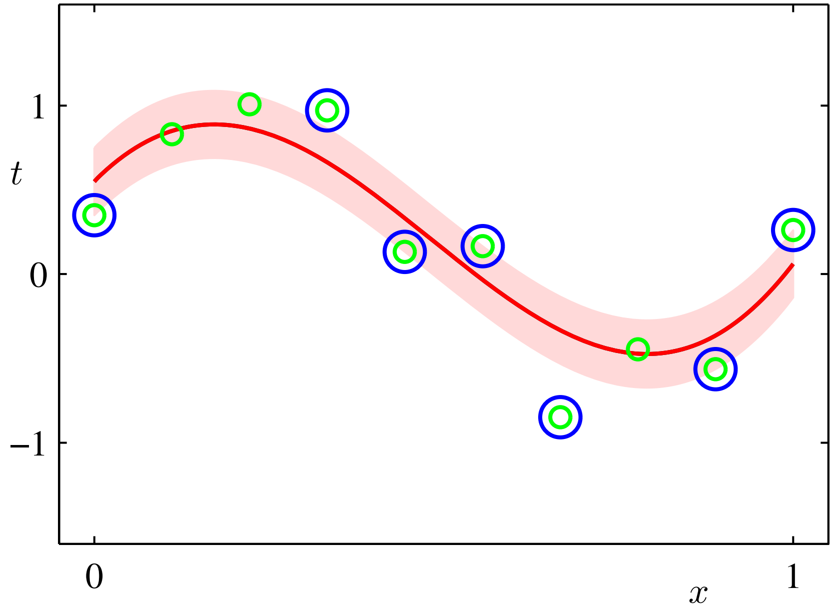


Figure 7.8 of Pattern Recognition and Machine Learning.

SVM Demos

- https://cs.stanford.edu/~karpathy/svmjs/demo/

MLP Demos

- https://cs.stanford.edu/~karpathy/svmjs/demo/demonn.html
- https://playground.tensorflow.org