Mathematics of Support Vector Machines (SVM) Excerpts from Python Machine Learning Second Edition By Sebastian Raschka and Vahid Mirjalili[1] and other sources

Stepan Oskin

July 10, 2019

Abstract

Support Vector Machines (SVM) are a family of powerful and widely used learning algorithms, which can be considered an extension of the perceptron. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods exist to use SVM in a probabilistic classification setting). While in perceptron, the optimization objective is to minimize misclassification errors, in SVMs the learning objective is to maximize the margin. The margin is defined as the distance between the separating hyperplane (decision boundary) and the training samples that are closest to this hyperplane, which are the so-called support vectors. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

1 Classification

Classifying data is a common task in machine learning. Suppose some given data points each belong to one of two classes, and the goal is to decide which class a new data point will be in. In the case of support-vector machines, a data point is viewed as a p-dimensional vector (a list of p numbers), and we want to know whether we can separate such points with a $(p-1)$-dimensional hyperplane. This is called a linear classifier. There are many hyperplanes that might classify the data. One reasonable choice as the best hyperplane is the one that represents the largest separation, or margin, between the two classes. So we choose the hyperplane so that the distance from it to the nearest data point on each side is maximized. If such a hyperplane exists, it is known as the **maximum-margin hyperplane** and the linear classifier it defines is known as a **maximum-margin classifier**; or equivalently, the perceptron of optimal stability.

2 Maximum margin classification with support vector machines

Support Vector Machine (SVM) is a powerful and widely used learning algorithm, which can be considered an extension of the perceptron. For perceptron algorithm, optimization objective is to minimize misclassification errors. However, in SVMs the optimization objective is to **maximize the margin**. The margin is defined as the **distance between the separating hyperplane (decision boundary) and the training samples** that are closest to this hyperplane, which are the so-called **support vectors**. This is illustrated in figure 1:

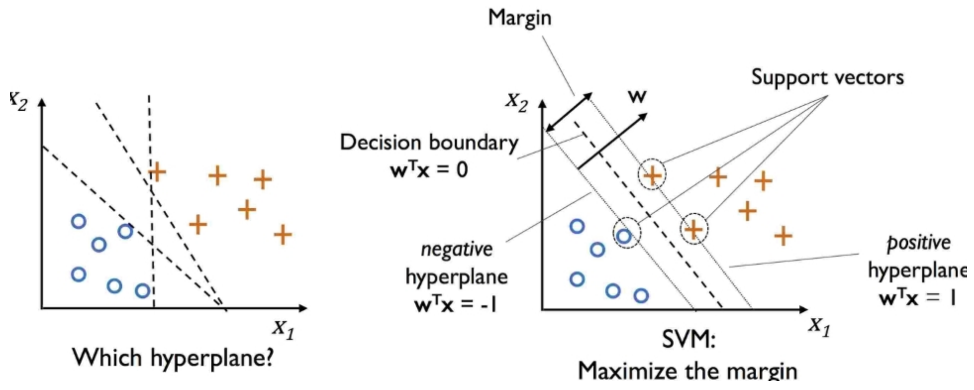


Figure 1: Optimization objective in Support Vector Machines is to maximize the margin between the training samples that are closest to the hyperplane (decision boundary).

An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

3 Maximum margin intuition

The rationale behind having decision boundaries with large margins is that they tend to have a lower generalization error whereas models with small margins are more prone to overfitting. To get an idea of the margin maximization, let's take a closer look at those positive and negative hyperplanes that are parallel to the decision boundary, which can be expressed as follows:

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}_{pos} = 1 \tag{1}$$

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}_{neg} = -1 \tag{2}$$

If we subtract those two linear equations 1 and 2 from each other, we get:

$$\implies \boldsymbol{w}^T (\boldsymbol{x}_{pos} - \boldsymbol{x}_{neg}) = 2 \tag{3}$$

We can normalize this equation (3) by the length of the vector $\boldsymbol{w}$, which is defined as follows:

$$\|\boldsymbol{w}\| = \sqrt{\sum_{j=1}^{m} w_j^2} \tag{4}$$

So we arrive at the following equation:

$$\frac{\boldsymbol{w}^T (\boldsymbol{x}_{pos} - \boldsymbol{x}_{neg})}{\|\boldsymbol{w}\|} = \frac{2}{\|\boldsymbol{w}\|} \tag{5}$$

The left side of the preceding equation can then be interpreted as the **distance between the positive and negative hyperplane**, which is the so-called **margin** that we want to maximize.

Now, the **objective function** of the SVM becomes the **maximization of this margin** by maximizing

$$\frac{2}{\|\boldsymbol{w}\|} \tag{6}$$

under the **constraint** that the samples are classified correctly, which can be written as:

$$\begin{gathered} w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} \geq 1 \text{ if } y^{(i)} = 1 \\ w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} \leq 1 \text{ if } y^{(i)} = -1 \\ \text{for } i = 1 \ldots N \end{gathered} \tag{7}$$

Here, **N** is the number of samples in our dataset.

These two equations basically say that **all negative samples should fall on one side of the negative hyperplane, whereas all the positive samples should fall behind the positive hyperplane**, which **can also be written more compactly** as follows:

$$y^{(i)} \left(w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} \right) \geq 1 \ \forall_i \tag{8}$$

In practice though, it is easier to minimize the **reciprocal term**:

$$\frac{1}{2} \|\boldsymbol{w}\|^2 \tag{9}$$

which can be solved by quadratic programming.

4 Dealing with a non-linearly separable case using slack variables

Although we don't want to dive much deeper into the more involved mathematical concepts behind the maximum-margin classification, let us briefly mention the slack variable ξ , which was introduced by Vladimir Vapnik in 1995 and led to the so-called **soft-margin classification**. The motivation for introducing the slack variable ξ was that the linear constraints need to be relaxed for non-linearly separable data to allow the convergence of the optimization in the presence of misclassifications, under appropriate cost penalization.

The positive-values slack variable is simply added to the linear constraints:

$$\begin{gathered} w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} \geq 1 - \xi^{(i)} \text{ if } y^{(i)} = 1 \\ w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} \leq -1 + \xi^{(i)} \text{ if } y^{(i)} = -1 \\ \text{for } i = 1 \ldots N \end{gathered} \tag{10}$$

Here, **N** is the number of samples in our dataset. So the new objective to be minimized (subject to the constraints) becomes:

$$\frac{1}{2}\|\boldsymbol{w}\|^2 + C\left(\sum_i \xi^{(i)}\right) \tag{11}$$

Via the variable `C`, we can then control the penalty for misclassification.

Large values of `C` correspond to large error penalties, whereas we are less strict about misclassification errors if we choose smaller values for `C`. We can then use the `C` parameter to control the width of the margin and therefore tune the bias-variance trade-off, as illustrated in the figure 2:

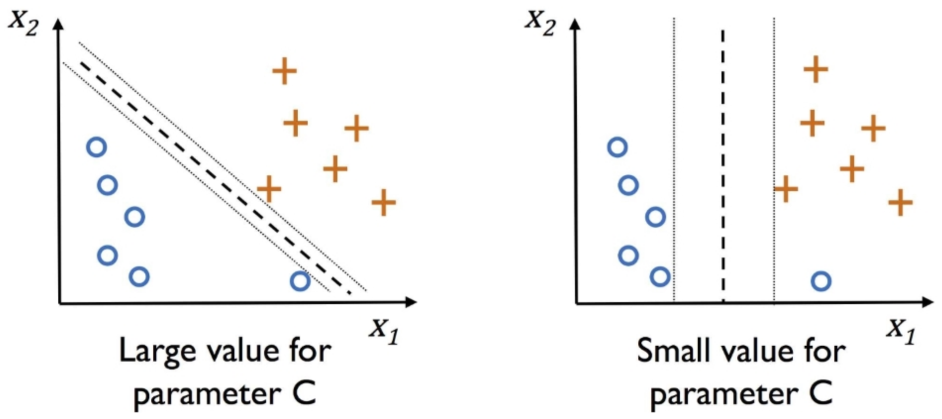


Figure 2: In Support Vector Machines, penalty for misclassification can be controlled by adjusting the values of parameter C. Large values of C correspond to large error penalties, whereas smaller values treat misclassifications less strictly.

This concept is related to regularization, which is discussed in the document covering Logistic Regression, in the context of regularized regression where decreasing the value of `C` increases the bias and lowers the variance of the model.

The three decision regions of the SVM, visualized after training the classifier on the Iris flower dataset, are presented on figure 3.

5 Logistic regression versus support vector machines

In practical classification tasks, linear logistic regression and linear SVMs often yield very similar results. Logistic regression tries to maximize the

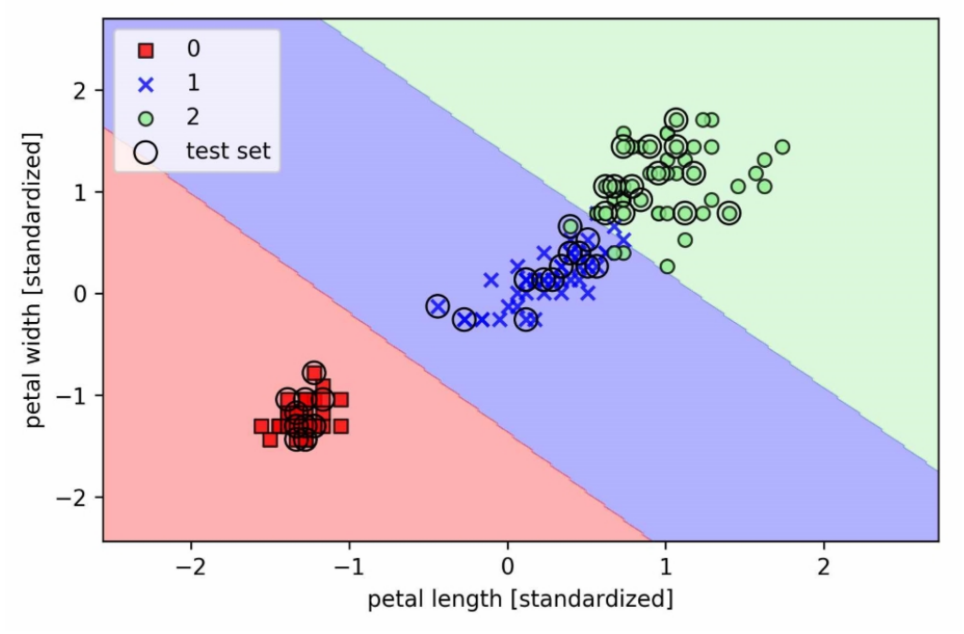


Figure 3: The three decision regions of the linear SVM after training the classifier on the Iris flower dataset. Results of SVM are similar to those yielded by a Logistic Regression model.

conditional likelihoods of the training data, which makes it more prone to outliers than SVMs, which mostly care about the points that are closest to the decision boundary (support vectors). On the other hand, logistic regression has the advantage that it is a simpler model and can be implemented more easily. Furthermore, logistic regression models can be easily updated, which is attractive when working with streaming data.

6 SVM implementations in `scikit-learn`

For practical purposes, it is better to use `scikit-learn`'s more optimized implementation of SVM that also supports multi-class settings off the shelf (OvR by default). The `scikit-learn` library's `Perceptron` and `LogisticRegression` classes make use of the *LIBLINEAR* library, which is a highly optimized C/C++ library developed at the National Taiwan University[2]. Similarly, the `SVC` class that we used to train an SVM makes use of *LIBSVM* library, which is an equivalent C/C++ library specialized for SVMs[3].

The advantage of using LIBLINEAR and LIBSVM over native Python implementations is that they allow the extremely quick training of large amounts of linear classifiers. However, sometimes our datasets are too large to fit into computer memory. Thus, `scikit-learn` also offers alternative implementations via the `SGDClassifier` class, which also supports online learning via the `partial_fit` method. The concept behind the `SGDClassifier` class is similar to the stochastic gradient algorithm that we implemented for Adaline.

7 Kernel methods for linearly inseparable data

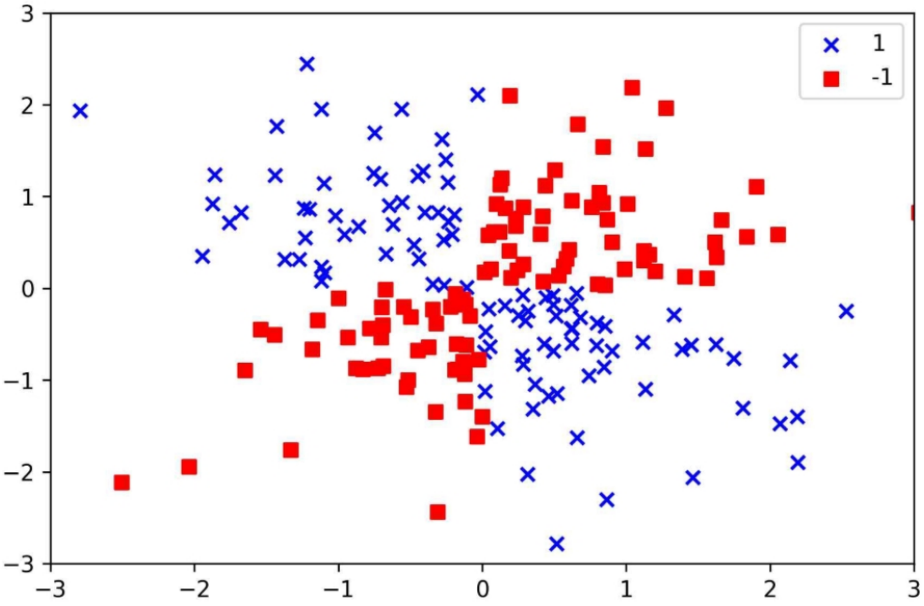


Figure 4: Sample dataset generated in a form of a XOR gate, where 100 samples are assigned the class label 1, and 100 samples are assigned class label -1. In this dataset, samples from positive and negative classes cannot be separated well using a linear hyperplane as a decision boundary.

Another reason why SVMs enjoy high popularity among machine learning practitioners is that it can be easily **kernelized** to solve nonlinear classification problems. On figure 4 we can see an example of a nonlinear classification problem. This dataset was randomly generated using `NumPy` library in Python; it consists of 100 samples of class 1 and 100 samples of class -1, and has a form of a XOR gate.

Obviously, we would not be able to separate samples from the positive and negative class very well using a linear hyperplane as a decision boundary via the linear logistic regression or linear SVM model that we discussed in earlier sections.

8 The basic idea behind kernel methods

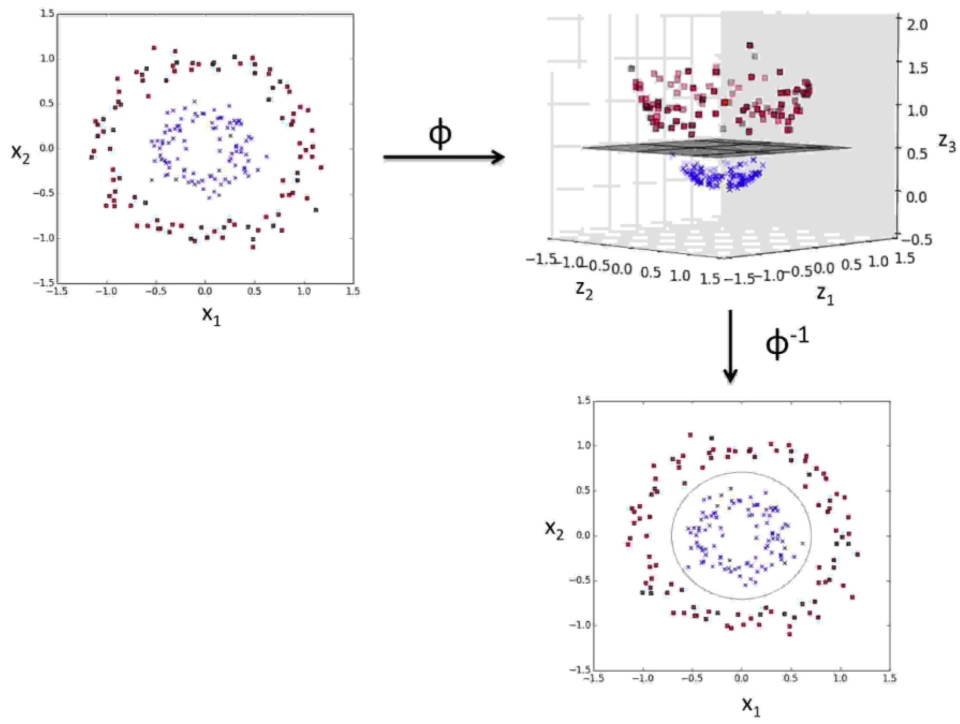


Figure 5: An example of a linearly inseparable two-dimensional dataset being transformed via a mapping function ϕ onto a three-dimensional space, where it becomes linearly separable, and then re-projected back to the two-dimensional space. This transformation represents the basic idea behind kernel methods.

The basic idea behind **kernel methods** to deal with such linearly inseparable data is to create nonlinear combinations of the original features to project them onto a higher-dimensional space via a mapping function ϕ

where it becomes linearly separable. As shown in figure 5, we can transform a two-dimensional dataset onto a new three-dimensional feature space where the classes become separable via the following projection:

$$\phi(x_1, x_2) = (z_1, z_2, z_3) = (x_1, x_2, x_1^2 + x_2^2) \tag{12}$$

This allows us to separate the two classes shown in the plot via a linear hyperplane that becomes a nonlinear decision boundary if we project it back onto the original feature space:

9 Using the kernel trick to find separating hyperplanes in high-dimensional space

To solve a nonlinear problem using an SVM, we would transform the training data onto a higher-dimensional feature space via a mapping function ϕ and train a linear SVM model to classify the data in this new feature space. Then, we can use the same mapping function to transform new, unseen data to classify it using the linear SVM model.

However, one problem with this mapping approach is that the construction of the new features is computationally very expensive, especially if we are dealing with high-dimensional data. This is where the so-called **kernel trick** comes into play. Although we didn't go into much detail about how to solve the quadratic programming task to train an SVM, in practice all we need is to replace the dot product $\boldsymbol{x}^{(i)T}\boldsymbol{x}^{(j)}$ by $\phi\left(\boldsymbol{x}^{(i)}\right)^T \phi\left(\boldsymbol{x}^{(j)}\right)$. In order to save the expensive step of calculating this dot product between two points explicitly, we define a so-called **kernel function**:

$$\kappa\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right) = \phi\left(\boldsymbol{x}^{(i)}\right)^T \phi\left(\boldsymbol{x}^{(j)}\right) \tag{13}$$

One of the most widely used kernels is the **Radial Basis Function (RBF)** kernel or simply called the **Gaussian kernel**:

$$\kappa(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \exp\left(-\frac{\left\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\right\|^2}{2\sigma^2}\right) \tag{14}$$

This is often simplified to:

$$\kappa(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \exp\left(-\gamma \left\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\right\|^2\right) \tag{15}$$

Here, $\gamma = \frac{1}{2\sigma^2}$ is a free parameter that is to be optimized.

Roughly speaking, the term **kernel** can be interpreted as a **similarity function** between a pair of samples. The **minus sign inverts the distance measure into a similarity score**, and, due to the exponential term, the resulting similarity score will fall into a range between 1 (for exactly similar samples) and 0 (for very dissimilar samples).

Now that we defined the big picture behind the kernel trick, let us see if we can train a kernel SVM that is able to draw a nonlinear decision boundary that separates the XOR data well. Here, we simply use the `SVC` class from `scikit-learn` that and replace the `kernel='linear'` parameter with `‘kernel='rbf'`. As we can see in the resulting plot on figure 6, the kernel SVM separates the XOR data relatively well:

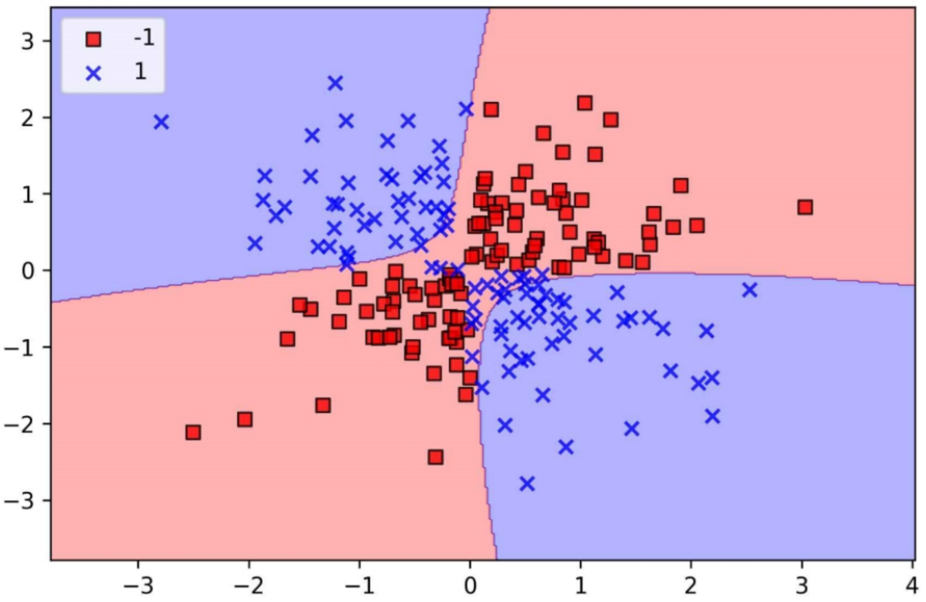
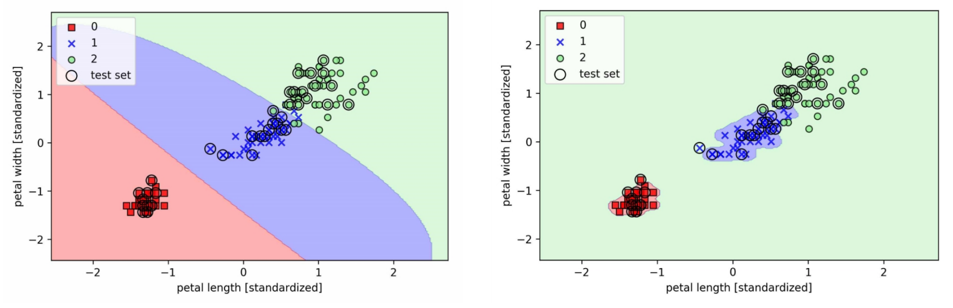


Figure 6: Nonlinear decision boundary drawn by a kernel SVM is separating the XOR data relatively well. Cut-off parameter γ was set to 0.1.

10 Cut-off parameter γ for the Gaussian sphere

The γ parameter, which was set to `gamma=0.1` in the example with XOR data, can be understood as a **cut-off** parameter for the Gaussian sphere. If we increase the value for γ, we increase the influence or reach of the training samples, which leads to a tighter and bumpier decision boundary. To get a better intuition for γ, below on figure 7 we present the two different values of γ (0.2, and 100) applied to an RBF kernel SVM that was used to classify Iris flower dataset:



(a) $\gamma = 0.2$ (b) $\gamma = 100$

Figure 7: Different values of parameter γ applied to an RBF kernel SVM that was used to classify Iris flower dataset. Large value of γ (7b) results in the model fitting the training dataset very well, but will likely have a high generalization error on unseen data. This illustrates that the γ parameter also plays an important role in controlling overfitting.

If we choose a relatively small value for γ, the resulting decision boundary of the RBF kernel SVM model will be relatively soft, as shown in figure 7a. When using a relatively large value of γ, the decision boundary around the classes 0 and 1 is much tighter, as can be seen in figure 7b. Although the model fits the training dataset very well, such a classifier will likely result in a high generalization error on unseen data. This illustrates that the γ parameter also plays an important role in controlling overfitting.

11 More information on SVM

More detailed description of support vector machines can be found in *The Nature of Statistical Learning Theory* book by Vladimir Vapnik[4], *A Practical Guide to Support Vector Classification* tutorial from National Taiwan University[5], or Chris J.C. Burges' excellent explanation in *A Tutorial on Support Vector Machines for Pattern Recognition* [6].

References

[1] S. Raschka and V. Mirjalili, *Python Machine Learning, 2nd Ed.* . Birmingham, UK: Packt Publishing, 2 ed., 2017.

[2] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, " LIBLINEAR: A Library for Large Linear Classification ," tech. rep., 2008.

[3] C.-C. Chang and C.-J. Lin, " LIBSVM: A Library for Support Vector Machines ," tech. rep., National Taiwan University, Taipei, 2001.

[4] V. N. Vapnik, *The Nature of Statistical Learning Theory* . New York, NY: Springer, New York, NY, second edi ed., 2000.

[5] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, " A Practical Guide to Support Vector Classification ," tech. rep., National Taiwan University, Taipei, 2016.

[6] C. J. C. Burges, " A Tutorial on Support Vector Machines for Pattern Recognition ," tech. rep., Bell Laboratories, Lucent Technologies, Boston, MA, 1998.