


Kernel Methods, SVM

Milan Straka

 November 09, 2020



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



unless otherwise stated

Consider linear regression with cubic features

$$\varphi(\mathbf{x}) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \dots \\ x_1^2 \\ x_1 x_2 \\ \dots \\ x_2 x_1 \\ \dots \\ x_1^3 \\ x_1^2 x_2 \\ \dots \end{bmatrix} .$$

The SGD update of a linear regression with batch with indices \mathbf{b} is then

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\alpha}{|\mathbf{b}|} \sum_{i \in \mathbf{b}} (\varphi(\mathbf{x}_i)^T \mathbf{w} - t_i) \varphi(\mathbf{x}_i).$$

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

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \frac{\alpha}{|\mathbf{b}|} \sum_{i \in \mathbf{b}} (\varphi(\mathbf{x}_i)^T \mathbf{w} - t_i) \varphi(\mathbf{x}_i) \\ &\leftarrow \sum_i \left(\beta_i - [i \in \mathbf{b}] \cdot \frac{\alpha}{|\mathbf{b}|} (\varphi(\mathbf{x}_i)^T \mathbf{w} - t_i) \right) \varphi(\mathbf{x}_i). \end{aligned}$$

Every β_i for $i \in \mathbf{b}$ changes to $\beta_i - \frac{\alpha}{|\mathbf{b}|} (\varphi(\mathbf{x}_i)^T \mathbf{w} - t_i)$, so after substituting for \mathbf{w} we get

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

We can formulate an alternative linear regression algorithm (a so-called **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 $\mathbf{K}_{i,j} = \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_j)$
- Repeat until convergence
 - Sample a batch \mathbf{b} (usually by generating a random permutation and splitting it)
 - Simultaneously for all $i \in \mathbf{b}$ (the β_j on the right side must not be modified during the batch update):
 - $\beta_i \leftarrow \beta_i - \frac{\alpha}{|\mathbf{b}|} \left(\sum_j (\beta_j \mathbf{K}_{i,j}) - t_i \right)$

The predictions are then performed by computing

$$y(\mathbf{z}) = \varphi(\mathbf{z})^T \mathbf{w} = \sum_i \beta_i \varphi(\mathbf{z})^T \varphi(\mathbf{x}_i).$$

Bias in Kernel Linear Regression

Until now we did not consider *bias*. Unlike the usual formulation, where we can “hide” it in the weights, we usually handle it manually in the dual formulation.

Specifically, if we want to include bias in kernel linear regression, we modify the predictions to

$$y(\mathbf{z}) = \varphi(\mathbf{z})^T \mathbf{w} + b = \sum_i \beta_i \varphi(\mathbf{z})^T \varphi(\mathbf{x}_i) + b$$

and update the bias b separately.

The bias can be updated by SGD, which is what we did in the algorithms until now; however, if we are considering a bias of an “output layer”, we can even estimate it as the mean of the training targets before the training of the rest of the weights.

Kernel Trick

A single SGD update of all β_i then takes $\mathcal{O}(N^2)$, given that we can evaluate scalar dot product of $\varphi(\mathbf{x})^T \varphi(\mathbf{z})$ quickly.

$$\begin{aligned}
 \varphi(\mathbf{x})^T \varphi(\mathbf{z}) &= 1 + \sum_i \mathbf{x}_i z_i + \sum_{i,j} \mathbf{x}_i \mathbf{x}_j z_i z_j + \sum_{i,j,k} \mathbf{x}_i \mathbf{x}_j \mathbf{x}_k z_i z_j z_k \\
 &= 1 + \sum_i \mathbf{x}_i z_i + \left(\sum_i \mathbf{x}_i z_i \right)^2 + \left(\sum_i \mathbf{x}_i z_i \right)^3 \\
 &= 1 + \mathbf{x}^T \mathbf{z} + \left(\mathbf{x}^T \mathbf{z} \right)^2 + \left(\mathbf{x}^T \mathbf{z} \right)^3.
 \end{aligned}$$

We define a **kernel** corresponding to a feature map φ as a function

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

There exist quite a lot of kernels, but the most commonly used are the following:

- **Polynomial kernel of degree d** , also called *homogenous polynomial kernel*

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

which corresponds to a feature map returning all combinations of exactly d input features.

Using $(a_1 + \dots + a_k)^d = \sum_{n_i \geq 0, \sum n_i = d} \binom{d}{n_1, \dots, n_k} a_1^{n_1} \dots a_k^{n_k}$, we can verify that

$$\varphi(\mathbf{x}) = \left(\sqrt{\gamma^d \binom{d}{n_1, \dots, n_D}} x_1^{n_1} \dots x_D^{n_D} \right)_{n_i \geq 0, \sum n_i = d}.$$

For example, for $d = 2$, $\varphi(x_1, x_2) = \gamma(x_1^2, \sqrt{2}x_1x_2, x_2^2)$.

- **Polynomial kernel of degree at most d** , also called *nonhomogenous polynomial kernel*

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

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

Given that $(\gamma \mathbf{x}^T \mathbf{z} + 1)^d = \sum_i \binom{d}{i} (\gamma \mathbf{x}^T \mathbf{z})^i$, it is not difficult to derive that

$$\varphi(\mathbf{x}) = \left(\sqrt{\gamma^{d-n_{D+1}} \binom{d}{n_1, \dots, n_{D+1}}} x_1^{n_1} \cdots x_D^{n_D} \right)_{n_i \geq 0, \sum_{i=1}^{D+1} n_i = d}.$$

For example, for $d = 2$, $\varphi(x_1, x_2) = (1, \sqrt{2\gamma}x_1, \sqrt{2\gamma}x_2, \gamma x_1^2, \sqrt{2\gamma}x_1x_2, \gamma x_2^2)$.

- **Gaussian Radial basis function (RBF) kernel**

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

corresponds to a scalar product in an infinite-dimensional space; it is a combination of polynomial kernels of all degrees. Assuming $\gamma = 1$ for simplicity, we get

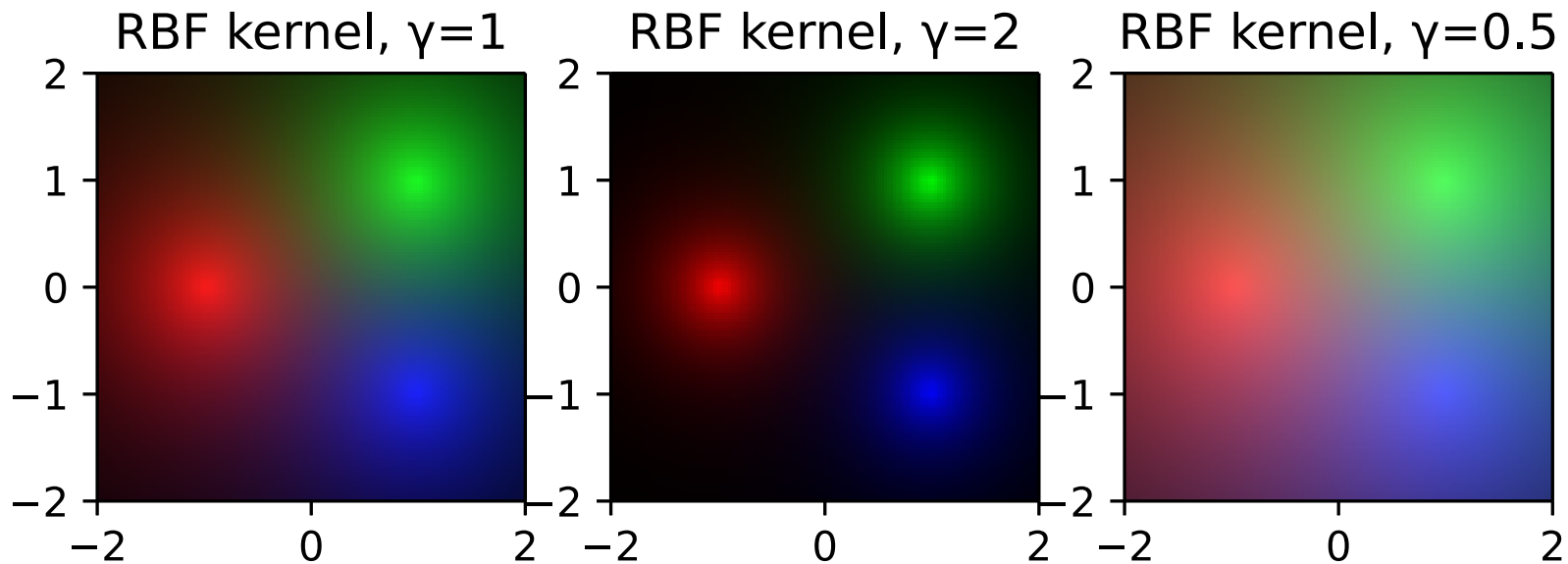
$$e^{-\|\mathbf{x} - \mathbf{z}\|^2} = e^{-\|\mathbf{x}\|^2 + 2\mathbf{x}^T \mathbf{z} - \|\mathbf{z}\|^2} = \sum_{d=0}^{\infty} \frac{(2\mathbf{x}^T \mathbf{z})^d}{d!} e^{-\|\mathbf{x}\|^2 - \|\mathbf{z}\|^2} = \sum_{d=0}^{\infty} \frac{2^d e^{-\|\mathbf{x}\|^2 - \|\mathbf{z}\|^2}}{d!} \left(\mathbf{x}^T \mathbf{z}\right)^d,$$

which is a combination of polynomial kernels; therefore, the feature map corresponding to the RBF kernel is

$$\varphi(\mathbf{x}) = \left(e^{-\gamma \|\mathbf{x}\|^2} \sqrt{\frac{(2\gamma)^d}{d!} \binom{d}{n_1, \dots, n_D}} x_1^{n_1} \cdots x_D^{n_D} \right)_{d=0,1,\dots,\infty, n_i \geq 0, \sum_{i=1}^D n_i = d}.$$

Note that the RBF kernel is a function of distance – it “weights” more similar examples more strongly. We could interpret it as an extended version of k-nearest neighbor algorithm, one which considers all examples, each weighted by similarity.

For illustration, we plot RBF kernel values to three points $(0, -1)$, $(1, 1)$ and $(1, -1)$ with different values of γ :



Support Vector Machines

Let us return to a binary classification task. The perceptron algorithm guaranteed finding some separating hyperplane if it existed; we now consider finding the one with *maximum margin*.

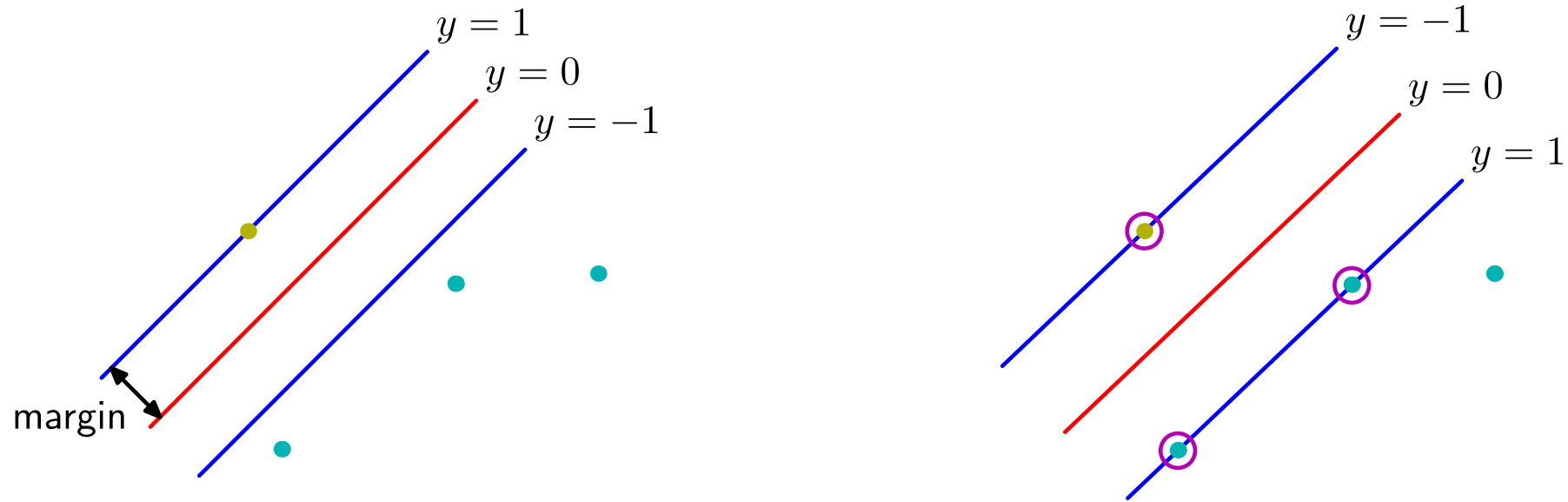


Figure 7.1 The margin is defined as the perpendicular distance between the decision boundary and the closest of the data points, as shown on the left figure. Maximizing the margin leads to a particular choice of decision boundary, as shown on the right. The location of this boundary is determined by a subset of the data points, known as support vectors, which are indicated by the circles.

Figure 7.1 of Pattern Recognition and Machine Learning.

Support Vector Machines

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

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

However, this problem is difficult to optimize directly.

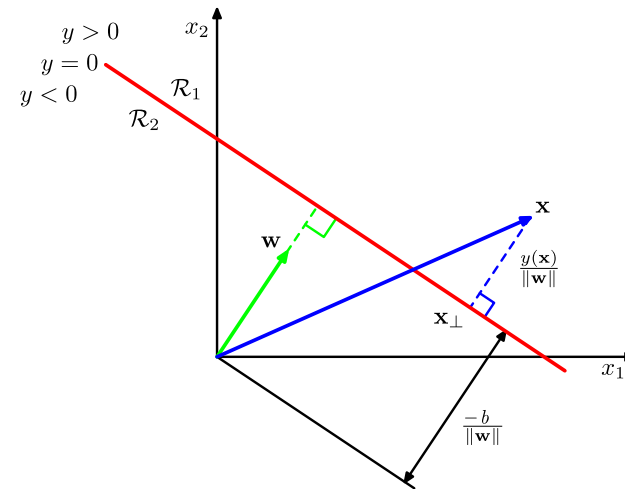


Figure 4.1 of Pattern Recognition and Machine Learning.

Because the model is invariant to multiplying \mathbf{w} and b by a constant, we can decide 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

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

to

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

Lagrange Multipliers – Inequality Constraints

Given a function $f(\mathbf{x})$, we can find a maximum with respect to a vector $\mathbf{x} \in \mathbb{R}^d$, by investigating the critical points $\nabla_{\mathbf{x}} f(\mathbf{x}) = 0$. We even know how to incorporate constraints of form $g(\mathbf{x}) = 0$. We now describe how to include inequality constraints $g(\mathbf{x}) \geq 0$.

We start by again forming a Lagrangian $f(\mathbf{x}) + \lambda g(\mathbf{x})$.

The optimum can either be attained for $g(\mathbf{x}) > 0$, when the constraint is said to be **inactive**, or for $g(\mathbf{x}) = 0$, when the constraint is said to be **active**. In the inactive case, the maximum is again a critical point of the Lagrangian with the condition $\lambda = 0$.

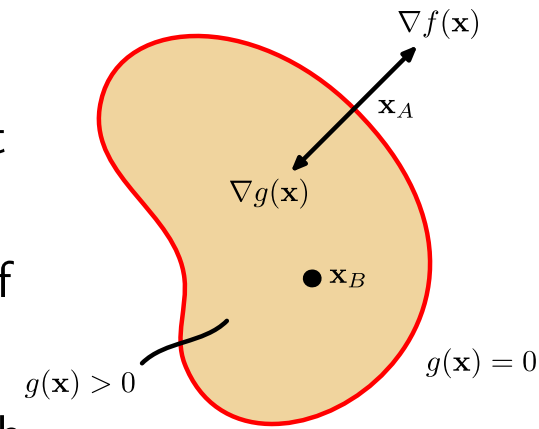


Figure E.3 of Pattern Recognition and Machine Learning.

When maximum is on a boundary, it corresponds to a critical point with $\lambda \neq 0$ – but note that this time the sign of the multiplier matters, because maximum is attained only when gradient of $f(\mathbf{x})$ is oriented away from the region $g(\mathbf{x}) \geq 0$. We therefore require $\nabla f(\mathbf{x}) = -\lambda \nabla g(\mathbf{x})$ for $\lambda > 0$.

In both cases, $\lambda g(\mathbf{x}) = 0$.

Karush-Kuhn-Tucker Conditions

Put together, every solution to a maximization problem of $f(\mathbf{x})$ subject to $g(\mathbf{x}) \geq 0$ must be a maximum of the Lagrangian with respect to \mathbf{x} and the following must hold:

$$\begin{aligned} g(\mathbf{x}) &\geq 0, \\ \lambda &\geq 0, \\ \lambda g(\mathbf{x}) &= 0. \end{aligned}$$

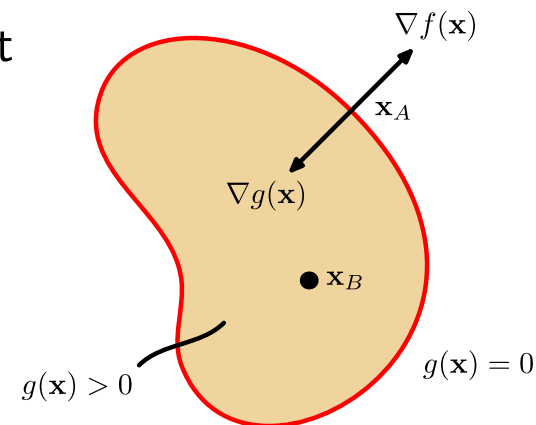


Figure E.3 of Pattern Recognition and Machine Learning.

If there exists a λ fulfilling these conditions, so does the λ **minimizing** the Lagrangian. Therefore, we maximize the Lagrangian with respect to \mathbf{x} , but minimize it with respect to λ .

Minimizing Given $f(\mathbf{x})$

If we instead want to find constrained minimum of $f(\mathbf{x})$, we can search for maximum of $-f(\mathbf{x})$, which results in *minimizing* the Lagrangian $f(\mathbf{x}) - \lambda g(\mathbf{x})$ with respect to \mathbf{x} and *maximizing* it for λ .

Necessary and Sufficient KKT Conditions

The KKT conditions are necessary conditions for a maximum (resp. minimum).

However, it can be proven that in the following settings, the conditions are also **sufficient**:

- if the objective to optimize is a *concave* function (resp. *convex* for minimization) with respect to \boldsymbol{x} ;
- the inequality constraints are continuously differentiable convex functions;
- the equality constraints are affine functions (linear functions with an offset).

It is easy to verify that these conditions hold for the SVM optimization problem.

In order to solve the constrained problem of

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

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 $a_i \geq 0$, $\sum_i a_i t_i = 0$ and a 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 \mathbf{x}_i is on a boundary, or $a_i = 0$. Given that the prediction for \mathbf{x} is $y(\mathbf{x}) = \sum_i a_i t_i K(\mathbf{x}, \mathbf{x}_i) + b$, we only need to keep the training points \mathbf{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.

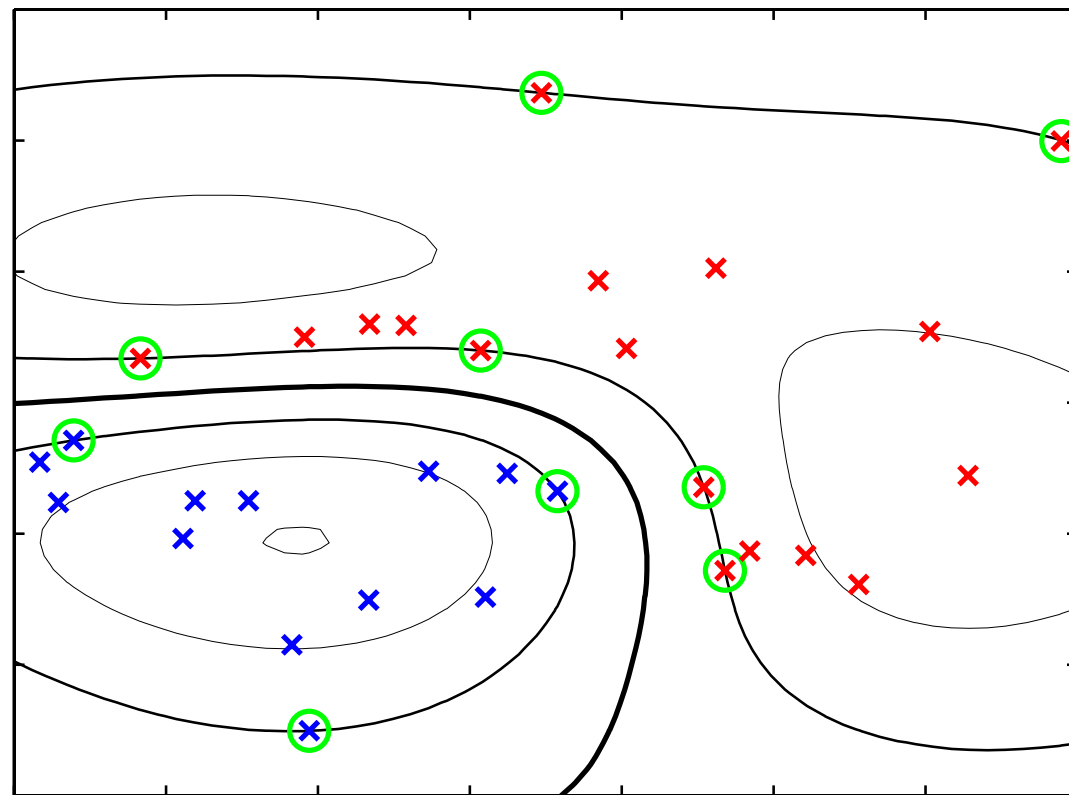


Figure 7.2 of Pattern Recognition and Machine Learning.