Tim Book, Adi Bronshtein

Support Vector Machines (SVMs)

GENERAL ASSEMBLY

OK, let's get my (1st) dad joke out of the way

Agenda

- Support vector machines, a play in three acts:
  - The maximal margin SVM
  - The soft margin SVM
  - The kernel SVM
- The “Kernel Trick”
- SVMs in Scikit-Learn

What are SVMs?

Support vector machines (SVMs) are **classification models**, that is, they predict categorical variables. They belong to a wider class of models called **discriminant models**.

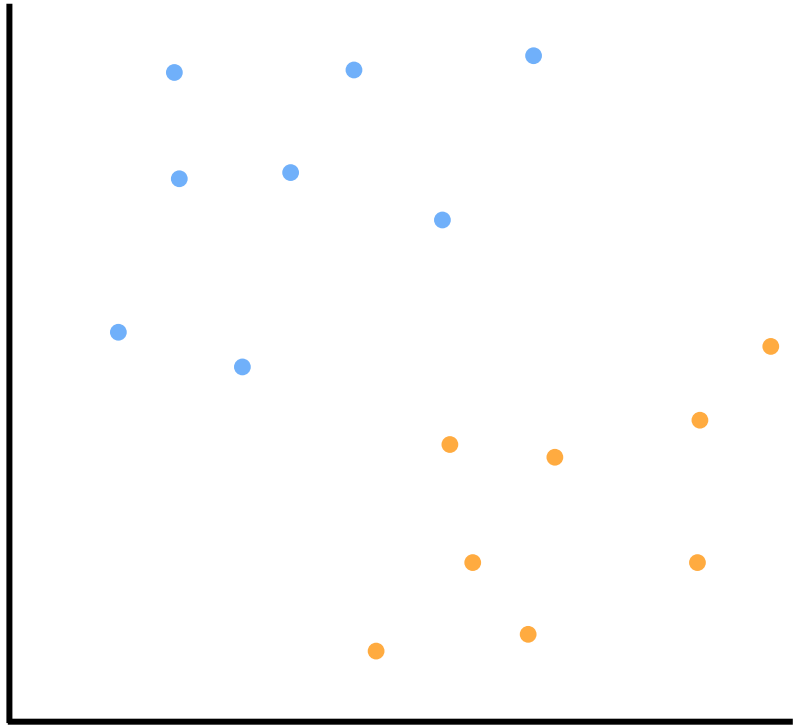
What are SVMs?

Support vector machines (SVMs) are **classification models***, that is, they predict categorical variables. They belong to a wider class of models called **discriminant models**.

*Actually, there are such things as support vector regressors, but they're very rarely used. Today we'll only focus on **support vector classifiers**.

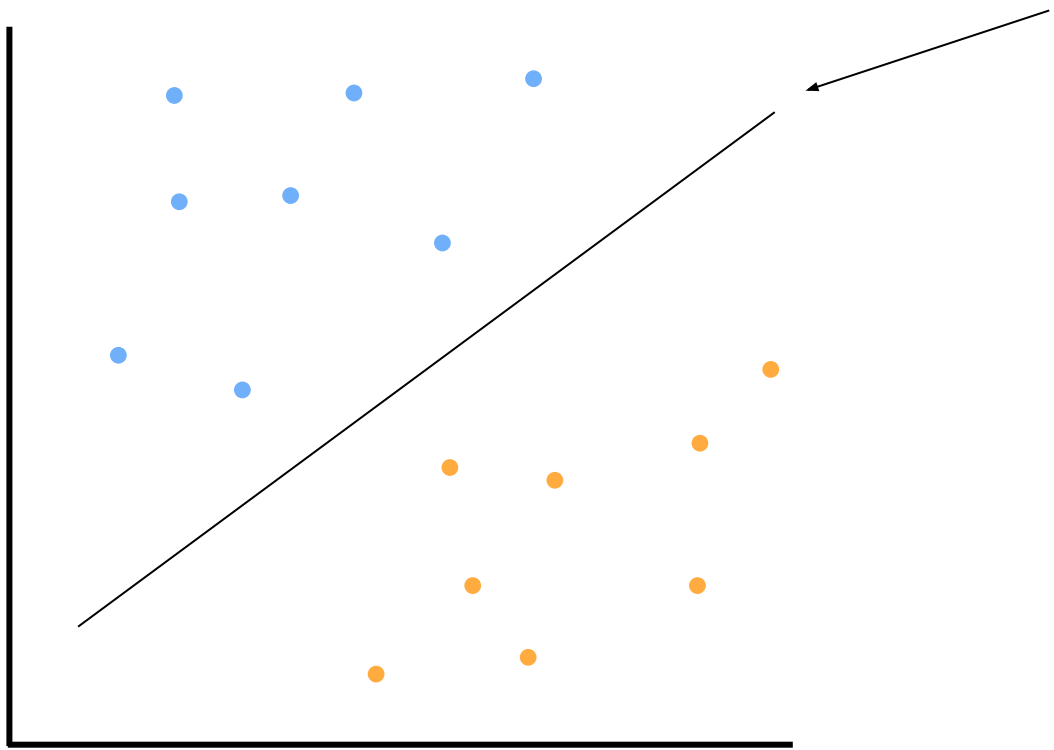
Maximal Margin SVMs

SVM Part I:



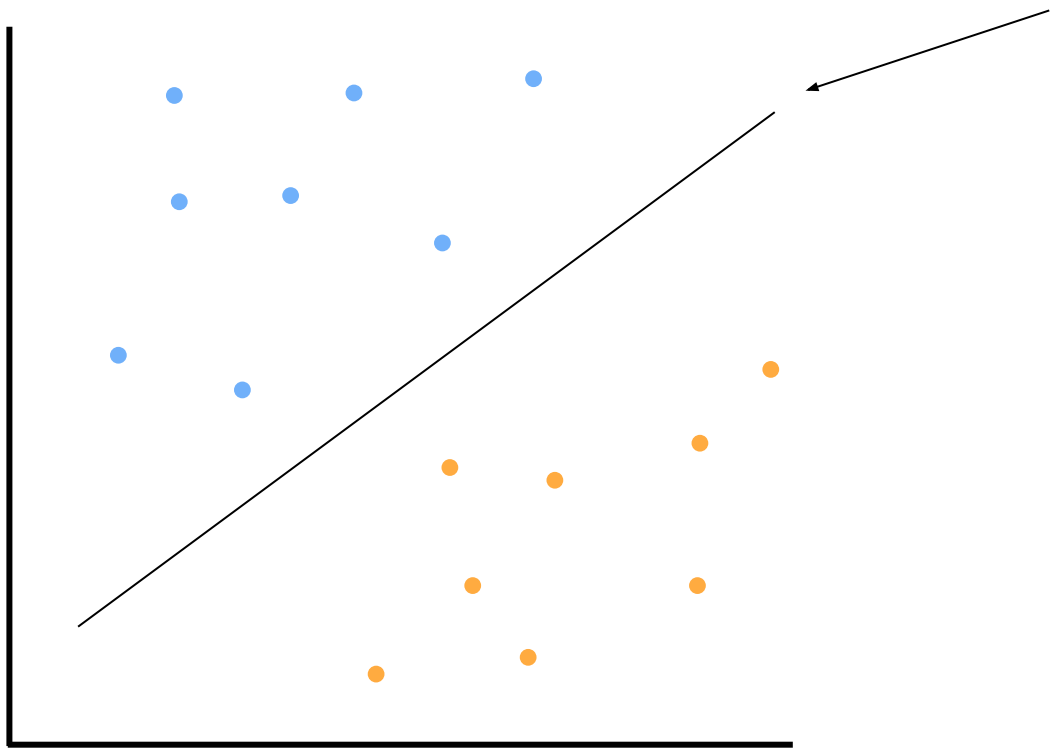
What's the easiest way to classify these two categories?

SVM Part I:



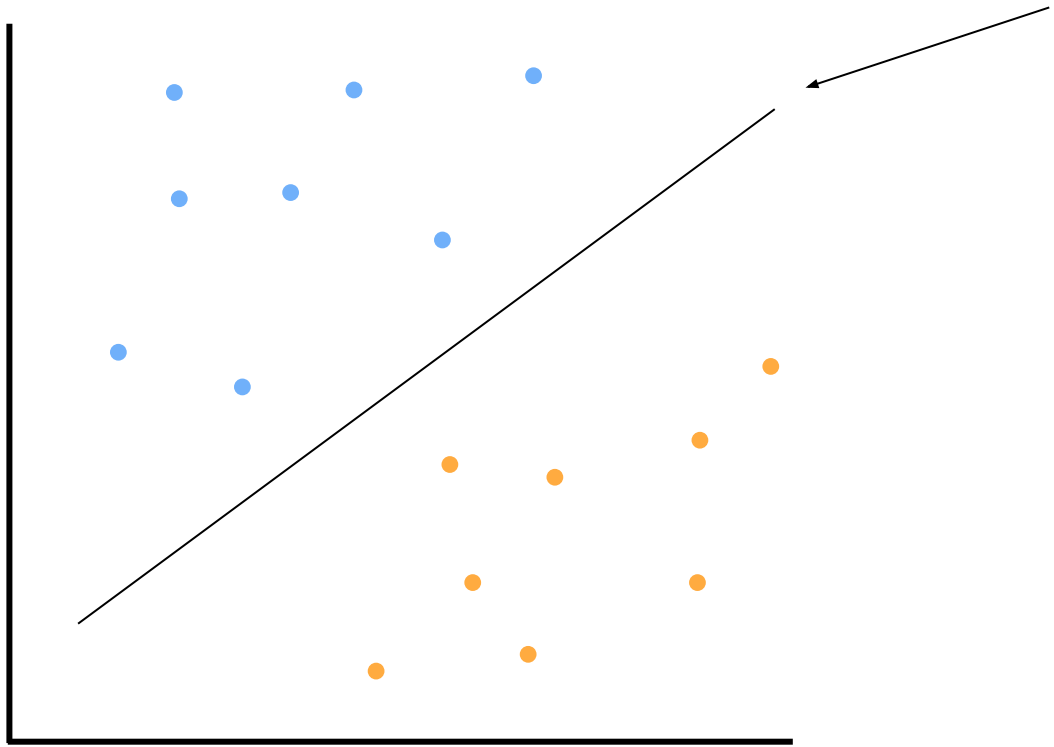
This is the **separating hyperplane**.

SVM Part I:



This is the **separating hyperplane**, or sometimes the **linear discriminant**, or even the **decision boundary**.

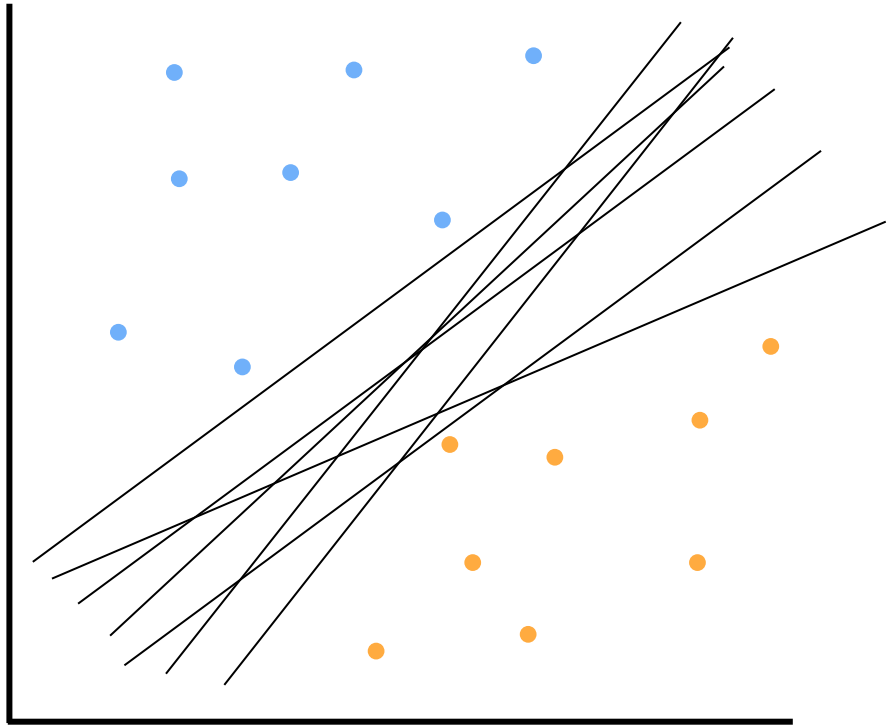
SVM Part I:



This is the separating hyperplane, or sometimes the linear discriminant, or even the decision boundary.

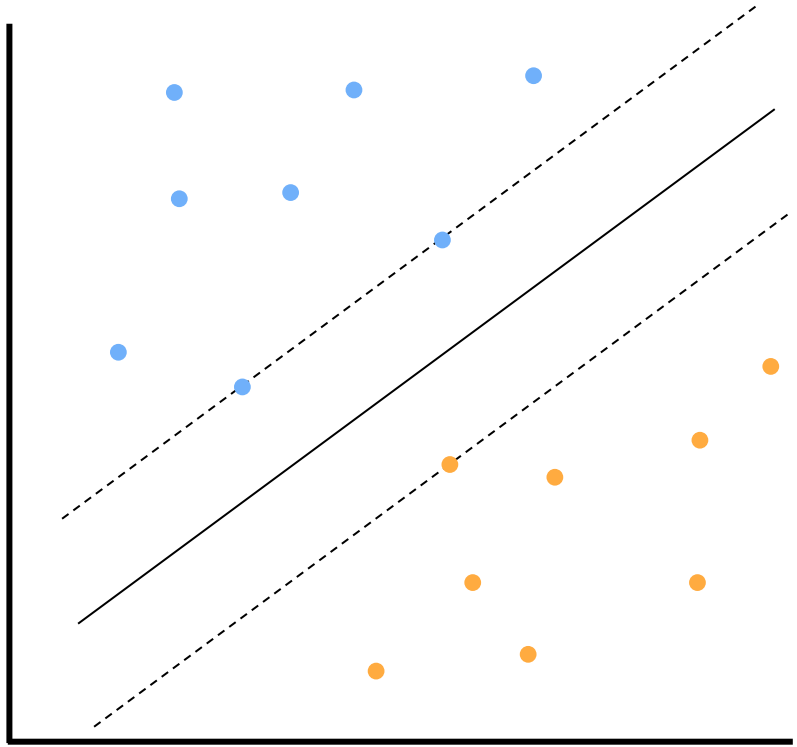
IT IS *NOT* THE SUPPORT VECTOR!

SVM Part I:



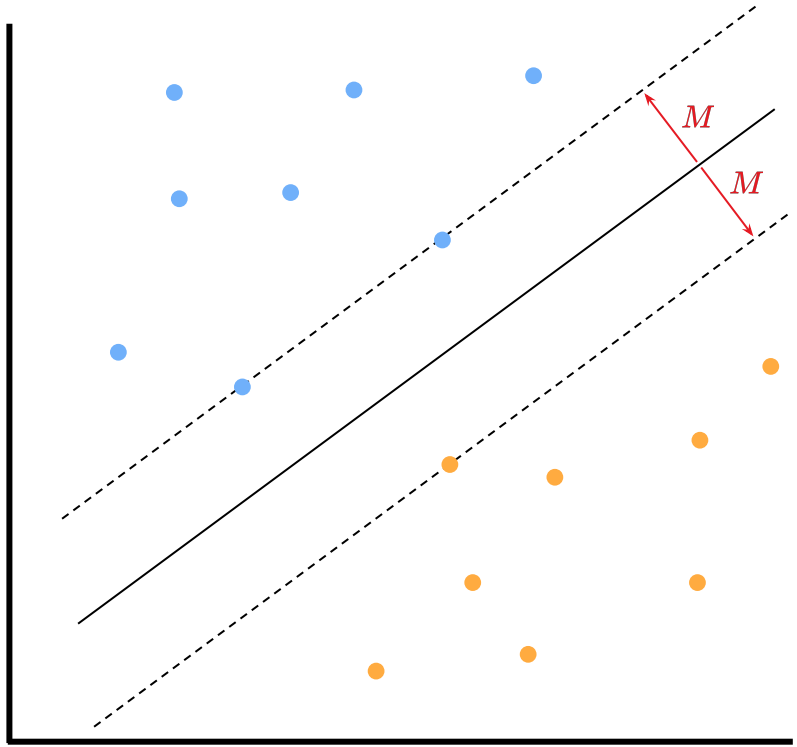
But how is the separating hyperplane decided? All of the lines shown do the same thing, right?

SVM Part I: The Maximal Margin SVM



The “best” separating hyperplane is the one that creates the **maximal margin**. That is, it has the widest space around it with no points inside.

SVM Part I: The Maximal Margin SVM



The “best” separating hyperplane is the one that creates the **maximal margin**. That is, it has the widest space around it with no points inside.

Quick Aside...

Optimization Problems

Aside: Optimization

(Almost) all machine learning models are fit via **optimizing** some **loss function**. We'll have a whole lesson on the details of this tomorrow, but it's worth talking a little bit about today.

Optimization problems have two components: the **objective function**, and the **constraints**. A problem with no constraints is said to be **unconstrained**.

We've already seen several optimization problems:

Optimization Example I: OLS

$$\text{minimize} \quad \frac{1}{n}\sum(y_i - \hat{y}_i)^2$$

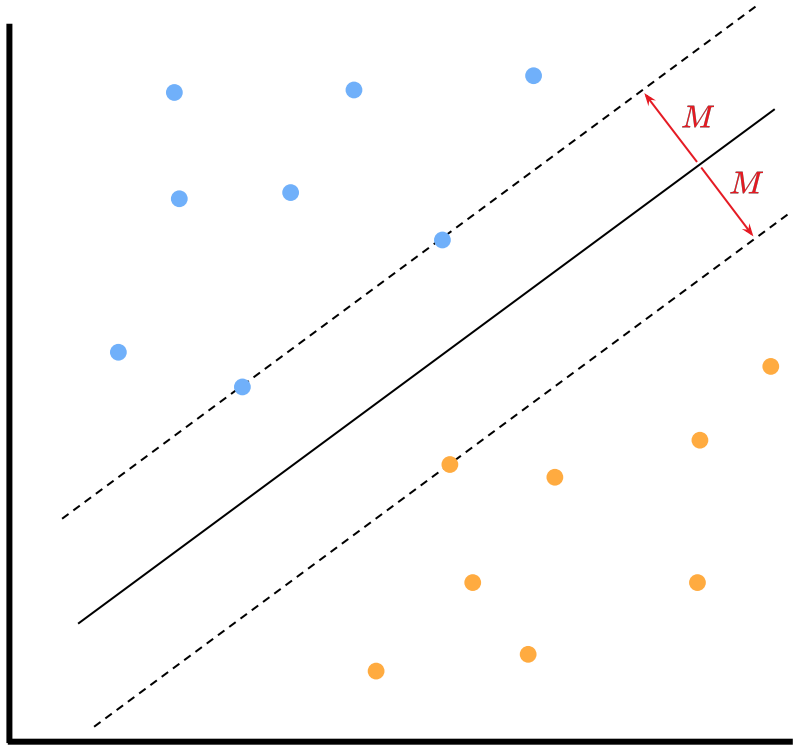
Optimization Example II: LASSO (Lagrangian form)

$$\text{minimize} \quad \frac{1}{n}\sum(y_i - \hat{y}_i)^2 + \lambda \sum |\beta_j|$$

Optimization Example II: LASSO (constrained form)

$$\begin{aligned} \text{minimize} \quad & \frac{1}{n}\sum(y_i - \hat{y}_i)^2 \\ \text{such that} \quad & \sum|\beta_j| \leq t \end{aligned}$$

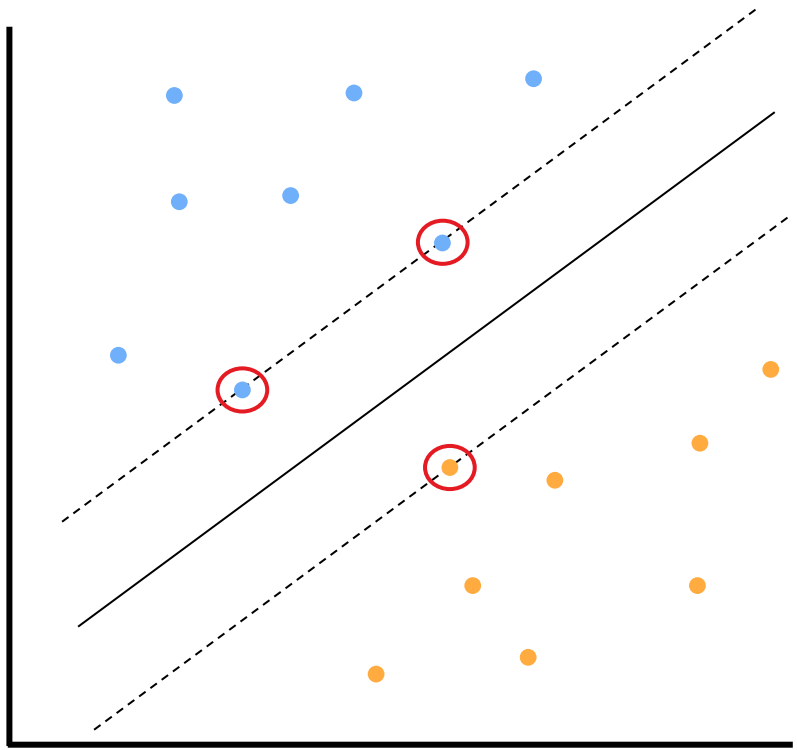
SVM Part I: The Maximal Margin SVM



Our maximal margin SVM is fit via the following optimization problem:

maximize	M
such that	all points on correct side of margin

SVM Part I: The Maximal Margin SVM



The points touching the margin exactly are known as the **support vectors**.

Even though they're what the whole model is named after, they're not too relevant for us.

They're important for the inner mathematics of how SVMs are fit, which we thankfully are not getting into.

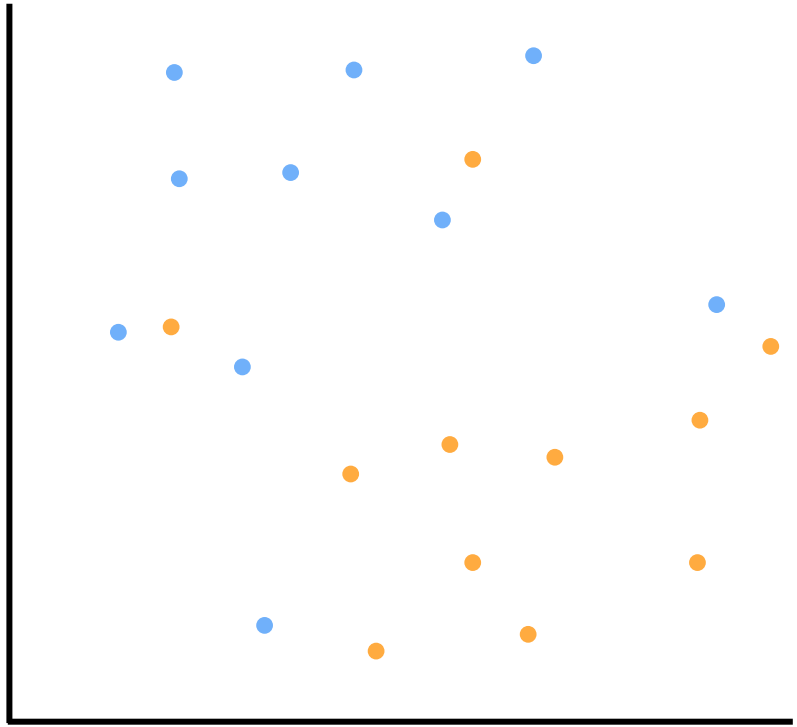
Soft Margin SVMs

SVM Part II: Soft Margin SVMs

As you may have guessed, the situation in Part I is unreasonable. We never have **separable** data in real life.

So let's extend this idea to a less-than-perfect data set:

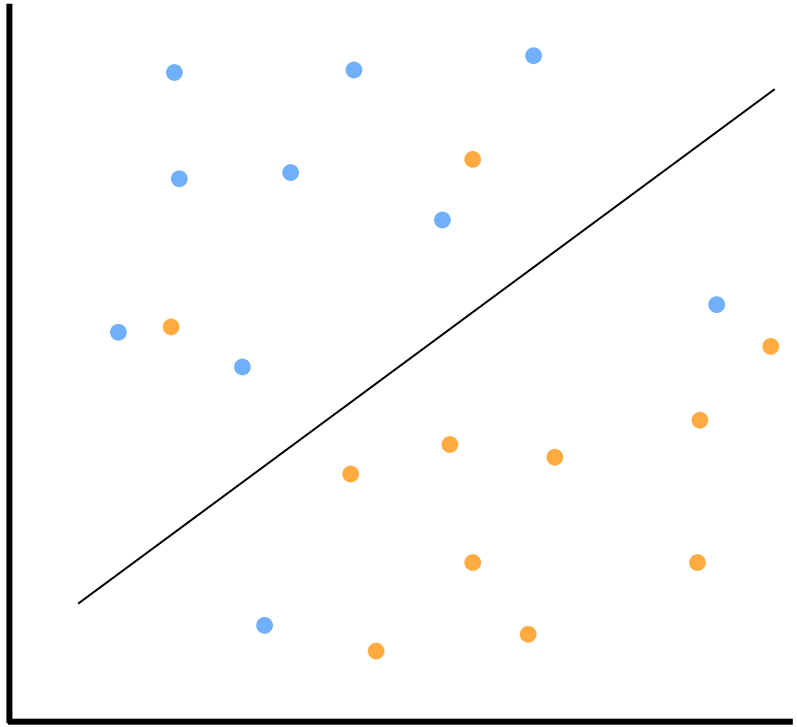
SVM Part II: The Soft Margin SVM



Things are **mostly** ok here. The data are still mostly separated, but there are a few points that are on the "wrong" side of where they should be.

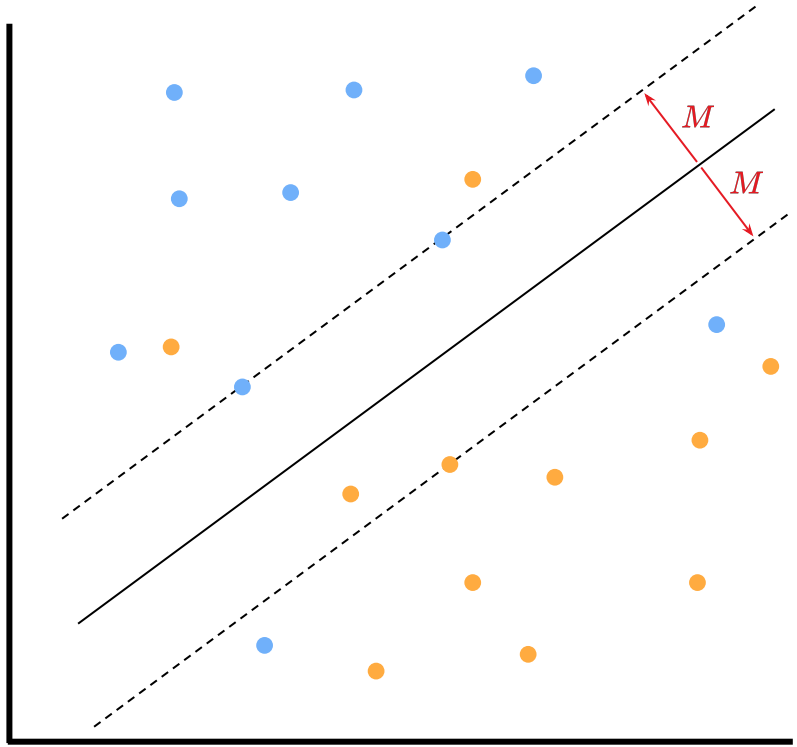
How can we still making a separating hyperplane while also **cutting the model some slack?**

SVM Part II: The Soft Margin SVM



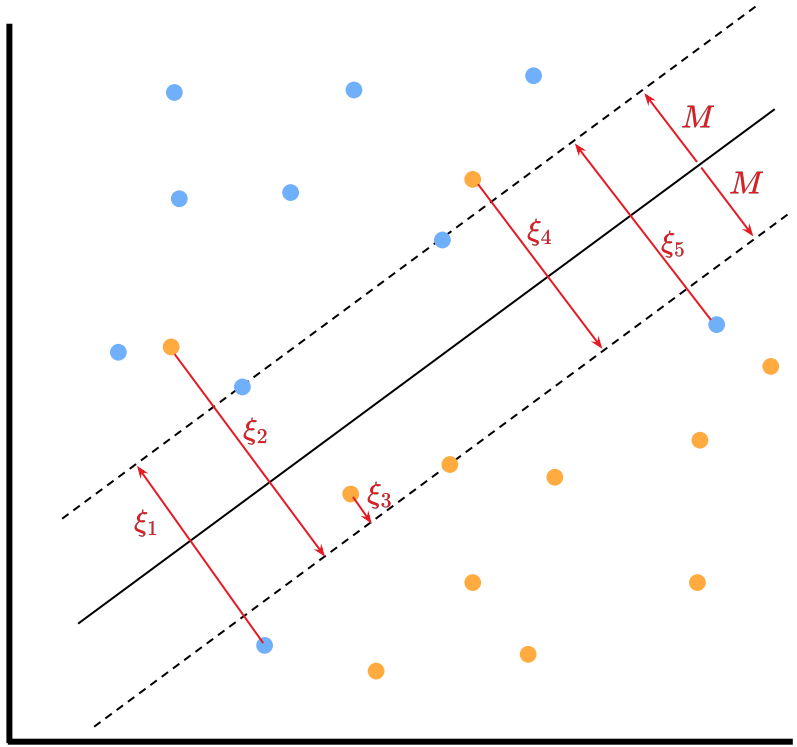
We’re still going to create a separating hyperplane.

SVM Part II: The Soft Margin SVM



We're still going to create a supporting hyperplane. And a pair of margins.

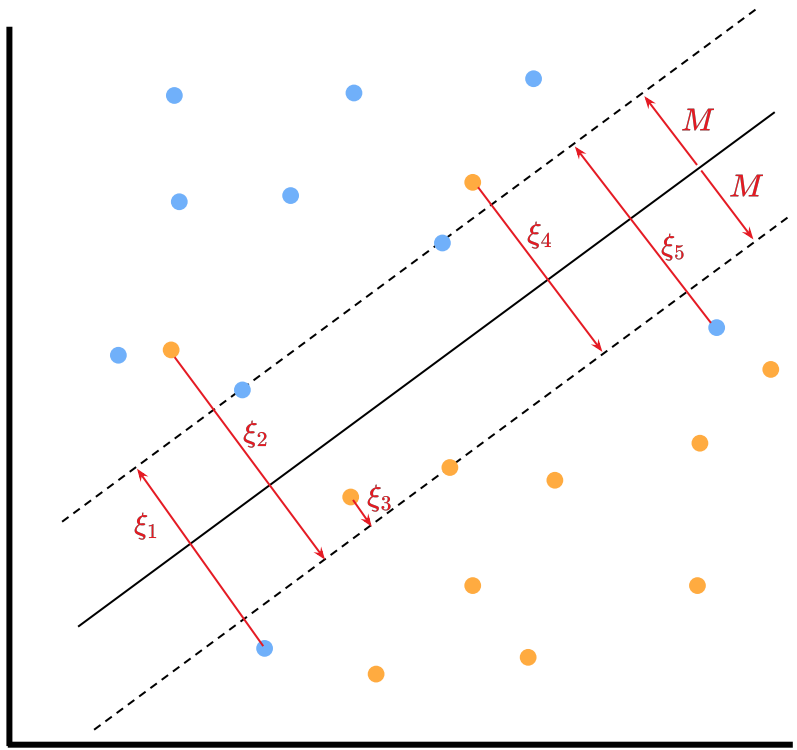
SVM Part II: The Soft Margin SVM



We're still going to create a supporting hyperplane. And a pair of margins.

Next, we measure the distance from each "wrong" point to its appropriate margin. We'll denote these **slack variables** as ξ_i.

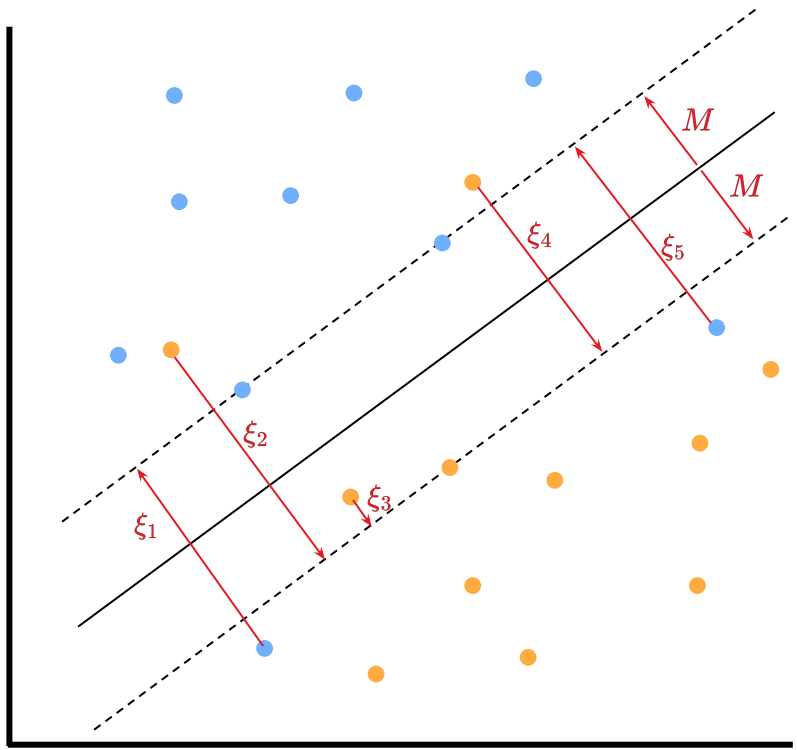
SVM Part II: The Soft Margin SVM



So we'll use these new slack variables to create a new optimization problem for our SVM:

maximize	M
such that	$\sum \xi_i \leq const$
	and more stuff

SVM Part II: The Soft Margin SVM



So we'll use these new slack variables to create a new optimization problem for our SVM:

$$\begin{aligned} &\text{maximize} && M \\ &\text{such that} && \sum \xi_i \leq const \\ & && \text{and more stuff} \end{aligned}$$

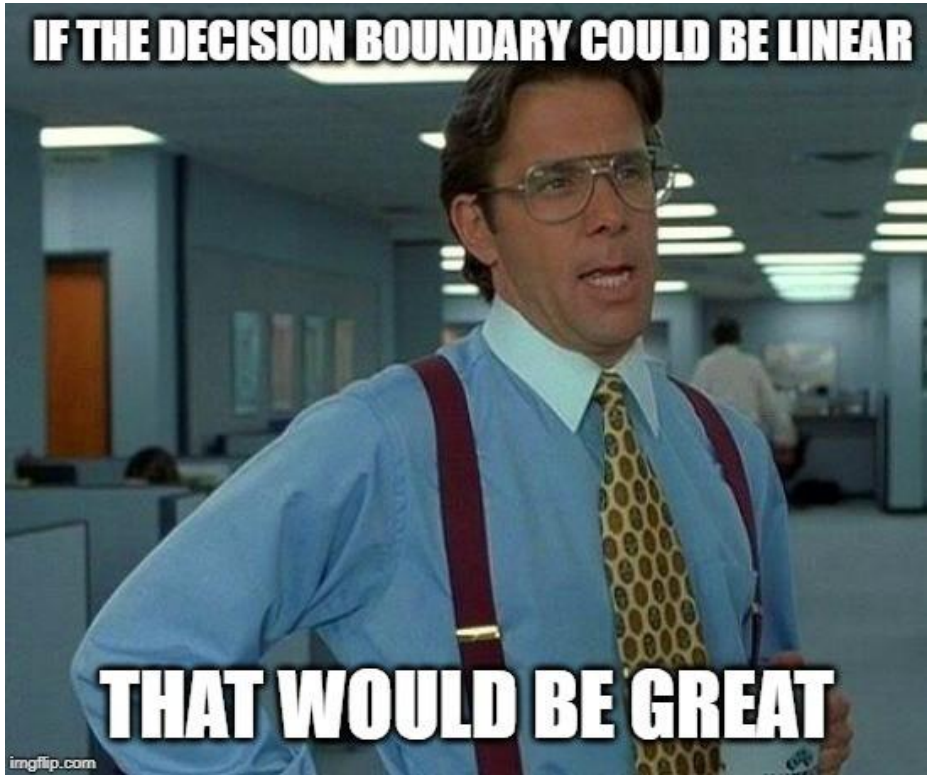
This value is a combination of a tuning parameter and theoretical considerations regarding the optimization problem.

It doesn't make intuitive sense right now, but it will in Part III.

Let's try it out!

Kernel SVMs

Unfortunately, that is not always the case!



“Linear” SVMs are not enough

What if the best separating hyperplane isn’t linear? What if you have one of these situations:

How to bend an SVM

There are two ways we can add curvature to our model:

1. Add polynomial features - for example, add square, cubic, etc. terms to our model.

We *won't* be doing this for a variety of reasons. A major reason is that this process is discrete - you can only add one term at a time. A better idea is to...

How to bend an SVM

2. **Kernelize our model**. Instead of adding curved terms to our model, we can actually curve the very axes our model is fit upon!

Sounds trippy. Let's talk about it.

Quick Aside...

The “Kernel Trick”

What is a Kernel?

Simply put, a **kernel** is any function that takes two vectors and returns one scalar.

For example, the **dot product** qualifies as a kernel:

$$K(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T \mathbf{y}$$

This one is known as the **linear kernel**. But there are many other more interesting ones.

The Kernel Trick

The kernel trick says, any time you see a dot product between two vectors, replace it with the kernel operation on those two vectors. Essentially:

$$\mathbf{x}_i^T \mathbf{x}_j \longmapsto K(\mathbf{x}_i, \mathbf{x}_j)$$

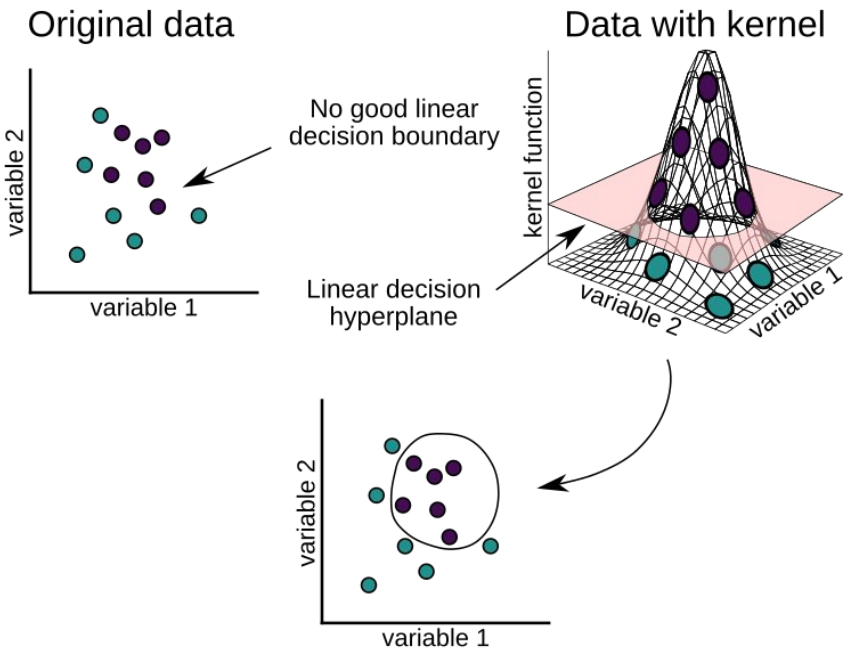
The kernel trick can be applied to many machine learning models where dot products show up. They are most commonly talked about with SVMs, though.

The Kernel Trick

The idea is to pick a kernel that **increases the dimensionality of your data**. That is, a kernel that can take something linear, and make it curve. Here are three popular choices that exist in Scikit-Learn:

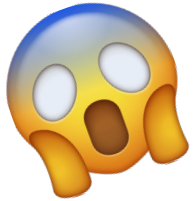
Kernel Name	Formula	Notes
Linear Kernel	$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$	This kernel does nothing. The same as not kernelizing at all.
Polynomial Kernel	$K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^d$	Can adjust power *d* to add more curvature.
Radial Basis Function Kernel	$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \lVert \mathbf{x}_i - \mathbf{x}_j \rVert^2)$	The most adjustable and therefore most common. Start with this!

Geometrically...



https://machinelearningwithmlr.files.wordpress.com/2019/10/ch06_fig_5_mlr.png?w=750

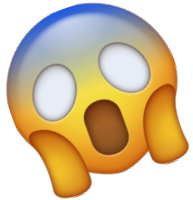
For the mathematicians: Why does this work?



In order for the kernel trick to work, the kernel you pick has to be a **Mercer kernel**. To be a Mercer kernel, there must be a function *h* that satisfies:

$$K(\mathbf{x}_i, \mathbf{x}_j) = h(\mathbf{x}_i)^T h(\mathbf{x}_j) \geq 0$$

For the mathematicians: Why does this work?



For example, if we have *n* features, the 2nd degree polynomial kernel is:

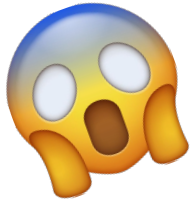
$$K(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x}^T \mathbf{y})^2 = \sum_{i=1}^{n} (x_i^2)(y_i^2) + \sum_{i=2}^{n} \sum_{j=1}^{i-1} (\sqrt{2} x_i x_j)(\sqrt{2} y_i y_j) + \sum_{i=1}^{n} (\sqrt{2} x_i)(\sqrt{2} y_i) + 1$$

So:

$$h(\mathbf{x}) = (x_n^2, \ldots, x_1^2, \sqrt{2} x_n x_{n-1}, \ldots, \sqrt{2} x_2 x_1, \sqrt{2} x_n, \ldots, \sqrt{2} x_1, 1)$$

Notice the original vector has length *n*, where the kernelized vector is length *n*(*n*+1)/2! We've dramatically **increased the dimensionality!**

For the mathematicians: Why does this work?



Ok, how about the **radial basis function (RBF) kernel**?

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$$

Does such an *h* exist? Yes, actually! It just happens to be **infinitely dimensional!**

We just discovered that our model is going to literally have **infinity *x*-variables**. What is the only tool that can help us from overfitting our model?

Let’s try it out!