Support Vector Machines

Aarav Khanna*

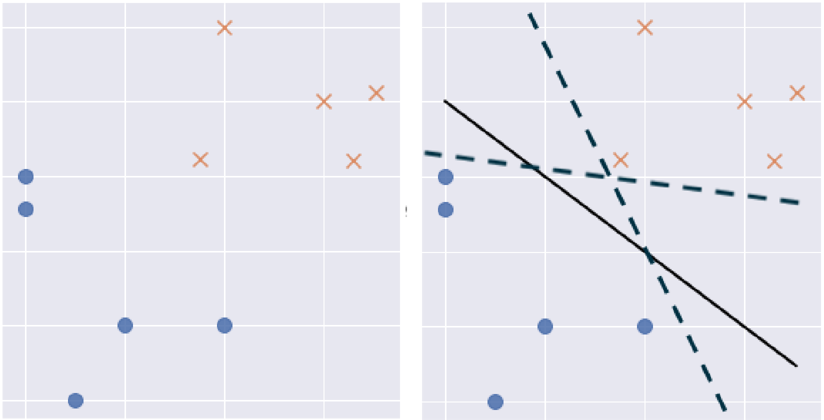
October 2020

1 Introduction

Support vector machines (SVMs), in their most basic form, are supervised learning models that solve binary linear classification problems. They are commonly modified to separate multiple classes, classify non-linearly separable data, or perform regression analysis.

2 The Basics

A binary linear classification problem is one where, given that we have two classes of data, we can draw a line to separate the data into the proper classes.



But as shown above, there are multiple ways to separate the given data into two classes. How do we pick the best one? Intuitively, the solid line is a better separator of these data than either dashed line. More rigorously, we can say that the solid separator line, or decision boundary, is best because it maximizes

*based on Sylesh Suresh and Kevin Fu's lectures of the same name the distance, or margin, from the nearest data points. In an SVM, each point is treated as a vector, hence we call the closest data points support vectors.

We want to maximize the margin because our goal is to use the decision boundary to predict where unclassified data points will fall. If the decision boundary is too close to either group, our model will perform poorly when given new points. By maximizing the margin, we minimize the potential error from generalizing.

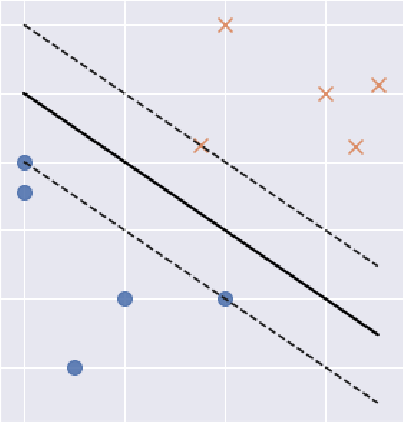


Figure 1: The optimal decision boundary and its margins, which run through the support vectors.

When we add input features to an SVM, the decision boundary will necessarily have to increase in dimension with it. This is why the margins are referred to as positive and negative hyperplanes. Note that the dimension of a hyperplane is one less than the dimension of the feature space.

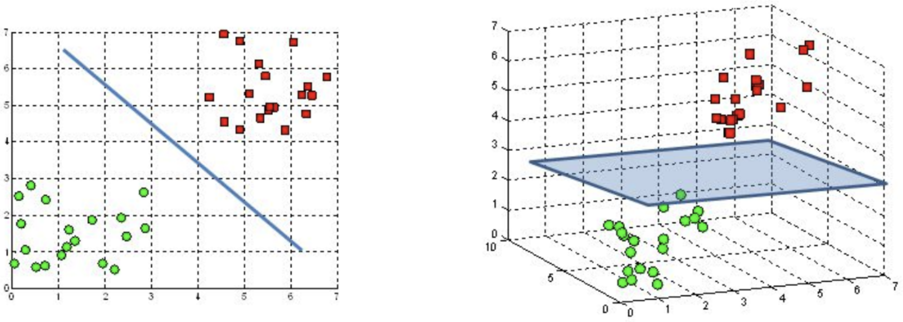


Figure 2: A hyperplane in 2D is a line; a hyperplane in 3D is a plane.

To mathematically find the maximum margin, we must define the hyperplanes as equations. We'll start by defining components: $\vec{w}$ is a vector normal to the hyperplanes and $\vec{x}$ is the vector form of a point x. Thus, the decision boundary can be written as

$$\vec{w} \cdot \vec{x} + b = 0$$

The equation equals zero because when a point is on the decision boundary, it's unclear which class it belongs to. You'll notice this is similar to the familiar slope-intercept form of a line, $y = mx + b$, but with a dot product since we're working with vectors. Continuing, the positive hyperplane, or "upper" margin, is

$$\vec{w} \cdot \vec{x}_{pos} + b = 1 \tag{1}$$

and the negative hyperplane is

$$\vec{w} \cdot \vec{x}_{neg} + b = -1 \tag{2}$$

These two equations are ±1 because points on and beyond these hyperplanes can be classified to either the "positive" or "negative" class.

The margin can be defined as a vector too; its magnitude is the distance between two support vectors, $\vec{x}_{pos} - \vec{x}_{neg}$, and its direction is the same as the normal vector $\vec{w}$. Using this knowledge, by multiplying the unit vector $\frac{\vec{w}}{|\vec{w}|}$ by the margin vector's magnitude, we can define the margin as

$$\frac{\vec{w} \cdot (\vec{x}_{pos} - \vec{x}_{neg})}{|\vec{w}|} \tag{3}$$

Our goal is to mathematically maximize the margin, but this equation has too many variables to maximize easily. We can subtract (2) from (1) to get

$$\vec{w} \cdot (\vec{x}_{pos} - \vec{x}_{neg}) = 2$$

and then apply it to (3) to reach a simple equation for the margin:

$$\frac{\vec{w} \cdot (\vec{x}_{pos} - \vec{x}_{neg})}{|\vec{w}|} = \frac{2}{|\vec{w}|} \tag{4}$$

To maximize the margin, we must maximize (4). Assuming our data are correctly classified, that is, for every input x_i the output y_i is accurate, or

$$y_i(\vec{w} \cdot \vec{x} + b) - 1 = 0$$

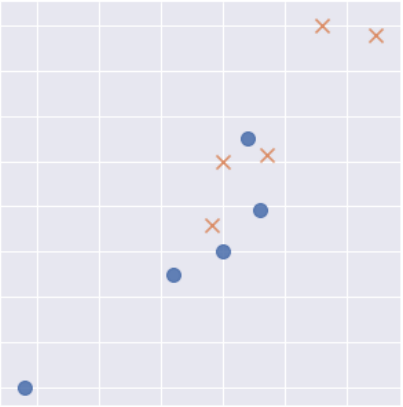
then we can treat maximizing (4) as equivalent to minimizing the equation

$$\frac{1}{2}|\vec{w}|^2 \tag{5}$$

This is the loss function of a basic SVM, and is minimizable using Lagrange multipliers. The details on how that works are beyond the scope of this lecture (since SVMs can deal with more than 2 dimensions, the solution requires multivariable calculus), but you're encouraged to look it up.

3 Soft-Margin Classification

In the real world, most data sets won't be perfectly linearly separable like the data shown in Figure 3, and thus the margin can't be minimized by equation (5). For example, there's no good way to draw a straight line to separate these points:



In cases like this, where the data aren't linearly separable but can almost be separated by a linear boundary, we add a slack variable ξ to our loss function.

$$\frac{1}{2}|\vec{w}|^2 + C\sum_i \xi_i \tag{6}$$

The idea is to punish the model for misclassifying points, while still allowing it to do so. By putting ξ into our loss function, the model will try to minimize the severity and number of misclassified points. The corresponding constraints become

$$y_i(\vec{w} \cdot \vec{x_i} + b) \geq 1 - \xi_i$$

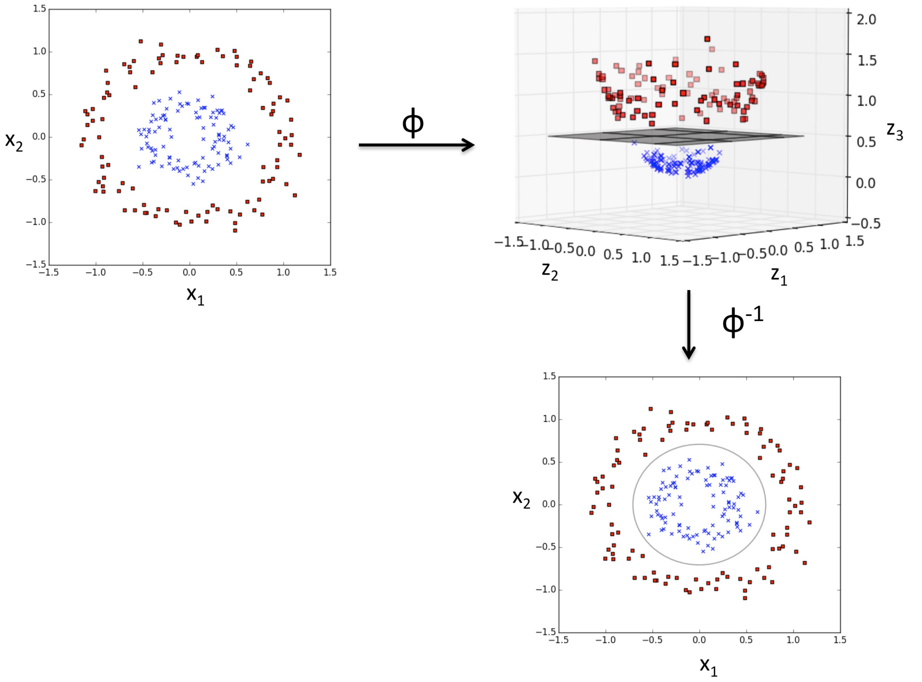
C here is a hyperparameter that controls the cost for each misclassified point. Higher values of C mean the model is punished more for wrong points, while lower values mean it is punished less. It follows that extremely high values of C will lead to overfitting, while extremely low values create a model that's not very predictive. (Note too that our original loss equation is the same as the soft-margin equation in (6), but with $C = \infty$.)

4 Nonlinear Classification

However, some real world datasets are neither linearly separable nor easily approximated by a linear decision boundary. In this case, even using a soft-margin SVM will fail to be predictive. We can work around this by using a mapping function $\phi(\cdot)$ to project our data to a higher dimension, finding a hyperplane, then returning to the original dimension. For example, this function

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

projects our 2D data to 3D space, allowing our SVM to draw a flat hyperplane, then flattens it back to 2D. The end result is a circular decision boundary:



In this case, we need to project the data because the SVM loss function is based on a linear decision boundary—there's no way to apply it to a circular boundary.

As you can imagine, this projection function and the subsequent dot products done on the projected points are computationally expensive. To get around this, we can use the kernel trick. Kernels are functions that allow us to calculate a dot product of higher-dimensional points, without actually doing any projection. In other words, functions of this form exist:

$$K(\vec{x_i}, \vec{x_j}) = \phi(\vec{x_i}) \cdot \phi(\vec{x_j})$$

where we only need the inputs $\vec{x_i}$ and $\vec{x_j}$ to calculate the dot product $\phi(\vec{x_i}) \cdot \phi(\vec{x_j})$. This is convenient because for an SVM, we don't care about the projected points, just the dot products of them.

One of the most popular kernel functions is the Radial Basis Function kernel (RBF kernel) or Gaussian kernel:

$$k(\vec{x_i}, \vec{x_j}) = \exp\left(-\gamma ||\vec{x_i} - \vec{x_j}||^2\right)$$

γ is a free parameter that can be optimized. How kernel functions work mathematically is beyond the scope of this lecture, but again, you can find the details online.

5 Non-Binary Classification

Though SVMs are designed for binary data, they can be extended to multiple classes. The two common ways of doing this are the one-vs-rest and one-vs-one (OvR and OvO) approaches.

OvR attempts to separate one class at a time from the rest of data, treating it as a series of binary classification problems, and so will create n models from n classes. This is computationally simple, but risks oversimplification. Meanwhile, OvO makes a model to separate every class from every other class, creating $\frac{n(n-1)}{2}$ models from n classes. This formula is derived from the summation formula

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

but with the bounds adjusted to 0 and $n-1$.

In practice, OvR and OvO give similar results, so due to OvO's $O(n^2)$ complexity, OvR is more prevalent.

6 Competitions

- Surprise! The Random Forests competition is up at `https://tjmachinelearning.com/competitions/1920`. It closes at 11:59 PM on Tuesday, October 29 and will count towards your ranking.
- The SVM competition is also up at `https://tjmachinelearning.com/competitions/1920` and ends at 11:59 PM on Tuesday, November 5 (two weeks from today). To make things easier, you should use scikit-learn, a package that has multi-class SVMs and kernel functions built-in. You can try it yourself now if you're feeling ambitious, but we'll give a more in-depth explanation next week.

7 References

7.1 Images

- Fig. 2: `shorturl.at/aiEQR`

- Projection visualization (from 2018-19 SVM lecture): `shorturl.at/FJPQ5`
- All other images (from my SVM demo): `shorturl.at/jmoNP`

7.2 More Resources

- sk-learn docs: `https://scikit-learn.org/stable/modules/svm.html`
- Simplified explanation: `https://youtu.be/N1vOgolbjSc`
- Slack variable explanation: `shorturl.at/sAQ23`