Support Vector Machine Review

Akhil Vasvani

March 2019

1 Questions

Exercise 1. How can the SVM optimization function be derived from the logistic regression optimization function?

Proof. Recall that logistic regression is defined as the sigmoid function:

$$g(\boldsymbol{z}) = \sigma(\boldsymbol{\theta}^\top \boldsymbol{x}) = \frac{1}{1 + e^{-(\boldsymbol{\theta}^\top \boldsymbol{x})}}, \tag{1}$$

where $\boldsymbol{\theta}$ are our weights and $\boldsymbol{x}$ is our input. So j can be positive non-zero integer

Note. The input $\boldsymbol{x}$ is a design matrix of shape $j \times i$ and $\boldsymbol{\theta}$ has a shape of $j \times 1$

Now, logistic regression outputs probabilities that are used to make a classification. In laymen's terms, if the outputted probability is greater than a threshold than we pick one class while if the outputted probability is less than a threshold than we pick the other class.

More formally put, say $\mathbf{y}$ is our class predictor and there are two classes: 0, 1:

If $y = 1$, then we want $\sigma(\boldsymbol{\theta}^\top \boldsymbol{x}) \approx 1 \rightarrow \boldsymbol{\theta}^\top \boldsymbol{x} \gg 0$.

If $y = 0$, then we want $\sigma(\boldsymbol{\theta}^\top \boldsymbol{x}) \approx 0 \rightarrow \boldsymbol{\theta}^\top \boldsymbol{x} \ll 0$.

Because there are two classes, this feels extremely similar to a Bernoulli distribution. Borrowing that distribution, and modifying it slightly we can arrive at the conditional probability $P(\mathbf{y} \mid \mathbf{x}; \boldsymbol{\theta})$ in order to predict $\mathbf{y}$ given $\mathbf{x}$:

$$P(\mathbf{y} = y \mid \mathbf{x}; \boldsymbol{\theta}) = P(\mathbf{y} = y; \sigma(\boldsymbol{\theta}^\top \boldsymbol{x})) = (\sigma(\boldsymbol{\theta}^\top \boldsymbol{x}))^y (1 - \sigma(\boldsymbol{\theta}^\top \boldsymbol{x}))^{1-y} = \Big(\frac{1}{1 + e^{-\boldsymbol{\theta}^\top \boldsymbol{x}}}\Big)^y \Big(1 - \frac{1}{1 + e^{-\boldsymbol{\theta}^\top \boldsymbol{x}}}\Big)^{1-y}. \tag{2}$$

Now, our goal is to find the optimal $\boldsymbol{\theta}$ to satisfy the above equation, so we the maximum likelihood procedure to find $\boldsymbol{\theta}_{\text{ML}}$:

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} P(\boldsymbol{Y} \mid \boldsymbol{X}; \boldsymbol{\theta}). \tag{3}$$

We can also further assume that each example is i.i.d (identically and independently distributed), then this can be decomposed down into

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log P(\boldsymbol{y}^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta}) = -\arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log \Big[(\sigma(\boldsymbol{\theta}^\top \boldsymbol{x}^{(i)}))^{y^{(i)}} (1 - \sigma(\boldsymbol{\theta}^\top \boldsymbol{x}^{(i)}))^{1-y^{(i)}} \Big]$$

$$= -\arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{m} y^{(i)} \log(\sigma(\boldsymbol{\theta}^\top \boldsymbol{x}^{(i)})) + (1 - y^{(i)}) \log(1 - \sigma(\boldsymbol{\theta}^\top \boldsymbol{x}^{(i)}))$$

$$= -\arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{m} y^{(i)} \log\Big[\frac{1}{1 + e^{-\boldsymbol{\theta}^\top \boldsymbol{x}^{(i)}}}\Big] + (1 - y^{(i)}) \log\Big[1 - \frac{1}{1 + e^{-\boldsymbol{\theta}^\top \boldsymbol{x}^{(i)}}}\Big].$$

However, note that if we sum all the training examples m (the total number of rows), we will get $m\boldsymbol{\theta}_{\text{ML}}$, so we must divide the above equation by m. This will result in an unregularized logistic regression:

$$\arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \Big(-\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log\Big[\frac{1}{1 + e^{-\boldsymbol{\theta}^\top \boldsymbol{x}^{(i)}}}\Big] + (1 - y^{(i)}) \log\Big[1 - \frac{1}{1 + e^{-\boldsymbol{\theta}^\top \boldsymbol{x}^{(i)}}}\Big]\Big) \quad (4)$$

However, let's say the number of training example we have is small so we want to add some penalty for large θ values, so we impose a L^2 regularization term $\frac{\lambda}{2}||\theta||^2$. Remember, we also divided the entire equation by m so we must add that into our regularization term $\frac{\lambda}{2m}||\theta||^2 = \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$. Next, we add that onto our cost function $J(\boldsymbol{\theta})$:

$$\arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \Big(-\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log\Big[\frac{1}{1 + e^{-\boldsymbol{\theta}^\top \boldsymbol{x}^{(i)}}}\Big] + (1 - y^{(i)}) \log\Big[1 - \frac{1}{1 + e^{-\boldsymbol{\theta}^\top \boldsymbol{x}^{(i)}}}\Big] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2\Big). \quad (5)$$

Note. n represents the number of features. The bounds for the summation for the regularization term are from $j = 1$ to n because we do not regularize the θ_0 term and because we can set which weights we want to penalize.

Now, let's say we set $cost_1(\boldsymbol{\theta}^\top \boldsymbol{x}^{(i)}) = -\log(\sigma(\boldsymbol{\theta}^\top \boldsymbol{x}^{(i)}))$ and $cost_0(\boldsymbol{\theta}^\top \boldsymbol{x}^{(i)}) = -\log(1 - \sigma(\boldsymbol{\theta}^\top \boldsymbol{x}^{(i)}))$ and plugged that back in:

$$\rightarrow \arg\min_{\boldsymbol{\theta}} \Big(\frac{1}{m}\Big[\sum_{i=1}^{m} y^{(i)} cost_1(\boldsymbol{\theta}^\top \boldsymbol{x}^{(i)}) + (1 - y^{(i)}) cost_0(\boldsymbol{\theta}^\top \boldsymbol{x}^{(i)})\Big] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2\Big)$$

.

Let's remove $\frac{1}{m}$ because it does not matter and multiply everything instead by: $\frac{1}{\lambda}$.

Example. Let's do example to explain why we can do this. Say,

$$\min_u (u - 5)^2 + 1 \rightarrow 2u - 10 = 0 \rightarrow u = 5.$$

Now,

$$\min_u 10(u - 5)^2 + 10 \rightarrow 20u - 100 = 0 \rightarrow u = 5.$$

Regardless of the multiplicative constant, the minimum will always be 5.

$$\Rightarrow \arg\min_{\boldsymbol{\theta}} \Big(C \sum_{i=1}^{m} y^{(i)} cost_1(\boldsymbol{\theta}^\top \boldsymbol{x}^{(i)}) + (1 - y^{(i)}) cost_0(\boldsymbol{\theta}^\top \boldsymbol{x}^{(i)}) + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2 \Big),$$

where $C = \frac{1}{\lambda}$.

Thus, we arrive at the cost function for **Support Vector Machines** (SVM). In comparison to logistic regression, it does not output a probability but we get to know directly whether the prediction is in class 1 or class 0.

$$\sigma(\boldsymbol{\theta}^\top \boldsymbol{x}) = \left\{ \begin{array}{cc} 1 & \text{if } \boldsymbol{\theta}^\top \boldsymbol{x} \geq 0 \\ 0 & \text{other} \end{array} \right\}$$

Plotting the cost_1 and cost_0 function looks something like this:

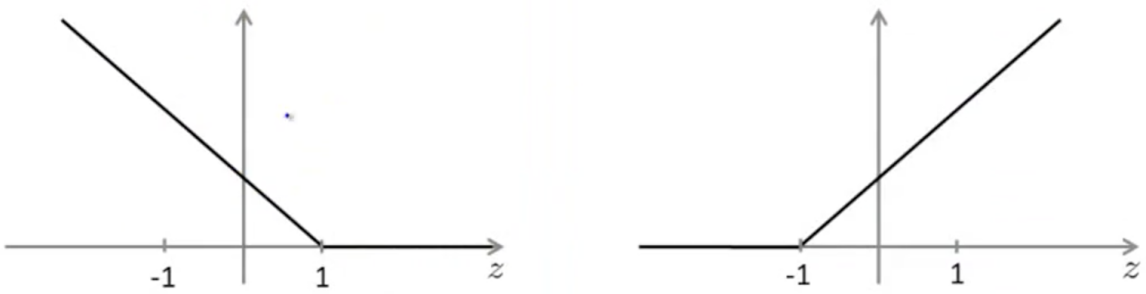


Figure 1: cost_1 (left), cost_0 (right)

Thus, if we really want:

$$y^{(i)} = \left\{ \begin{array}{cc} 1 & \boldsymbol{\theta}^\top \boldsymbol{x} \geq 1 \\ 0 & \boldsymbol{\theta}^\top \boldsymbol{x} \leq -1 \end{array} \right\}$$

What we have done is just have simplified the cost function in order to use geometry for further steps.

□

Exercise 2. What is a large margin classifier? What is the mathematical intuition of a large margin classifier?

Proof. In the figure below, there is some data plotted between two variables.

You wish to find some type of line separating the plotted data into two groups, so that you easily distinguish between the two. More formally put, the line we are creating is called a **decision boundary**. One side of the decision boundary will belong to one class and all those on the other side will belong to the other class—call one class (label 1) and the other class (label -1).

Note. A decision boundary or decision surface is a hypersurface that partitions the underlying vector space into two sets, one for each class. So technically, a decision boundary does not always have to be line. It could be a plane. We will see more of that in a little bit.

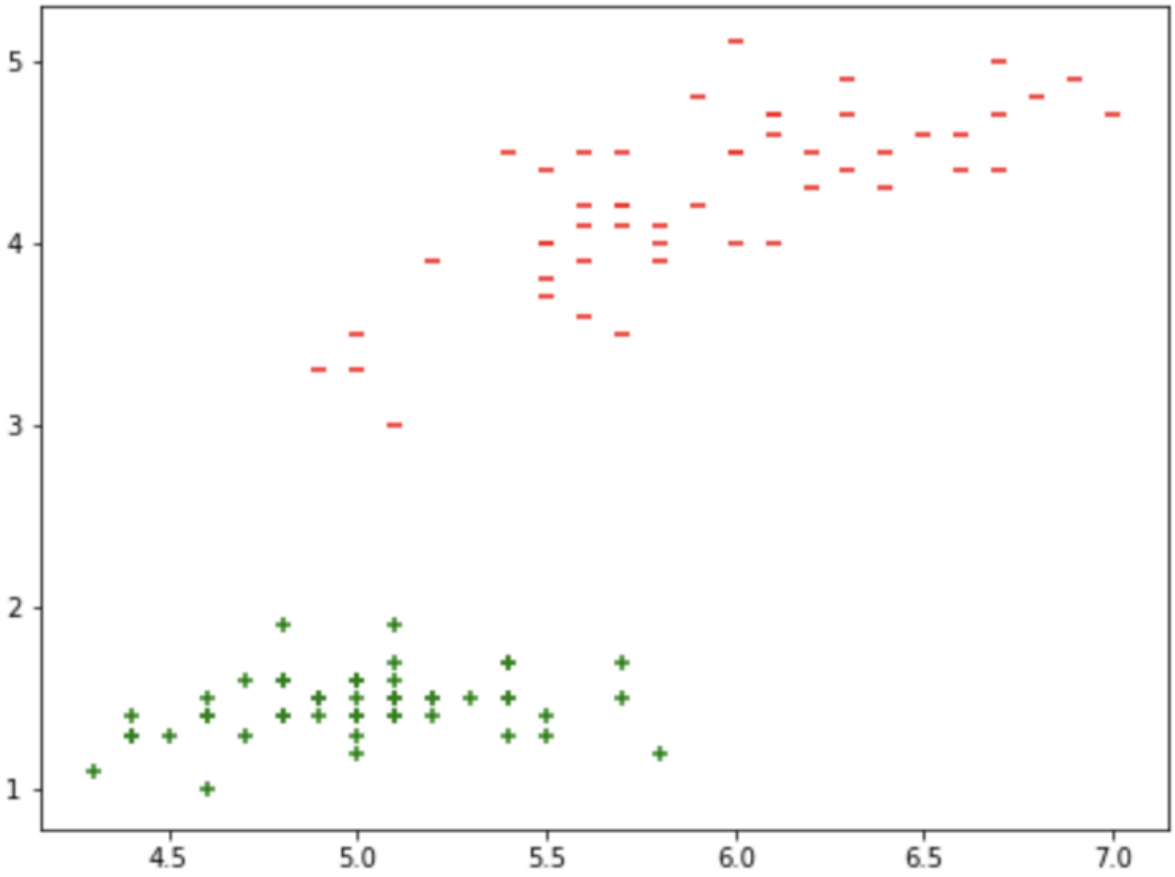


Figure 2: Linearly separable data

Now, how do we go about creating this decision boundary? The function of a line is $y = ax + b$. Let's rename x with x_1 and y with x_2 and rewrite this equation in standard form: $ax_1 - x_2 + b = 0$.

If we define $\boldsymbol{x} = (x_1, x_2)$ and $\boldsymbol{w} = (a, -1)$, we get:

$$\boldsymbol{w} \cdot \boldsymbol{x} + b = 0. \tag{6}$$

Once we have the decision boundary, we can then use it to make predictions. We define the hypothesis function h as:

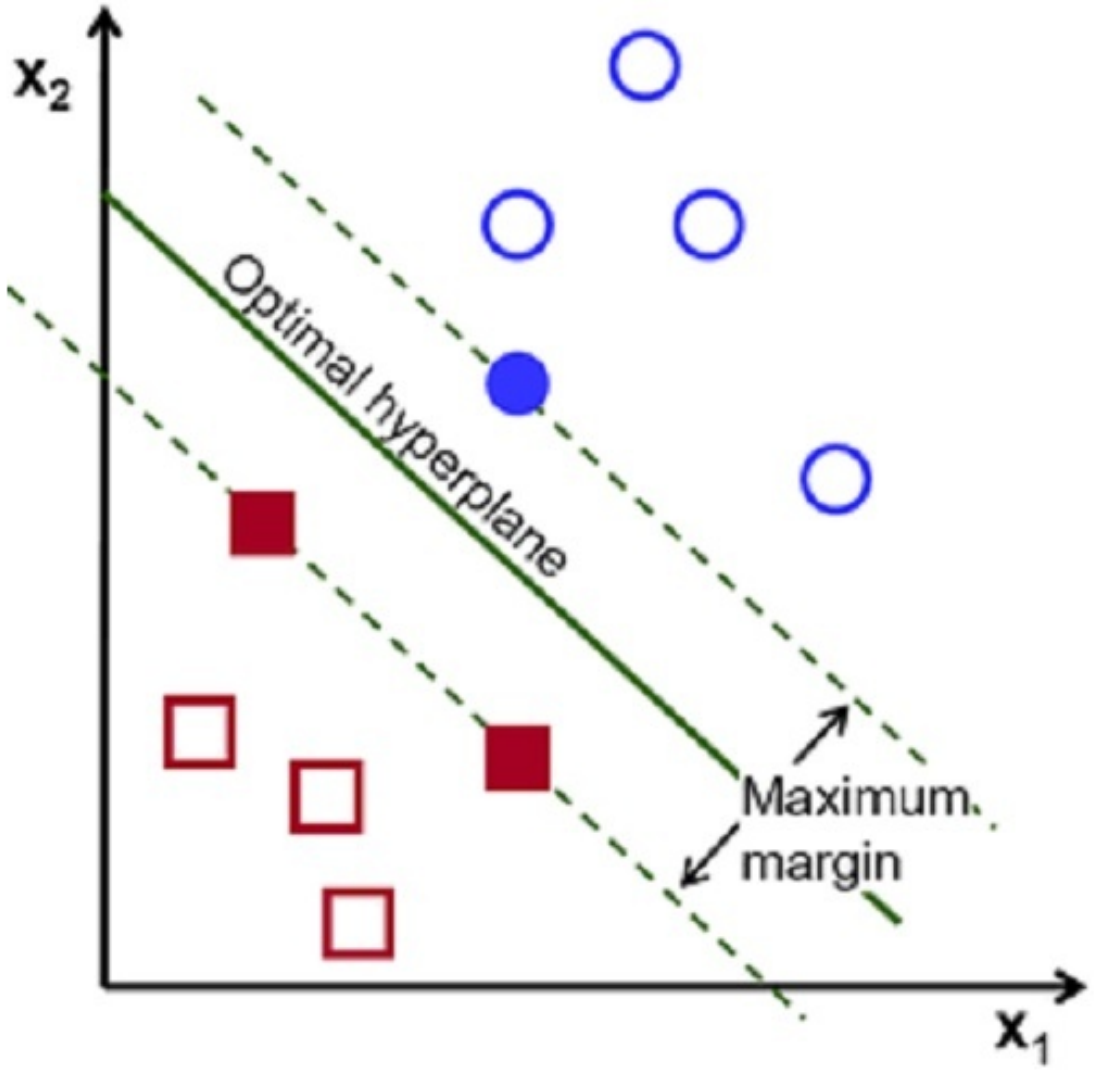
$$h(x_i) = \left\{ \begin{array}{ll} +1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b \geq 0 \\ -1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b < 0 \end{array} \right\}$$

The point above or on the line will be classified as class +1, and the point below the line will be classified as class -1.

Then, the question arises what happens when there are more than three dimensions. What do we use to separate the multi-dimensional data? We use a **hyperplane**, a type of decision boundary. And the equation of the **hyperplane** is the exact same as the one above, which in fact works for any number of dimensions.

Let's say you have found a hyperplane that completely separates the two classes in your training set. We expect that when new data comes along (i.e. your test set), the new data will look like your training data. Points that should be classified as one class or the other should lie near the points in your training data with the corresponding class. Now, if your hyperplane is oriented such that it is close to some of the points in your training set, there's a good chance that the new data will lie on the wrong side of the hyperplane, even if the new points lie close to training examples of the correct class.

Therefore, we want to find the hyperplane with the **maximum margin**. In laymen's terms, find a hyperplane that divides your data properly, but is also as far as possible from your data points. That way, when new data comes in, even if it is a little closer to the wrong class than the training points, it will still lie on the right side of the hyperplane.



Note. One important note that should be mentioned is **linear separability**. Linear separable means that the data (in the two-dimensional case) can be separated by a line or can

be separated by a hyperplane (in the n-dimensional case where $n > 2$). However, there are times when the data is **non-linearly separable**—there does not exist a line to separate the data in the two dimensional case.

The goal of the SVM learning algorithm is to find a hyperplane which could separate the data accurately. There might be many such hyperplanes. And we need to find the best one, which is often referred as the **optimal hyperplane**.

TL;DR. A **large margin classifier** is a classifier which is able to give an associated distance from the **decision boundary** for each example. For instance, if a linear classifier is used, the distance (typically euclidean distance, though others may be used) of an example from the separating hyperplane is the margin of that example. If your data is separable, then there are infinitely many hyperplanes that will separate it. SVM (and some other classifiers) optimizes for the one with the maximum margin.

□

Exercise 3. What are support vectors?

Note. The $\boldsymbol{\theta}$ parameter is the SAME as the $\boldsymbol{w}$ parameter.

Proof. Let $\boldsymbol{\theta}$ be the minimizer of the SVM cost function for some training dataset with m examples: $(\boldsymbol{x}^{(i)}, y^{(i)})$. Then for $i, ..., m$ there exists $\alpha_i \geq 0$ such that the optimum θ can be written:

$$\boldsymbol{\theta} = \sum_{i=1}^{m} \alpha_i y^{(i)} \boldsymbol{x}^{(i)}$$

So, if

$$\alpha_i = 0 \Rightarrow y^{(i)} \boldsymbol{\theta} x^{(i)} \geq 1$$

$$\alpha_i = C \Rightarrow y^{(i)} \boldsymbol{\theta} x^{(i)} \leq 1.$$

All points on the wrong side of the margin! Where C is constant (say its a large value). But,

$$0 \leq \alpha_i \leq C \Rightarrow y^{(i)} \boldsymbol{\theta} x^{(i)} = 1.$$

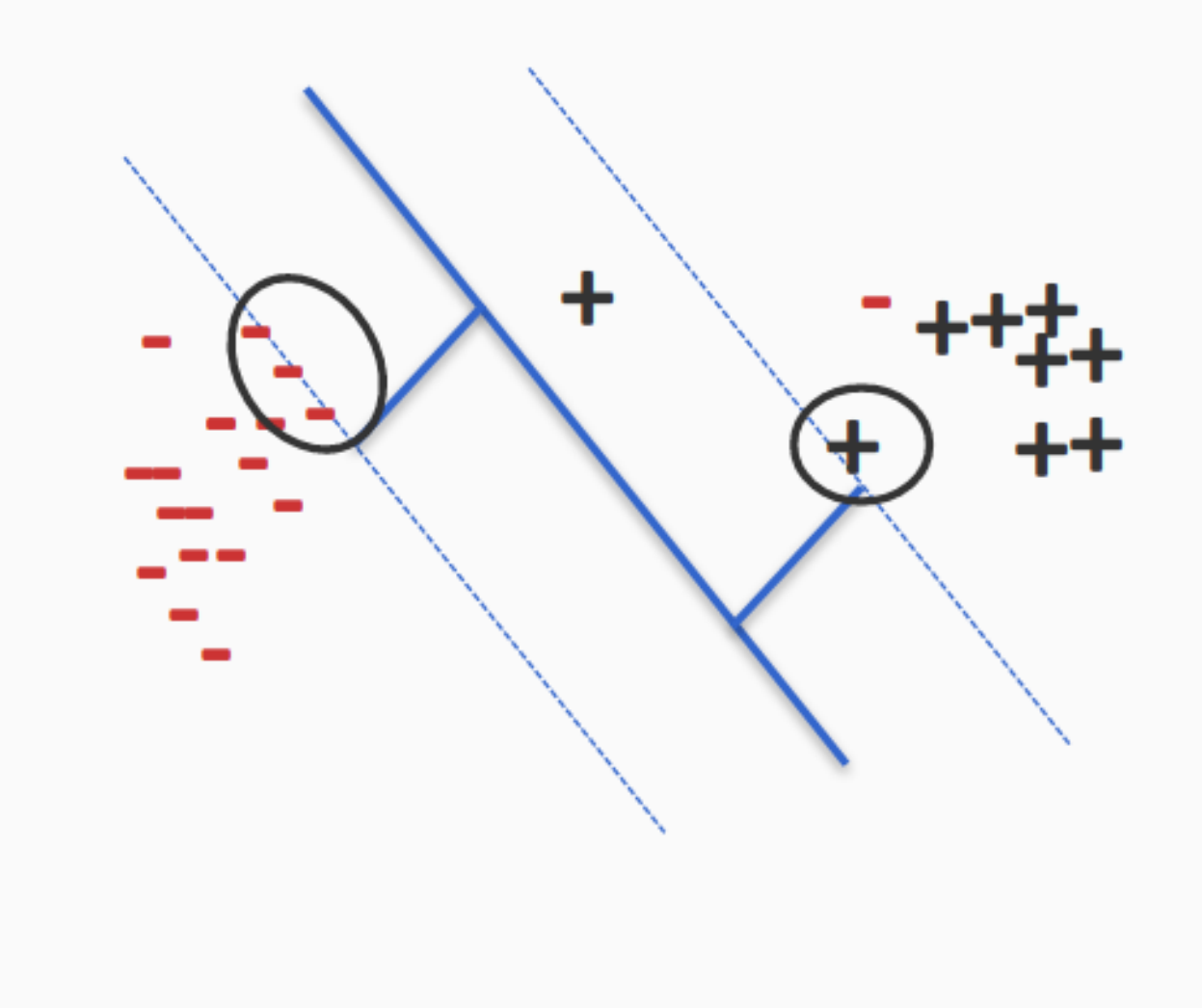
These are all points on the margin.

Therefore, the $\boldsymbol{\theta}$ vector is completely defined by training examples whose α_is are nonzero

$$\boldsymbol{\theta} = \sum_{i=1}^{m} \alpha_i y^{(i)} \boldsymbol{x}^{(i)}.$$

These examples are called **support vectors**.

□

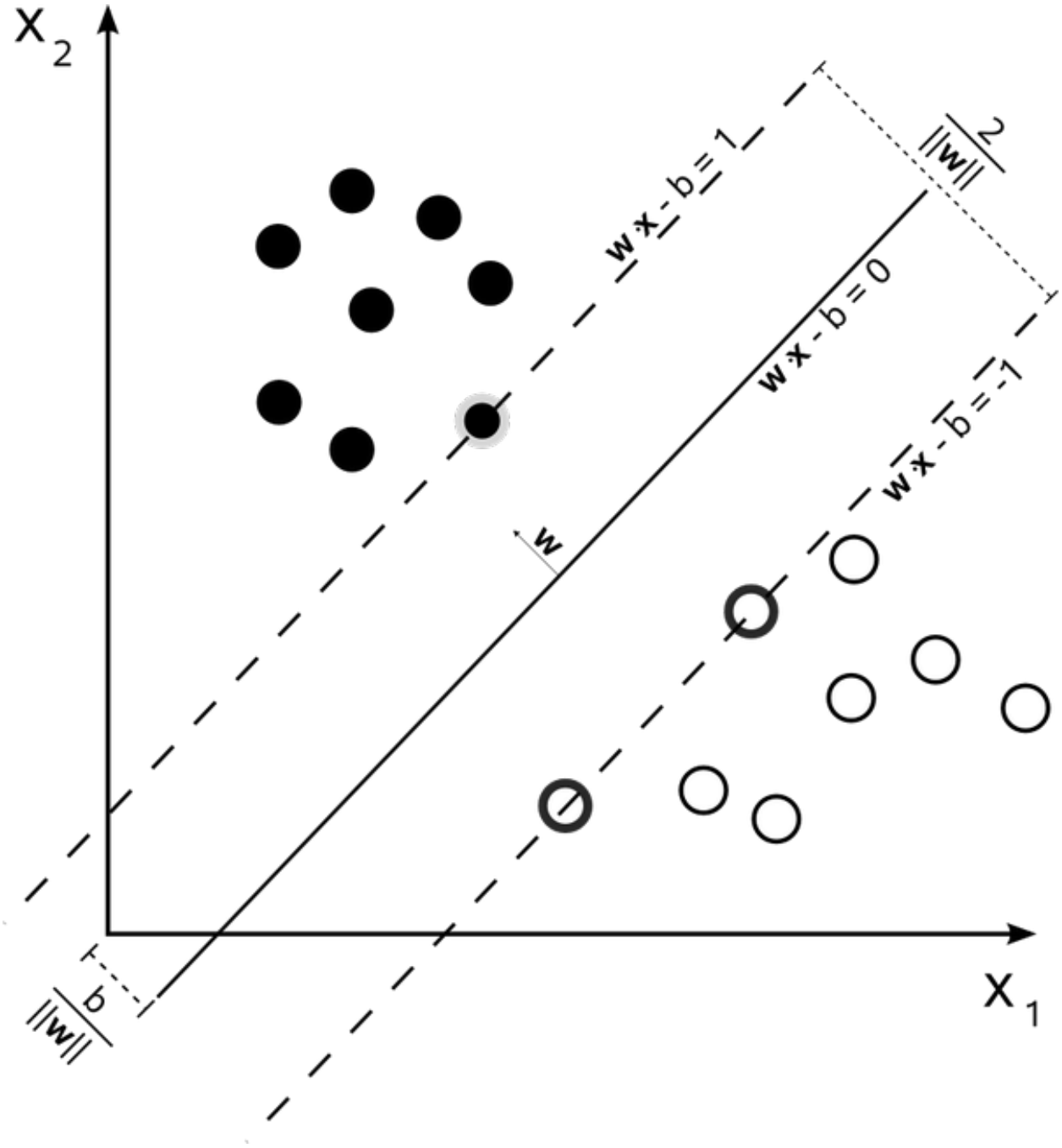


Exercise 4. Why SVM is an example of a large margin classifier?

Proof. SVM is an example of a large margin classifier because it places the decision boundary such that it maximizes the distance between two clusters—the positive and negative examples (equal distant from the boundary lines), which avoids overfitting.

Let's consider a linear SVM that separates two classes $y = 1, y = -1$. It will find a hyperplane such that: $\boldsymbol{w} \cdot \boldsymbol{x} - b = 0$, where $\boldsymbol{x}$ are the points on the plane, and $\boldsymbol{w}$ is the normal vector to the plane. The closest points from the two categories will be found when $\boldsymbol{w} \cdot \boldsymbol{x} - b = 1$ and $\boldsymbol{w} \cdot \boldsymbol{x} - b = -1$, respectively.

Because this is a linear separation, it must be the case that halfway between the two edges, the margin is maximized (which is the optimal hyperplane). Hence, this is why $\boldsymbol{w} \cdot \boldsymbol{x} - b = 0$ maximizes the margin. Therefore, SVM is an example of a large margin classifier.


X2
X1
w x - b = 1
w x - b = 0
w x - b = -1
2 / ||w||
w
b / ||w||

Note. If you for some reason do not want to maximize the margin, you can change the zero for a number that better suits your problem.

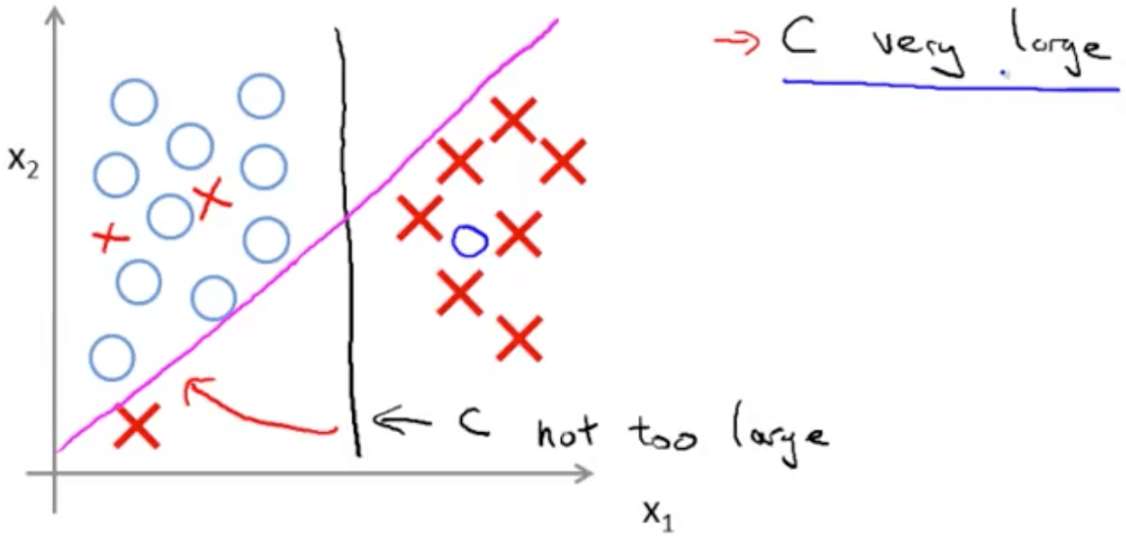
Also $\frac{b}{||\boldsymbol{w}||}$ is the offset from the origin and $\frac{2}{||\boldsymbol{w}||}$ is the width of the margin.

□

Exercise 5. Since SVM being a large margin classifier, is it influenced by outliers?

Proof. Yes, SVM is influenced by outliers if C is large. Otherwise it is not.

Note. As C gets larger and larger, the decision boundary goes from a fully vertical line to a horizontal line.



□

Exercise 6. What is the role of C in SVM?

Proof. The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. For very tiny values of C, you should get misclassified examples, often even if your training data is linearly separable. □

Exercise 7. In SVM, what is the angle between the decision boundary and θ?

Proof. The angle between the decision boundary and θ is orthogonal (at 90°).

Note. Note, this is a little confusing notation. The $\boldsymbol{\theta}$ parameter we used in the cost function is the same as the $\boldsymbol{w}$ parameter in the hyperplane equation.

□

Exercise 8. What is a kernel in SVM? Why do we use kernels in SVM?

Proof. Sometimes, your data in two dimensions is non-linearly separable.

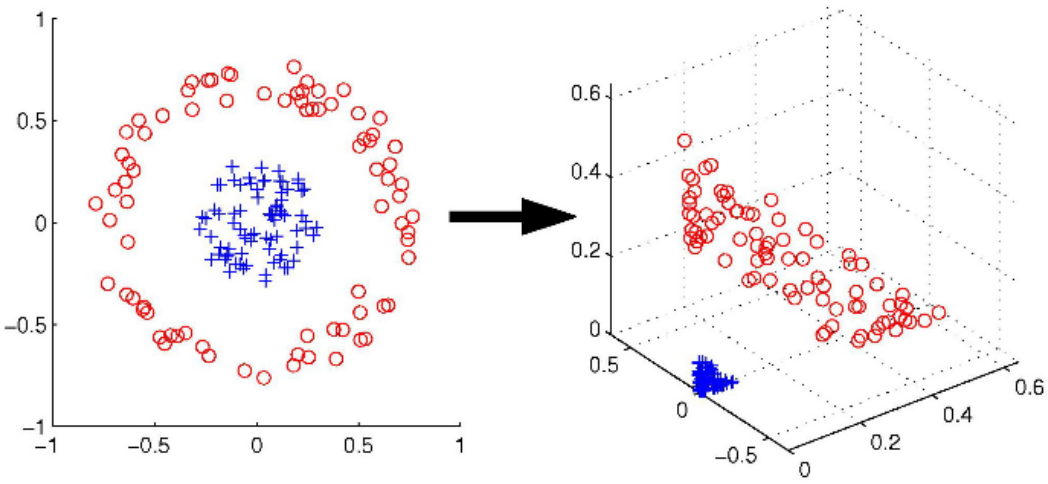


Figure 1: Transforming the data can make it linearly separable

Yet if we map it to a three-dimensional space using

$$\phi : \mathbb{R}^2 \to \mathbb{R}^3 \tag{7}$$

$$(x_1, x_2) \to (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \tag{8}$$

and if we try to linearly separate the mapped data, our decision boundaries will be hyperplanes in $\mathbb{R}$, of the form $\boldsymbol{\theta}^\top \boldsymbol{z} + b = 0$ i.e. as a function of x they will be of the form

$$w_1x_1^2 + w_2\sqrt{2}x_1x_2 + w_3x_2^2 = 0.$$

Now, let's say we provide a test set $(\boldsymbol{x}^{(j)}, \boldsymbol{y}^{(j)})$. Our goal is to predict $\boldsymbol{y}^{(j)} = \sigma(\boldsymbol{\theta}^\top \boldsymbol{x}^{(j)})$, where we define $\boldsymbol{\theta}$ to be made up from our support vectors $\boldsymbol{\theta} = \sum_i^m \alpha_i y_i \boldsymbol{x}^{(i)}$ where $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$ is our training set made up of m examples. Therefore we can represent our hyperplane equation as the dot product of all the support vectors and the test examples,

$$\boldsymbol{\theta}^\top \boldsymbol{x}^{(j)} = \sum_i \alpha_i y_i \boldsymbol{x}^{(i)} \cdot \boldsymbol{x}^{(j)}.$$

which is basically the dot products between training examples and the new (test) example $\boldsymbol{x}^{(j)}$. This is true even if we map examples to a high dimensional space

$$\boldsymbol{\theta}^\top \phi(\boldsymbol{x}^{(j)}) = \sum_i \alpha_i y_i \phi(\boldsymbol{x}^{(i)})^\top \cdot \phi(\boldsymbol{x}^{(j)}).$$

Now, we will call this modified inner product $\phi(\boldsymbol{x}^{(i)})^\top \cdot \phi(\boldsymbol{x}^{(j)}) = K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$ a **kernel**. We shall also call K the kernel matrix because it contains the value of the kernel for every pair of data points, thus using the same letter both for the function and its matrix.

Note. The kernel matrix K is known as **Gram Matrix**

$$K = \begin{bmatrix} x^{(1)\top}x^{(1)} & x^{(1)\top}x^{(2)} & \dots \\ x^{(2)\top}x^{(1)} & \dots & \\ \dots & & \end{bmatrix} = X^{(i)} \cdot X^{(j)\top}.$$

The Gram matrix has an interesting property: it is a positive semidefinite matrix

Now, maping our data via ϕ and computing the inner product for each pair of $\boldsymbol{x}^{(i)}$) and $\boldsymbol{x}^{(j)}$) to build the Gram matrix is very computational slow. But in fact, we can do it in one operation, leaving the mapping completely implicit. We do not even need to know ϕ, all we need to know is how to compute the modified inner product or the kernel. This is called the **kernel trick**.

The kernel trick is powerful for two reasons. First it calculates the result of a dot product performed in another space. We now have the ability to change the kernel function in order to classify non-linearly separable data. Second, the kernel function k often admits an implementation that is significantly more computational efficient than naively constructing $\boldsymbol{x}$ vectors and explicitly taking their dot product.

There are multiple kernel types we could use to classify the data. Some of the most popular ones are **linear kernel**, **polynomial kernel**, and **RBF kernel**.

The **linear kernel** is defined as: $k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \boldsymbol{x}^{(i)} \cdot \boldsymbol{x}^{(j)}$. This is the same as the one we used in the above discussion. In practice, you should know that a linear kernel works well for text classification.

The **polynomial kernel** is defined as:

$$k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = (x^{(i)} \cdot x^{(j)} + c)^d$$

This kernel contains two parameters: a constant c and a degree of freedom d. A d value with 1 is just the linear kernel. A larger value of d will make the decision boundary more complex and might result in overfitting.

The **Radial Basis Function (RBF) kernel** is defined as: $k(\boldsymbol{u}, \boldsymbol{v}) = \mathcal{N}(\boldsymbol{u} - \boldsymbol{v}; 0, \sigma^2 \boldsymbol{I})$. The RBF kernel is also called the Gaussian kernel. It will result in a more complex decision boundary. A small difference between $\boldsymbol{u} - \boldsymbol{v}$ will make the model behave like a linear SVM. A large difference between $\boldsymbol{u} - \boldsymbol{v}$ will make the model heavily impacted by the support vectors examples.

In practice, it is recommended to try RBF kernel first cause it normally performs well.

TL;DR. SVM algorithms use a set of mathematical functions that are defined as the kernel. The function of kernel is to take data as input and transform it into the required form. Different SVM algorithms use different types of kernel functions. These functions can be different types. For example linear, nonlinear, polynomial, radial basis function (RBF), and sigmoid. Introduce Kernel functions for sequence data, graphs, text, images, as well as vectors. The most used type of kernel function is RBF. Because it has localized and finite response along the entire x-axis. The kernel functions return the inner product between two points in a suitable feature space. Thus by defining a notion of similarity, with little computational cost even in very high-dimensional spaces.

□

Exercise 9. What is a similarity function in SVM? Why it is named so?

Proof. Now, let's talk about a slightly different view of kernels. Intuitively, (and there are things wrong with this intuition, but nevermind), if $\phi(\boldsymbol{x}^{(i)})$ and $\phi(\boldsymbol{x}^{(j)})$ are close together, then we might expect $K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \phi(\boldsymbol{x}^{(i)})^\top \cdot \phi(\boldsymbol{x}^{(j)})$ to be large. Conversely, if $\phi(\boldsymbol{x}^{(i)})$ and $\phi(\boldsymbol{x}^{(j)})$ are far apart—say nearly orthogonal to each other—then $K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \phi(\boldsymbol{x}^{(i)})^\top \cdot \phi(\boldsymbol{x}^{(j)})$ will be small. So, we can think of $K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$ as some measurement of how similar are $\phi(\boldsymbol{x}^{(i)})$ and $\phi(\boldsymbol{x}^{(j)})$, or of how similar are $\boldsymbol{x}^{(i)}$ and $\boldsymbol{x}^{(j)}$.

Given this intuition, suppose that for some learning problem that you are working on, you have come up with some function $K(x, z)$ that you think might be a reasonable measure of how similar x and z are. For instance, perhaps you chose

$$K(x, z) = \exp\big(-\frac{||x - z||^2}{2\sigma^2}\big).$$

The Gaussian kernel gives a reasonable measure of x and z's similarity, and is close to 1 when x and z are close, and near 0 when x and z are far apart. Given two vectors, the similarity will diminish with the radius of σ. The distance between two objects is "reweighted" by this radius parameter.

Note. Perform feature scaling before you using Gaussian kernel.

TL;DR. A very simple and intuitive way of thinking about kernels (at least for SVMs) is a similarity function. Given two objects, the kernel outputs some similarity score. The objects can be anything starting from two integers, two real valued vectors, trees whatever provided that the kernel function knows how to compare them.

□

Exercise 10. How are the landmarks initially chosen in an SVM? How many and where?

Proof. To compute the landmarks we adapt the cost function to the following:

$$\arg\min_{\boldsymbol{\theta}} \Big(C \sum_{i=1}^{m} y^{(i)} cost_1(\boldsymbol{\theta}^\top \boldsymbol{f}^{(i)}) + (1 - y^{(i)}) cost_0(\boldsymbol{\theta}^\top \boldsymbol{f}^{(i)}) + \frac{1}{2} \sum_{j=1}^{m} \theta_j^2 \Big),$$

in which we denote each feature as f.

Note. How to Choose Landmarks?

Suppose we have m training examples $\{(x^{(i)}, y(i))\}$. Now we just put a landmark at the exactly same locations, i.e. $l^{(i)} = x^{(i)}$ and we will end up with m landmarks $l^{(i)}$. Keep in mind that for the regularizing part, instead of n (number of features) m (training examples) should be used. This makes sense, since we want to calculate the landmarks which are related to the examples.

Example. Suppose we have the following model:

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$$

And $\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 1 \\ 1 \\ 0 \end{bmatrix}$.

Say we have a x near $l^{(1)}$ (green one on the left). If we have $f^{(1)} \approx 1$,$f^{(2)} \approx 0$, and $f^{(3)} \approx 0$, putting it to the model we get

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \approx 0.5 + 1 = 0.5 \geq 0,$$

so we predict $y = 1$.

Next, say we have a x close to $l^{(3)}$ (blue one on the bottom), so $f^{(1)} \approx 0$, $f^{(2)} \approx 0$, $f^{(3)} \approx 1$ and we have

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \approx -0.5 < 0,$$

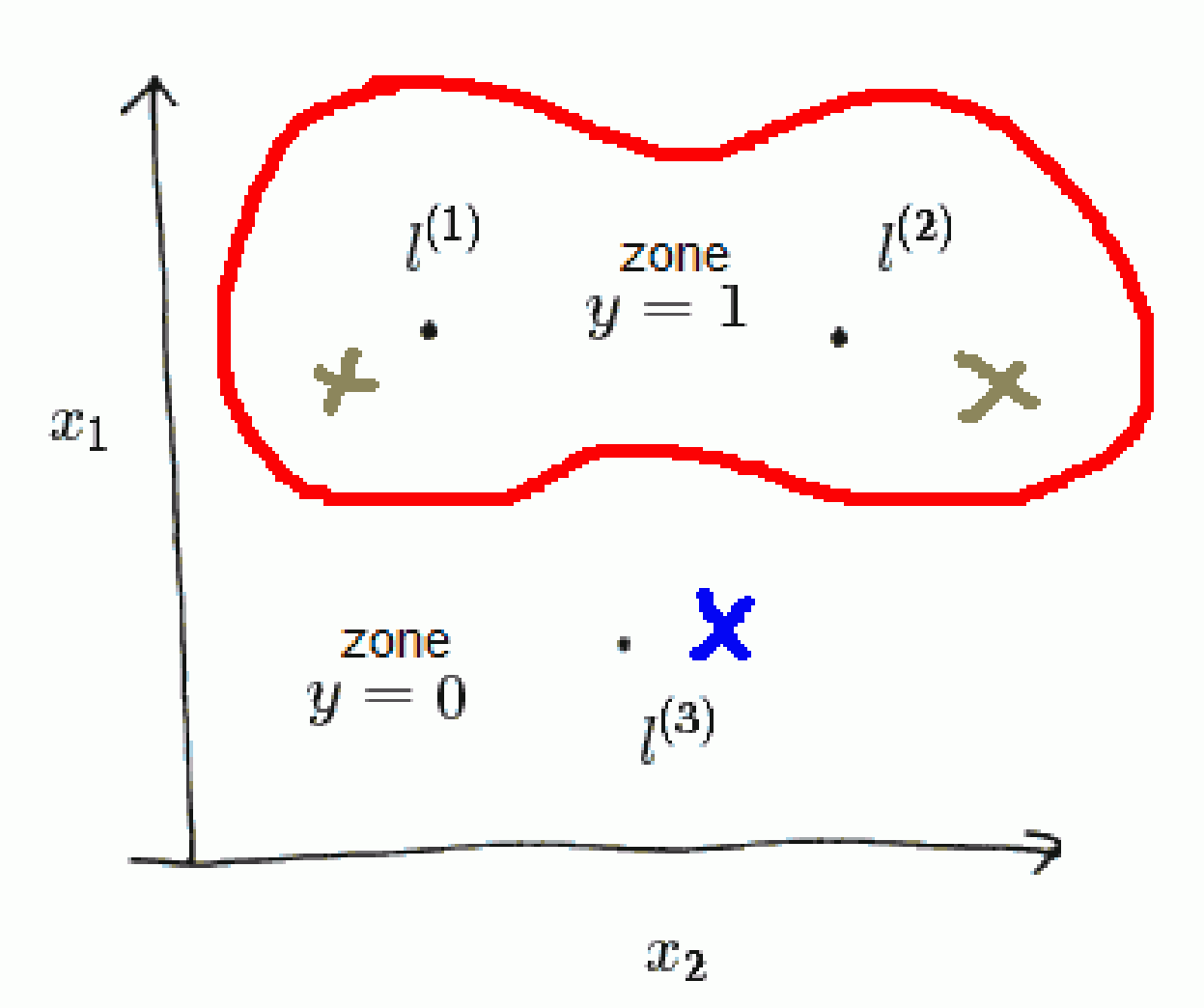
so we predict $y = 0$.

□

Exercise 11. Can we apply the kernel trick to logistic regression? Why is it not used in practice then?

Proof. Yes, we can apply the kernel trick to logisitic regression and the classification performance is almost identical in both cases. KLR (**Kernal Logistic Regression**) can provide class probabilities whereas SVM is a deterministic classifier. KLR:

$$P(\mathbf{y} = y \mid \mathbf{x}; \boldsymbol{\theta}), \quad \text{where } \boldsymbol{\theta} = \sum_i \alpha_i \phi(\boldsymbol{x}^{(i)}) = \sum_i \alpha_i \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{x}^{(i)}) = \sum_i \alpha_i k(\boldsymbol{x}, \boldsymbol{x}^{(i)})$$



$$P(\mathbf{y} = y; \sigma(\sum_i \alpha_i k(\boldsymbol{x}, \boldsymbol{x}^{(i)})) = (\sigma(\sum_i \alpha_i k(\boldsymbol{x}, \boldsymbol{x}^{(i)}))^y (1 - \sigma(\sum_i \alpha_i k(\boldsymbol{x}, \boldsymbol{x}^{(i)})))^{1-y} =$$
$$\Big(\frac{1}{1 + e^{-\sum_i \alpha_i k(\boldsymbol{x}, \boldsymbol{x}^{(i)})}}\Big)^y \Big(1 - \frac{1}{1 + e^{-\sum_i \alpha_i k(\boldsymbol{x}, \boldsymbol{x}^{(i)})}}\Big)^{1-y}.$$

KLR has a natural extension to multi-class classification whereas in SVM, there are multiple ways to extend it to multi-class classification (and it is still an area of research whether there is a version which has provably superior qualities over the others). Surprisingly or unsurprisingly, KLR also has optimal margin properties that the SVMs enjoy (well in the limit at least)!

Note. Although it almost feels like kernel logistic regression is what you should be using, there are certain advantages that SVMs enjoy:

1) KLR is computationally more expensive than SVM—$O(N^3)$ vs $O(N^2 k)$ where k is the number of support vectors.

2) The classifier in SVM is designed such that it is defined only in terms of the support vectors, whereas in KLR, the classifier is defined over all the points and not just the support

vectors. This allows SVMs to enjoy some natural speed-ups (in terms of efficient code-writing) that is hard to achieve for KLR.

□

Exercise 12. What is the difference between logistic regression and SVM without a kernel?

Proof. Only in implementation. One is much more efficient and has good optimization packages. □

Exercise 13. How does the SVM parameter C affect the bias/variance trade off?

Proof. Remember $C = \frac{1}{\lambda}$, so if λ increases there is a higher bias and lower variance. If λ decreases, there is a lower bias, but higher variance. □

Exercise 14. How does the SVM kernel parameter σ^2 affect the bias/variance trade off?

Proof. For the SVM kernel parameter σ^2—sometimes referred to as $\gamma = 2\sigma^2$, as σ^2 increases the features (f_i) vary more smoothly $\rightarrow$ higher bias, lower variance. However, as σ^2 decreases the features (f_i) vary less smoothly $\rightarrow$ lower bias, higher variance.

TL;DR. Another way to think about it is that the gamma parameter in SVM tuning signifies the influence of points either near or far away from the hyperplane. For a low gamma, the model will be too constrained and include all points of the training dataset, without really capturing the shape. For a higher gamma, the model will capture the shape of the dataset well.

□

Exercise 15. Can any similarity function be used for SVM?

Proof. No, the similarity function has to satify's Mercer's theorem.

Mercer's Theorem:

A symmetric function $K(x, y)$ can be expressed as an inner product $\langle \phi(x), \phi(y) \rangle$ for some ϕ if and only if $K(x, y)$ is positive semi-definite

$$\int K(x,y)g(x)g(y)dxdy \geq 0, \quad \forall g$$

or, equivalently: $\begin{bmatrix} K(x^{(1)}, x^{(1)}) & K(x^{(1)}, x^{(2)}) & ... \\ K(x^{(2)}, x^{(1)}) & ... & ... \\ ... & & \end{bmatrix}$ is positive semidefinite for any collection $\{x^{(1)}, ..., x^{(n)}\}$.

TL;DR. In order for a similarity function to be a valid (Mercer) kernel, it is necessary and sufficient that for any $\{x^{(1)}, ..., x^{(m)}\}, (m < \infty)$, the corresponding kernel matrix is symmetric positive semi-definite. This is because it (Mercer's Theorem) makes sure they run correctly and do not diverge (to support optimization).

□

Exercise 16. Logistic regression vs. SVMs: When to use which one?

Proof. Suppose we have n number of features ($x \in \mathbb{R}^{n+1}$) and m training samples.

Note. Defined an extra feature ($f_0 = 1$) as an intercept term. Hence, we shall have $n + 1$ features.

- If n is large (relative to m):
  - e.g. $n = 10,000$, $m \in [10, 1000]$
  - use Logisitc Regression
  - or SVM without a kernel (linear kernel)
- If n is small and m is intermediate:
  - e.g. $n \in [1, 1000]$, $m \in [10, 10000]$
  - use SVM with Gaussian kernel
- If n is small and m is large:
  - e.g. $n \in [1, 1000]$, $m \geq 50000$
  - SVM is too slow for that, so you will need to create/ add more features
  - so use Logistic Regression or SVM without a kernel hat

Note. The key thing is that if there is a huge number of training examples, a Gaussian kernel takes a long time The optimization problem of an SVM is a convex problem, so you will always find the global minimum. However a Neural Network may work well for all these settings, but it might be slower to train and it is non-convex, so you may find local optima instead of a global minima.

□

Exercise 17. Give some situations where you will use an SVM over a RandomForest Machine Learning algorithm and vice-versa?

Proof. The performance depends on many factors

- the number of training instances
- the distribution of the data
- linear vs. non-linear problems
- input scale of the features
- the chosen hyperparameters
- how you validate/evaluate your model

In general, it is easier to train a well-performing Random Forest classifier since you have to worry less about hyperparameter optimization. Due to the nature Random Forests, you are less likely to overfit. You simply grow n trees on n bootstrap samples of the training set on feature subspaces—using the majority vote, the estimate will be pretty robust.

Using Support Vector Machines, you have "more things" to "worry" about such as choosing an appropriate kernel (poly, RBF, linear ...), the regularization penalty, the regularization strength, kernel parameters such as the poly degree or gamma, and so forth.

TL;DR. So, in short, Random Forests are much more automated and thus "easier" to train compared to SVMs, but there are many examples in literature where SVMs outperform Random Forests and vice versa on different datasets. So, if you like to compare these two, make sure that you run a large enough grid search for the SVM and use nested cross-validation to reduce the performance estimation bias.

□

2 Extra

Exercise 18. Hard-margin SVM vs Soft-margin SVM? What's the difference?

Proof. **Hard-margin SVM** is used when the data is linearly separable. Say we have one class (label 1) and other class (label -1), then our decision boundary can be described by the following:

$$y^{(i)} = \left\{ \begin{array}{ll} 1 & \boldsymbol{\theta}^\top \boldsymbol{x} - b \geq 1 \\ \text{-1} & \boldsymbol{\theta}^\top \boldsymbol{x} - b \leq -1 \end{array} \right\},$$

which can be rewritten as $y^{(i)}(\boldsymbol{\theta}^\top \boldsymbol{x} - b) \geq 1$, $\forall\, 1 \leq i \leq m$ (for a given training set m). We can put this together to get the optimization problem:

"Minimize $\|\boldsymbol{\theta}\|$ subject to $y^{(i)}(\boldsymbol{\theta}^\top \boldsymbol{x} - b) \geq 1$ for $i = 1, \ldots, m$." The $\boldsymbol{\theta}$ and b that solve this problem determine our classifier, $\boldsymbol{x} \mapsto \sigma(\boldsymbol{\theta} \cdot \boldsymbol{x} - b)$.

Note.

$$\min ||\boldsymbol{\theta}||$$
$$\text{subject to } y^{(i)}(\boldsymbol{\theta}^\top \boldsymbol{x} - b) \geq 1$$

is equivalent to the following minimization problem:

$$\min \frac{1}{2}||\boldsymbol{\theta}||^2$$

$$\text{subject to } y^{(i)}(\boldsymbol{\theta}^\top \boldsymbol{x} - b) \geq 1$$

The above statement is the SVM optimization problem. It is called a convex quadratic optimization problem, which will help us find the global minima.

Now, as we determined sometimes our data is not linearly separable. We introduced the idea of the kernel trick which allows us to map our features into a higher dimensional space where it can be linearly separable. However, there was an important that needs to be asked before using the kernel trick: **why is your data non-linearly separable?**

If your data contains some noise, then it might be non-linearly separable. So we shall employ the **Soft Margin SVM**. The Soft Margin SVM adds slack variables ζ_i to the constraints of the optimization problem. The constraints now become:

$$y^{(i)}(\boldsymbol{\theta}^\top \boldsymbol{x} - b) \geq 1 - \zeta_i, \text{ for } i = 1, \ldots, m$$

Solving for ζ_i we get,

$$\zeta_i \geq 1 - y^{(i)}(\boldsymbol{\theta}^\top \boldsymbol{x} - b), \text{ for } i = 1, \ldots, m.$$

By adding the slack variables, when minimizing the objective function, it is possible to satisfy the constraint even if the example does not meet the original constraint. The problem is we can always choose a large enough value of ζ so that all the examples will satisfy the constraints.

One technique to handle this is to use regularization. For example, we could use L^1 regularization to penalize large values of ζ. The regularized optimization problem becomes:

$$\min_{\theta, b, \zeta} \frac{1}{2}||\boldsymbol{\theta}||^2 + \sum_{i=1}^{m} \zeta_i$$

$$\text{subject to } y^{(i)}(\boldsymbol{\theta}^\top \boldsymbol{x} - b) \geq 1 - \zeta_i, i = 1...m$$

Also, we want to make sure that we do not minimize the objective function by choosing negative values of ζ. We add the constraints $\zeta_i \geq 0 \rightarrow 1 - y^{(i)}(\boldsymbol{\theta}^\top \boldsymbol{x} - b) \geq 0$. Now we can rewrite this expression as in lieu of ζ:

$$\max\left(0, 1 - y^{(i)}(\boldsymbol{\theta}^\top \boldsymbol{x} - b)\right).$$

We will put this this back into our optimization problem, multiply the summation by $\frac{1}{m}$, and add multiply our L^2 regularization term by λ:

$$\arg\min_{\theta} \Big[\frac{1}{m} \sum_{i=1}^{m} \max\left(0, 1 - y^{(i)}(\boldsymbol{\theta}^\top \boldsymbol{x} - b)\right) + \frac{\lambda}{2}||\boldsymbol{\theta}||^2\Big].$$

The λ determines the trade-off between increasing the margin size and ensuring that the $\boldsymbol{x}^{(i)}$ lie on the correct side of the margin. Thus, for sufficiently small values of λ, the second term in the loss function will become negligible, hence, it will behave similar to the hard-margin SVM, if the input data are linearly classifiable, but will still learn if a classification rule is viable or not.

If the data is non-linearly seperable and it is not noisy, then you need to employ the kernel trick.

Note. $\max\left(0, 1 - y^{(i)}(\boldsymbol{\theta}^\top \boldsymbol{x} - b)\right)$ is called the **hinge loss** function.

□

Exercise 19. What is the difference between logistic regression and SVM?

Proof. Logistic regression assumes that the predictors are not sufficient to determine the response variable, but determine a probability that is a logistic function of a linear combination of them. If there is a lot of noise in the data, logistic regression (usually fit with maximum-likelihood techniques) is a great technique.

On the other hand, there are problems where you have thousands of dimensions and the predictors do nearly-certainly determine the response, but in some hard-to-explicitly-program way. An example would be image recognition. If you have a grayscale image, 100 by 100 pixels, you have 10,000 dimensions already. With various basis transforms (kernel trick) you will be able to get a linear separator of the data.

Non-regularized logistic regression techniques do not work well (in fact, the fitted coefficients diverge) when there is a separating hyperplane, because the maximum likelihood is achieved by any separating plane, and there's no guarantee that you will get the best one. What you get is an extremely confident model with poor predictive power near the margin.

SVMs get you the best separating hyperplane, and they are efficient in high dimensional spaces. They are similar to regularization in terms of trying to find the lowest-normed vector that separates the data, but with a margin condition that favors choosing a good hyperplane. A hard-margin SVM will find a hyperplane that separates all the data (if one exists) and fail if there is none; soft-margin SVMs (generally preferred) do better when there's noise in the data.

Additionally, SVMs only consider points near the margin (support vectors). Logistic regression considers all the points in the data set. Which you prefer depends on your problem.

Logistic regression is great in a low number of dimensions and when the predictors do not suffice to give more than a probabilistic estimate of the response. SVMs do better when there's a higher number of dimensions, and especially on problems where the predictors do certainly (or near-certainly) determine the responses. □