Support Vector Machines

Jacob Katzeff

June 6, 2018

1 Abstract

This paper provides an overview of a data classification method in the realm of Machine Learning, called Support Vector Machines (SVMs for short). SVMs provide a convenient way of learning data patterns from linearly separable binary data. In viewing SVMs as a constraint optimization problem, an efficient algorithm for predicting the output for new input data is discovered. By relaxing constraints, soft margins provide flexibility for nearly-linearly separable data. By changing the inner product associated with the separating hyperplane, non-linear patterns can also be learned. By adjusting the prediction model, SVMs can also be used to predict non-binary data.

2 Machine Learning Basics and Notation Used

2.1 Introduction to Machine Learning

Let $\mathbf{X} \in \mathbb{R}^d$ for some chosen $d \in \mathbb{N}$, and let $y \in \{-1, 1\}$. $\mathbf{X}$ is called the *input* and y is called the *label*. The ordered pair $(\mathbf{X}, y)$ is called a **training example**. In the rest of this paper, we assume there are m existing training examples, which are labeled $(\mathbf{X}_i, y_i)$, $i = 1...m$, and that $\mathbf{X}$ without subscript will refer to an **unlabeled input**: a vector without an associated $y \in \{-1, 1\}$

We would like to "learn" a **hypothesis** $h : \mathbb{R}^d \to \{-1, 1\}$ which takes in an unlabeled input $\mathbf{X}$, and outputs a prediction for the output y, based on our existing training examples. In particular, we would like to find a hypothesis which accurately labels input data that has not been seen before.

2.2 Notation

Training examples with a label of +1 are called **positive examples**, while training examples with a label of -1 are called **negative examples**. +1 and -1 may also be called the positive or negative **classes**.

When discussing optimization problems, the function maximized or minimized is called the **objective** functions and the functions that must hold are called the **constraints**.

Bolded words are used for important terms introduced for the first time. Bolded variables are used for vectors.

3 SVMs as an Optimization problem

3.1 Motivation

A reasonable way of predicting unlabeled inputs is by using a $d-1$ dimensional hyperplane that separates the (d-dimensional) data into their two labels. For example, in two dimensions, if we could find a line that separates the data according to their classes, we could predict unlabeled inputs by which side of the line they lie on. [1]

Thus, we need to learn $\mathbf{w}, b$ (also called the **weight** and **bias** parameters) associated with the general equation of the hyperplane, $\mathbf{w}^T\mathbf{x} + b$. Once we can do that, we need only check if the result for $\mathbf{x} = \mathbf{X}$ is positive or negative in order to predict if $\mathbf{X}$ is a positive or negative example.

More formally, our hypothesis h would look like $h(\mathbf{X}) = \text{sgn}(\mathbf{w} \cdot \mathbf{X} + b)$ $= \text{sgn}(\mathbf{w}^T\mathbf{X} + b)$, where sgn indicates the sgn function:

$$\text{sgn}(x) = \begin{cases} -1 & x < 0 \\ 1 & x \geq 0. \end{cases}$$

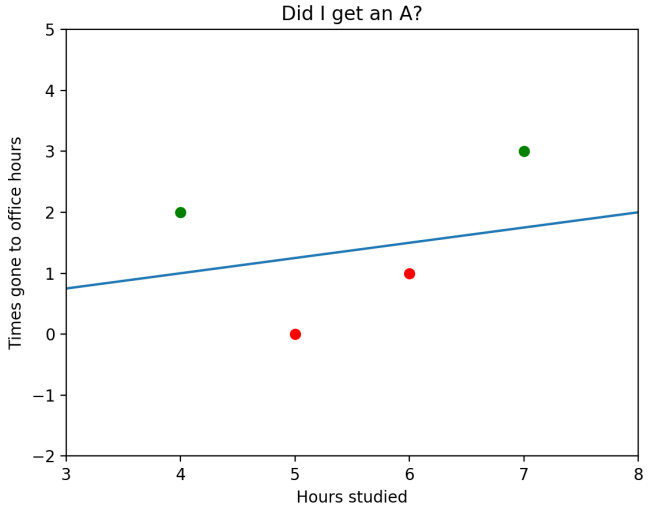
This seems reasonable, but there are (infinitely!) many different hyperplanes that can separate a list of data which is linearly separable. How can we choose the best one? There are many different ways, but it seems reasonable that we would like to choose the hyperplane such that the closest data points to the hyperplane are as far away as possible.

[1] For now, we will assume that this is possible, though this may not always be the case. Later on, methods for dealing with the non-linearly separable case will be discussed.

In more concrete terms, our optimal weights and biases $\mathbf{w}^*$ and b^* would be:

$$\mathbf{w}^*, b^* = \underset{\mathbf{w},b}{\operatorname{argmax}} \min_{\mathbf{X} \in \mathbf{X}_i} \mathbf{w}^T \mathbf{X} + b$$

But, again, since there are infinitely many separating hyperplanes, it would be impossible to try all $\mathbf{w}, b$ in order to find the best weights. There must be a more intelligent way of finding the optimal weight and bias.



Above: A hyperplane separating a two-dimensional dataset.

3.2 Functional and Geometric Margins

We want to narrow down what it means to be "good" weights and biases. To do so, we introduce the concept of the **functional margin**

$$\hat{\gamma}_i = y_i(\mathbf{w}^T \mathbf{X}_i + b) \tag{1}$$

for each training example. Observe that for a given training example, if the label is positive, then $\hat{\gamma}_i$ is large if $\mathbf{w}^T \mathbf{X}_i + b$ is a large positive number, and that if the label is negative, $\hat{\gamma}_i$ is large if $\mathbf{w}^T \mathbf{X}_i + b$ is a large negative number.

It follows that large values of $\hat{\gamma}_i$ represent high confidence in the prediction of the label, since it is far from the hyperplane, and on the correct side. Similarly, training examples where we are least confident are the ones where $\hat{\gamma}_i$ are very small. The functional margin therefore measures confidence in a prediction.

In order to measure the accuracy of the model as a whole, we can take the worst functional margin of the training examples: $\hat{\gamma} = \min_i \hat{\gamma_i}$.

However, there exists a major problem with the idea of the functional margins. By rescaling $\mathbf{w}$ and b, we can make the functional margin very small or very large, while retaining the same predictions. In other words, the magnitude of the functional margin makes no difference in our prediction; we could double $\mathbf{w}$ and b and get the same predictions, for example.

A better measure of the confidence in our hypothesis is the **geometric margin**

$$\gamma_i = y_i((\frac{\mathbf{w}}{||\mathbf{w}||})^T\mathbf{X}_i + \frac{b}{||\mathbf{w}||}) \tag{2}$$

in which the weight vector is normalized to magnitude 1. Similarly to above, we can measure the overall hypothesis by taking the minimum of the geometric margins

$$\gamma = \min_i \gamma_i.$$

We can relate the geometric and functional margins by the equation

$$\gamma = \frac{\hat{\gamma}}{||\mathbf{w}||}. \tag{3}$$

3.3 Constraint Optimization

With the geometric margin as a way of measuring how "good" a set of weights and biases are, it seems natural to try and maximize it. This leads to the optimization problem

$$\begin{aligned} \underset{\gamma, \mathbf{w}, b}{\text{maximize}} \quad & \gamma \\ \text{subject to} \quad & y_i(\mathbf{w}^T\mathbf{X}_i + b) \geq \gamma \text{ for } \; i = 0, \ldots, m, \\ & ||\mathbf{w}|| = 1. \end{aligned}$$

However, this optimization problem is hard:[2] $||\mathbf{w}|| = 1$ is a highly non-convex constraint, and as such, is not easily solvable with use of computer software. By equation 3, we can relate the geometric and functional margins:

[2]Computers are able to solve convex constraint problems relatively easily. In particular, if the objective function is linear and all constraints are linear, it can be solved using **linear programming**. If the objective function is quadratic and all constraints are linear, it can be solved using **quadratic programming**.

$$\begin{aligned} \underset{\gamma, \mathbf{w}, b}{\text{maximize}} \quad & \gamma = \frac{\hat{\gamma}_i}{||\mathbf{w}||} \\ \text{subject to} \quad & y_i(\mathbf{w}^T \mathbf{X}_i + b) \geq \hat{\gamma}_i \text{ for } \; i = 0, \ldots, m. \end{aligned}$$

The same problem arises, except this time as the objective: $\frac{\hat{\gamma}_i}{||\mathbf{w}||}$ is not convex. However, we can set $\hat{\gamma}_i = 1$, just by rescaling $\mathbf{w}$ and b, as noted above. This allows us to maximize over $\frac{1}{||\mathbf{w}||}$ instead of $\frac{\hat{\gamma}_i}{||\mathbf{w}||}$. Another observation is that maximizing $\frac{1}{||\mathbf{w}||}$ is the same as minimizing $||\mathbf{w}||^2$, which leads to the following optimization problem:[3]

$$\begin{aligned} \underset{\gamma, \mathbf{w}, b}{\text{minimize}} \quad & \frac{1}{2}||\mathbf{w}||^2 \\ \text{subject to} \quad & 1 - y_i(\mathbf{w}^T \mathbf{X}_i + b) \leq 0 \text{ for } \; i = 0, \ldots, m. \end{aligned} \tag{4}$$

This is a quadratic objective with only linear constraints, which is convex, and solvable using available software called quadratic programming.

4 Functional Optimization in the General Case

4.1 The Primal Problem

In order to gain more insight on the structure of the optimization problem arrived at in the previous subsection, we now look at the general optimization problem:

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{w}) \\ \text{subject to} \quad & g_i(\mathbf{w}) \leq 0 \text{ for } \; i = 0, \ldots, k, \\ & h_i(\mathbf{w}) = 0 \text{ for } \; i = 0, \ldots, n. \end{aligned} \tag{5}$$

Observe that this optimization problem (called the **primal** problem) is an extension of the optimization problem we are concerned with; we just need the inequality constraint. Nonetheless, we will look into this generalized formulation.

We define the Lagrangian, in terms of the primal problem functions, as

[3] The constant of $\frac{1}{2}$ is frequently included for convenience when computing gradients.

$$\mathcal{L}(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{w}) + \sum_{i=1}^{k} \alpha_i g_i(\mathbf{w}) + \sum_{i=1}^{n} \boldsymbol{\beta}_i h_i(\mathbf{w}). \tag{6}$$

And define also:

$$\theta_{primal}(\mathbf{w}) = \max_{\boldsymbol{\alpha}, \boldsymbol{\beta} \text{ s.t. } \boldsymbol{\alpha}_i \geq 0} \mathcal{L}(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}).$$

Choose a vector $\mathbf{w}$ and consider $\theta_{primal}(\mathbf{w})$. If $\mathbf{w}$ doesn't satisfy the constraints in equation 5, then it is clear that we can set the Lagrangian in equation 6 as large as possible by choosing respective α_i or β_i. In other words:

$$\theta_{primal}(\mathbf{w}) = \begin{cases} f(\mathbf{w}) & \text{if } \mathbf{w} \text{ satisfies primal constraints} \\ \infty & \text{otherwise.} \end{cases}$$

It is now clear to see that minimizing $\theta_{primal}(\mathbf{w})$ is the same as minimizing $f(\mathbf{w})$. We will call the solution to this problem p^*.

4.2 The Dual Problem

In the previous subsection, we discussed maximizing the Lagrangian over all possible $\boldsymbol{\alpha}, \boldsymbol{\beta}$. Now, let's discuss minimizing the Lagrangian over all possible $\mathbf{w}$, in order to get further insight into the nature of the (general) optimization problem at hand. We will call this the **Dual Problem**:

$$\theta_{dual}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \min_{\mathbf{w}} \mathcal{L}(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}).$$

In the Primal problem, we minimized the maximum of the Lagrangian. Now, let's maximize the minimum of the Lagrangian, instead:

$$\max_{\boldsymbol{\alpha}, \boldsymbol{\beta} \text{ s.t. } \boldsymbol{\alpha}_i \geq 0} \theta_{dual}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \max_{\boldsymbol{\alpha}, \boldsymbol{\beta} \text{ s.t. } \boldsymbol{\alpha}_i \geq 0} \min_{\mathbf{w}} \mathcal{L}(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}).$$

Define the solution to this optimization to be d^*. It's simple to show $d^* \leq p^*$ because the "maximin" function is always less than or equal to the "minimax" function, but in some circumstances $d^* = p^*$. When this is the case, we can elect to solve the dual problem in lieu of the primal problem.

Sufficient conditions in which $d^* = p^*$ include: f and g_i convex (i.e., Hessian Matrix is positive semi-definite), $h_i(\mathbf{w}) = \mathbf{a}_i^T \mathbf{w} + b$ for some $\mathbf{a}_i$ and b.

If these conditions hold, then there exists $\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*$, such that $\mathbf{w}^*$ solves the primal problem and $\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*$ solves the dual problem, and $p^* = d^* = \mathcal{L}(\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$. Additionally, $\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*$ follow the **Karush-Kuhn-Tucker** (KKT) conditions:[4]

$$\frac{\partial}{\partial \mathbf{w}_i}\mathcal{L}(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = 0 \qquad i = 1 \dots m \tag{7}$$

$$\frac{\partial}{\partial \boldsymbol{\beta}_i}\mathcal{L}(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = 0 \qquad i = 1 \dots n \tag{8}$$

$$\alpha_i^* g_i(\mathbf{w}^*) = 0 \qquad i = 1 \dots k \tag{9}$$

$$g_i(\mathbf{w}^*) \leq 0 \qquad i = 1 \dots k \tag{10}$$

$$\alpha_i^* \geq 0 \qquad i = 1 \dots k. \tag{11}$$

Note that by equation 9, we have that either $\alpha_i^* = 0$ or $g_i(\mathbf{w}^*) = 0$. In particular, if $\alpha_i^* > 0$, then $g_i(\mathbf{w}) = 0$.

4.3 Applying the KKT Conditions to SVMs

Recall our optimization problem from equation 4.

It is easily be seen that $\frac{1}{2}||\mathbf{w}||^2$ is convex as well as $1 - y_i(\mathbf{w}^T\mathbf{X}_i + b)$. By our observation from equation 9, we see that if $\alpha_i^* > 0$, then the functional margin (see equation 1), $y_i(\mathbf{w}^T\mathbf{X}_i + b) = 1$. Since the functional margin, by our constraint, must be greater than or equal to 1, these training examples are the ones with the smallest functional margin. In other words, the training examples with $\alpha_i > 0$ are only the ones which are closest to the separating hyperplane. Such training examples are called **Support Vectors**. It is worth noting that the number of Support Vectors are usually much, much smaller than the number of total vectors.

The Lagrangian of our SVM optimization problem is:

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{m} \alpha_i (y_i(\mathbf{w}^T\mathbf{X}_i + b) - 1) \tag{12}$$

Or, expanded out, we get:

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{m} \alpha_i (y_i(\mathbf{w}^T\mathbf{X}_i)) + b\sum_{i=1}^{m} \alpha_i y_i + \sum_{i=1}^{m} \alpha_i \tag{13}$$

[4]The proof of the existence of such $\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*$, such that $\mathbf{w}^*$, as well as the proof of the KKT conditions are out of the scope of this paper.

By applying equation 7, we get $0 = \frac{\partial}{\partial \mathbf{w}} \mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \mathbf{w} - \sum_{i=1}^{m} \alpha_i y_i \mathbf{X}_i$, or simply:

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{X}_i \tag{14}$$

Similarly, by applying equation 16, we get:

$$\frac{\partial}{\partial b} \mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i y_i = 0 \tag{15}$$

By equation 15, the term $b \sum_{i=1}^{m} \alpha_i y_i$ in equation 13 is equal to 0. By plugging in $\mathbf{w}$ as above, we get:

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \mathbf{w}^T \mathbf{w} = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \mathbf{X}_i^T \mathbf{X}_j \tag{16}$$

Observe that $\mathbf{X}_i^T \mathbf{X}_j$ is the standard inner product $\langle \mathbf{X}_i, \mathbf{X}_j \rangle$. The dual problem becomes:

$$\begin{aligned} \underset{\boldsymbol{\alpha}}{\text{maximize}} \quad & W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \mathbf{X}_i, \mathbf{X}_j \rangle \\ \text{subject to} \quad & \alpha_i \geq 0 \text{ for } \; i = 0, \ldots, m, \\ & \sum_{i=1}^{m} \alpha_i y_i = 0. \end{aligned} \tag{17}$$

Since the conditions for $p^* = d^*$ hold, we can solve this dual problem instead of the primal problem.

Also observe from equation 14 that our prediction, $\text{sgn}(\mathbf{w}^T \mathbf{X} + b)$ can be rewritten as $\text{sgn}((\sum_{i=1}^{m} \alpha_i y_i \mathbf{X}_i)^T \mathbf{X} + b)$. Simplifying, we can predict with:

$$\text{sgn}(\sum_{i=1}^{m} \alpha_i y_i \langle \mathbf{X}_i, \mathbf{X} \rangle + b) \tag{18}$$

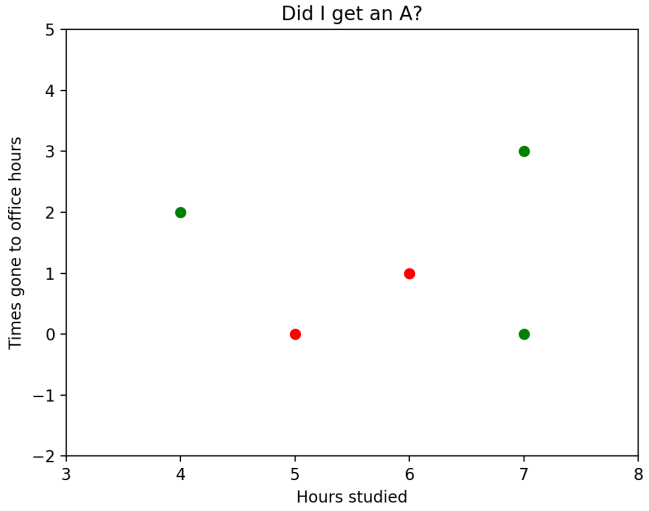
This implies that in order to make a new prediction, we need to calculate a value that depends only on support vectors, instead of having to run through the

whole dataset. This is a massive improvement in efficiency, when one considers that there can be an unbounded number of training examples. Moreover, the entire computation relies on inner products.

5 Dealing with Nonlinearly Separable Data

5.1 Motivation

Until now, we've assumed that the data we are dealing with can be neatly separated into one class or the other with a simple hyperplane. However, most data does not follow a linear pattern. Even data that does have an underlying linear model is prone to have random noise present. It is clear that there needs to be more versatility to allow for boundaries that are not so clear.



A simple example of nonlinearly separable data.

5.2 Soft Margins

If we have a barely non-separable dataset, such as one with a linear pattern but with random noise, it might make sense to allow only a few training examples to cross over the boundary. In other words, allow the SVM model to incorrectly classify a small amount of training examples which don't seem to fit the overall trend of the data. This is called a **soft margin**, as opposed to the hard margin in which no training examples are permitted to be misclassified.

Recall our original optimization problem (4). A standard way of altering it to allow a soft margin is to permit our training examples to have a functional margin of less than 1, but for each such example incur a cost to our objective function. The soft-margin optimization problem becomes

$$\begin{array}{ll} \underset{\gamma, \mathbf{w}, b}{\text{minimize}} & \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{m}\xi_i \\ \text{subject to} & y_i(\mathbf{w}^T\mathbf{X}_i + b) \geq 1 - \xi_i \text{ for } \ i = 0, \ldots, m, \\ & \xi_i \geq 0 \text{ for } \ i = 0, \ldots, m. \end{array} \tag{19}$$

where C is a hyper-parameter that controls how heavy a penalty crossing the margin is.

Employing our previous methods at solving optimization problems, we can easily see that the Lagrangian (6) of this new optimization problem is

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}, \boldsymbol{\gamma}, \boldsymbol{\xi}) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\alpha_i(y_i(\mathbf{w}^T\mathbf{X}_i + b) - 1 + \xi_i) - \sum_{i=1}^{m}\gamma_i\eta_i. \tag{20}$$

Observe that in relation to the structure of the primal problem (5), the soft-margin optimization problem has two $g-$like functions (inequality constraints). The γ_i represents the multipliers associated with the second inequality constraint.

After some substitution and relating the Lagrangian to the KKT conditions, we get the dual optimization problem:

$$\begin{array}{ll} \underset{\boldsymbol{\alpha}}{\text{maximize}} & W(\boldsymbol{\alpha}) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \mathbf{X}_i, \mathbf{X}_j \rangle \\ \text{subject to} & 0 \leq \alpha_i \leq C \text{ for } \ i = 0, \ldots, m, \\ & \sum_{i=1}^{m}\alpha_i y_i = 0. \end{array} \tag{21}$$

The KKT conditions (9-11) mean that:

$0 < \alpha_i < C \implies y_i(\mathbf{w}^T\mathbf{X}_i + b) = 1$

$\alpha_i = C \implies y_i(\mathbf{w}^T\mathbf{X}_i + b) \leq 1$

$\alpha_i = 0 \implies y_i(\mathbf{w}^T\mathbf{X}_i + b) \geq 1$

In particular, we automatically know the α_i values for any support vectors that are not on the border of our new SVM problem. Only training examples with functional margin equal to 1 have α_i values we need to calculate, and only training examples with functional margin less than or equal to 1 contribute to classifications.

5.3 Nonbinary data

Up until now, we've assumed that all data can be classified into one of two classes. In practice, however, we frequently want to classify more than two classes. For example, "is this a dog, a cat, or a bird?"

There are two popular ways of dealing with multi-label classification.

The **One vs All** method trains one binary classifier per class. In the above example, we would train one model that predicts whether or not a picture is a dog, another that predicts whether or not a picture is a cat, and a third that predicts whether or not the picture is a bird. The one with the highest score (in SVMs, geometric margin), is predicted as the correct label. If there are n classes, we have n binary classifiers.

The **One vs One** method trains a binary classifier for each pair of labels. In the above example, we would train a model for each of Cat vs Dog, Cat vs Bird, Bird vs Dog. When an unlabeled example is input, we would predict the label which has the most "+1" predictions. If there are n classes, we have $n(n-1)/2$ binary classifiers.

Both have their upsides and downsides. When it's more important to correctly classify every class, One-Vs-One is used. When we are mostly concerned about computability and speed, One vs All is used, since less SVMs need to be trained.

5.4 Kernel Functions

Recall that the prediction model for SVMs relies only on the inner product between support vectors and new training examples.

In many ways, the standard inner product $\langle \mathbf{X}, \mathbf{X}' \rangle$ between two vectors measures similarity. Ignoring magnitudes, the inner product is maximized when $\mathbf{X} = \mathbf{X}'$ and minimized when they are far apart.

An idea to exploit this property is to change the inner product. The resulting method, called **Kernel Functions**, or the **Kernel Trick**, does just that.

A Kernel Function is a function K such that

$$K(\mathbf{X}, \mathbf{X}') = \phi(\mathbf{X})^T \phi(\mathbf{X}') \tag{22}$$

for some feature mapping ϕ.

A simple example of a Kernel Function is $K(\mathbf{X}, \mathbf{X}') = (\mathbf{X}^T\mathbf{X}' + 1)^2$. This has a feature mapping:

$$\phi(\mathbf{X}) = [X_1X_1, X_1X_2, \ldots, X_1X_m, X_2X_1, \ldots X_m, \ldots X_mX_m, \sqrt{2}X_1, \sqrt{2}X_2, \ldots, \sqrt{2}X_m, 1].$$

You may verify that the above feature mapping ϕ satisfies

$$(\mathbf{X}^T\mathbf{X}' + 1)^2 = \phi(\mathbf{X})^T \phi(\mathbf{X}').$$

This is quite a complicated feature mapping. For most practical applications, finding the implicit function ϕ is difficult, but finding the kernel $K(\mathbf{X}, \mathbf{X}')$ for two vectors $\mathbf{X}, \mathbf{X}'$ is fairly easy. It suffices to calculate $K(\mathbf{X}, \mathbf{X}')$ without finding ϕ explicitly.

As seen in the example feature mapping above, Kernels implicitly encode mixed terms from vector elements. For example, $X_1 \cdot X_7'$. In doing so, we greatly expanded the number of features used. The versatility of Kernels in SVMs comes from the fact that Kernels allow SVMs to learn in a higher dimensional feature space than the number of features would initially suggest.

5.5 Popular Kernels

The Linear Kernel is given as $K(\mathbf{X}, \mathbf{X}') = \mathbf{X}^T\mathbf{X}' + c$. It is just the standard inner product, with an optional hyper-parameter c.

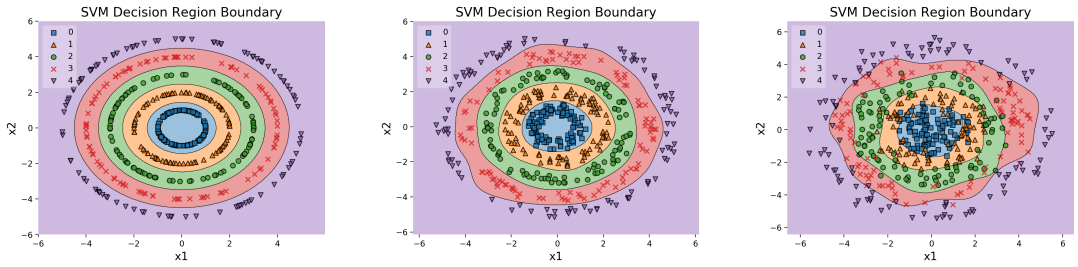
The Polynomial Kernel is given as $K(\mathbf{X}, \mathbf{X}') = (\mathbf{X}^T\mathbf{X}' + c)^d$. It is popular in image processing. c and d are adjustable hyper-parameters; d is called the degree of the Polynomial Kernel.

The Radial Basis/Gaussian Kernel is given as $K(\mathbf{X}, \mathbf{X}') = e^{-\frac{||\mathbf{X}-\mathbf{X}'||^2}{2\sigma^2}}$. It is useful for when there is no assumption made on the data. As such, RBF is used in nearly all data classification fields.

The Hyperbolic Tangent/Sigmoid Kernel is given as $K(\mathbf{X}, \mathbf{X}') = \tanh(\alpha\mathbf{X}^T\mathbf{X}' + c)$. It is used in neural networks (not covered), but is not very useful in standard data classification.

5.6 Combining the Two

While soft margins and Kernels individually create a lot of versatility for Support Vector Machines, when combined they add a lot of prediction power. Kernels utilizing soft margins can discover imperfect relationships between variables fairly easily. For example, the below images draw random data points from circles of various radius, and add random noise. All are trained using a Gaussian Kernel. The first has no noise, and the decision margin is very clean. The second adds a little noise, and the SVMs are able to create a decision boundary which correctly predicts all training examples without overlap. The last adds more noise, and the resulting data is not separable using a reasonable model. With soft margins, the SVMs predict some of the training examples incorrectly but provide a much more scaleable model. The SVMs trained use the one-vs-one classification method.



6 Conclusion

Support Vector Machines are a key tool in any data analysis. In this paper, we discussed how to arrive at the maximum margin separating hyperplane, how to deal with barely-inseparable data, and how to classify multi-class problems. Combining the different approaches, we arrive at a powerful data classification method, the so called "Support Vector Machine". SVMs are widely used throughout the field of machine learning, and continue to perform competitively when compared to other machine learning methods.

7 References

"cesarsouza, /. ?Csar Souza.? Csar Souza, crsouza.com/2010/03/17/kernel-functions-for-machine-learning-applications/."

References

[1] Cesar Souza: Kernel Functions for Machine Learning Applications,
crsouza.com/2010/03/17/kernel-functions-for-machine-learning-applications/

[2] Cristianini, Nello, and John Shawe-Taylor:
An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods.
Cambridge University Press, 2014.

[3] DataFlair: Kernel Functions-Introduction to SVM Kernel & Examples,
data-flair.training/blogs/svm-kernel-functions/

[4] Andrew Ng: CS229 Lecture Notes. Support Vector Machines,
see.stanford.edu/materials/aimlcs229/cs229-notes3.pdf.

[5] Plot SVM with Matplotlib?,
stackoverflow.com/questions/43284811/plot-svm-with-matplotlib

[6] Scholkopf, Bernhard, and Alexander J. Smola:
Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond.
MIT Press, 2009.

[7] Soft Margin Classification,
Support Vector Machines: The Linearly Separable Case,
nlp.stanford.edu/IR-book/html/htmledition/soft-margin-classification-1.html