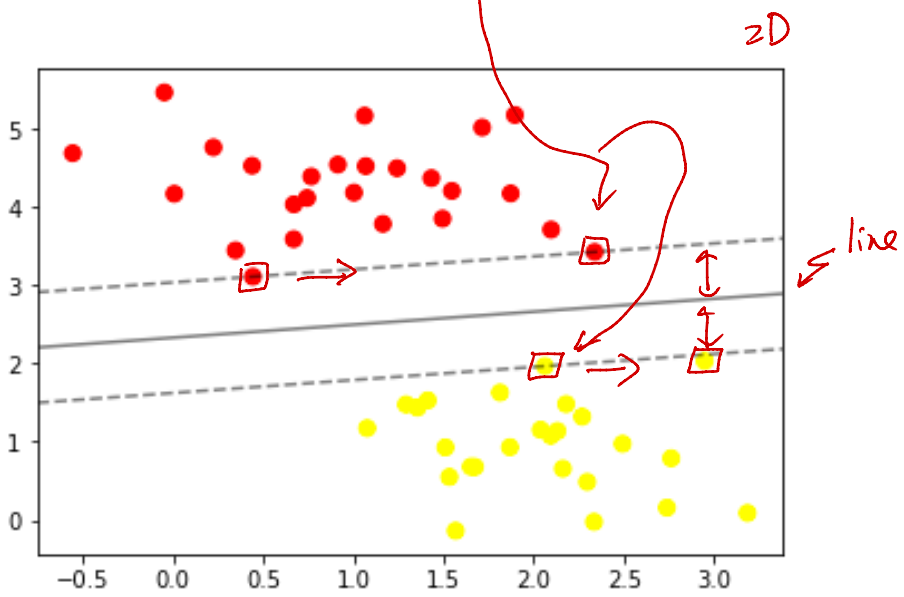
Support Vector Machine (SVM)

In this section, we will discuss the support vector machine (SVM), an approach for classification that was developed in the computer science community in the 1990s and that has grown in popularity since then. Support vector machines are a particularly powerful and flexible class of supervised algorithms for **both classification and regression**.

1. Rationale for Maximum Margin

In support vector machines, **the hyperplane that maximizes this margin is the one we will choose as the optimal model**. In two dimensions, this is **the dividing line that maximizes the margin between the two sets of points**. Notice that a few of the training points just touch the margin. **These points are the pivotal elements of this fit, and are known as the support vectors, and give the algorithm its name**.



Hyperplanes with large margins tend to have better generalization performance than those with small margins. **Intuitively, if the margin is small, then any slight perturbation in the hyperplane or the training instances located at the boundary can have quite an impact on the classification performance.** Small margin hyperplanes are thus more susceptible to **overfitting**, as they are barely able to separate the classes with a very narrow room to allow perturbations. On the other hand, a hyperplane that is farther away from training instances of both classes has sufficient leeway to be robust to minor modifications in the data, and thus shows superior generalization performance.

The idea of choosing the maximum margin separating hyperplane also has strong foundations in statistical learning theory. It can be shown that the margin of such a hyperplane is inversely related to

$$d \propto \frac{1}{|w|}$$

the VC-dimension of the classifier, which is a commonly used measure of the complexity of a model. As will be addressed later in the Occam's Razor learning principle, **a simpler model should be preferred over a more complex model if they both show similar training performance.** Hence, maximizing the margin results in the selection of a separating hyperplane with the lowest model complexity, which is expected to show better generalization performance.

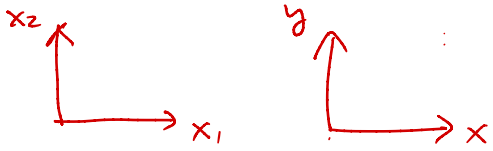
2. Theoretical Minimum: Support Vector Machine[1]

A SVM is a classifier that searches for an **optimal separating hyperplane** with the largest margin. In this section, we define a hyperplane and introduce the concept of a separating hyperplane, margin and an optimal separating hyperplane.

What Is Hyperplane?

In a d-dimensional space, a hyperplane is a flat subspace of dimension $d - 1$. For instance, in two dimensions, a hyperplane is a flat one-dimensional subspace—in other words, a line. In three dimensions, a hyperplane is a flat two-dimensional subspace—that is, a plane. In $d > 3$ dimensions, it can be hard to visualize a hyperplane, but the notion of a $(d-1)$-dimensional flat subspace still applies.

The mathematical definition of a hyperplane is quite simple. In two dimensions, a hyperplane is defined by the equation:



$$w_1 x_1 + w_2 x_2 + b = 0 \quad (1)$$

for parameters w_1, w_2, and b. When we say the equation "defines" the hyperplane, we mean that any point $\mathbf{x} = [x_1, x_2]^T$ for which the equation holds is a point on the hyperplane.

Q: Equation (1) is simply the equation of a line, since indeed in two dimensions a hyperplane is a line. Convert $\mathbf{x}$ into 2D Cartesian coordinate and convince yourself it is the equation of a line.

A:

$$w_1 x + w_2 y + b = 0 \text{ so } y = -\frac{w_1}{w_2} x + b$$

We can easily extend the hyperplane to the d-dimensional setting:

$$w_1 x_1 + w_2 x_2 + \cdots w_d x_d + b = 0$$

[1] Major contents are adapted from dynamic e-chapters of Abu-Mostafa, Y S, Magdon-Ismail, M., Lin, H-T (2012) *Learning from Data*, AMLbook.com.

Or

(2) $\mathbf{w}^T\mathbf{x} + b = 0$

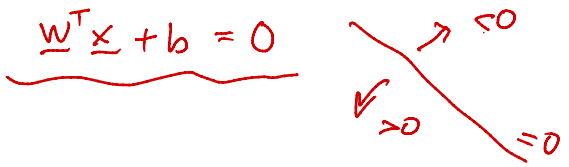
in which $\mathbf{w} = [w_1, w_2, \ldots, w_d]^T$, $\mathbf{x} = [x_1, x_2, \ldots, x_d]^T$; $\mathbf{w}, \mathbf{x} \in \mathbb{R}^d$ where $\mathbb{R}^d$ is the d-dimensional Euclidean space and $b \in \mathbb{R}$.

Separating Hyperplane

Consider a binary classification problem in the d-dimensional Euclidean space. The problem consists of N training examples: $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)$ where $y_n = f(\mathbf{x}_n)$ for $n = 1, \cdots, N$. Again let $\mathbf{x} \in \mathbb{R}^d$ where $\mathbb{R}^d$ is the d-dimensional Euclidean space and by convention let $y \in \{-1, 1\}$ denote its class label.

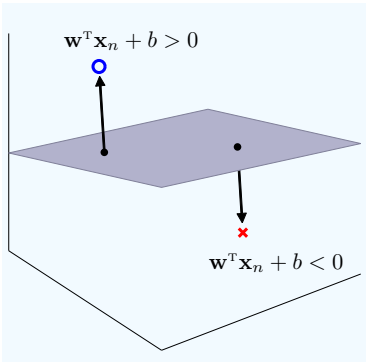
We are interested in finding a hyperplane that places instances of both classes on **opposite** sides of the hyperplane, thus resulting in a **separation** of the two classes. This means that:

$\mathbf{w}^T\mathbf{x}_n + b > 0$ if $y_n = 1$
$\mathbf{w}^T\mathbf{x}_n + b < 0$ if $y_n = -1$



In other words, the hyperplane h, defined by $(b, \mathbf{w})$, separates the training examples if and only if:

(3) $y_n(\mathbf{w}^T\mathbf{x}_n + b) > 0$ for $n = 1, \cdots, N$ separating hyperplane



The signal $y_n(\mathbf{w}^T\mathbf{x}_n + b)$ is positive for each data point. However, the magnitude of the signal is not meaningful since we can make it arbitrarily small or large for the same hyperplane by rescaling $\mathbf{w}$ and b.

positive

Q: Why?

A:

From Equation (2), we observe that $(b, \mathbf{w})$ is the same hyperplane as $(b/\rho, \mathbf{w}/\rho)$ for any $\rho > 0$.

By rescaling the weights, we can control the magnitude of the signal for our data points. Let us pick a particular value of ρ,

$$\rho = \min_{n=1,\cdots,N} y_n(\mathbf{w}^T\mathbf{x}_n + b)$$

which is positive because of Equation (3). Now, rescale the weights to obtain the same hyperplane $(b/\rho, \mathbf{w}/\rho)$. **For these rescaled weights**,

$$\min_{n=1,\cdots,N} y_n\left(\frac{\mathbf{w}^T}{\rho}\mathbf{x}_n + \frac{b}{\rho}\right) = \frac{1}{\rho}\min_{n=1,\cdots,N} y_n(\mathbf{w}^T\mathbf{x}_n + b) = \frac{\rho}{\rho} = 1$$

Thus, for any separating hyperplane, it is always **possible** to choose weights so that all the signals $y_n(\mathbf{w}^T\mathbf{x}_n + b)$ are of magnitude greater than or equal to 1, with equality satisfied by at least one $(\mathbf{x}_n, y_n)$. This motivates our new definition of a separating hyperplane, equivalent to Equation (3):

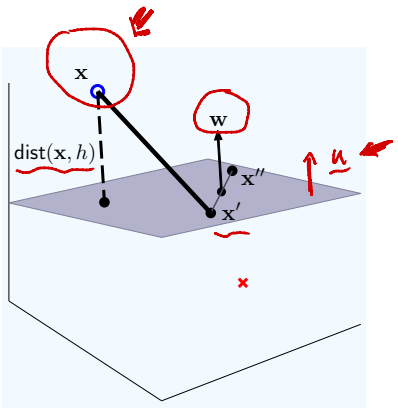
Definition (Separating Hyperplane). The hyperplane h separates the data if and only if it can be represented by weights $(b, \mathbf{w})$ that satisfy

$$\min_{n=1,\cdots,N} y_n(\mathbf{w}^T\mathbf{x}_n + b) = 1 \quad (4)$$

Remark: some terminology: the parameters $(b, \mathbf{w})$ of a separating hyperplane that satisfy (4) are called **the canonical representation of the hyperplane.**

Margin of Separating Hyperplane

To compute the margin of a separating hyperplane, we need to compute the distance from the hyperplane to the nearest data point. As a start, let us compute the distance from an arbitrary point $\mathbf{x}$ to a separating hyperplane $h = (b, \mathbf{w})$ that satisfies Equation (4). Denote this distance by $\text{dist}(\mathbf{x}, h)$. Referring to the figure below, $\text{dist}(\mathbf{x}, h)$ is the length of the perpendicular line from $\mathbf{x}$ to h.



Q: Let $\mathbf{x}'$ be any point on the hyperplane, which means $\mathbf{w}^T\mathbf{x}' + b = 0$ (from Equation (1)). Let $\mathbf{u}$ be a unit vector that is normal to the hyperplane h. What is $\text{dist}(\mathbf{x}, h)$ in terms of $\mathbf{x}$, $\mathbf{x}'$, and $\mathbf{u}$?

A:

$\text{dist}(\mathbf{x}, h) = |\mathbf{u}^T(\mathbf{x} - \mathbf{x}')|$. This is the projection of the vector $(\mathbf{x} - \mathbf{x}')$ onto $\mathbf{u}$.

From the figure above, we realize any vector lying on the hyperplane can be expressed by $(\mathbf{x}'' - \mathbf{x}')$ for some points $\mathbf{x}'$ and $\mathbf{x}''$ **on the hyperplane**. This leads to $\mathbf{w}^T\mathbf{x}' + b = 0$ and $\mathbf{w}^T\mathbf{x}'' + b = 0$ or:

(5) $\mathbf{w}^T(\mathbf{x}'' - \mathbf{x}') = 0$

Q: From Equation (5), what a remarkable geometric property we have for $\mathbf{w}$?

A:

$\mathbf{w}$ is normal to the hyperplane

Setting $\mathbf{u} = \mathbf{w}/\|\mathbf{w}\|$, the distance from $\mathbf{x}$ to h becomes:

$$\text{dist}(\mathbf{x}, h) = |\mathbf{u}^T(\mathbf{x} - \mathbf{x}')| = \frac{|\mathbf{w}^T\mathbf{x} - \mathbf{w}^T\mathbf{x}'|}{\|\mathbf{w}\|} = \frac{|\mathbf{w}^T\mathbf{x} + b|}{\|\mathbf{w}\|}$$

$\mathbf{w}^T\mathbf{x}' + b = 0$

$\mathbf{w}^T\mathbf{x}' = -b$

We thus lead to an important conclusion: distance of any data points $(\mathbf{x}_n, y_n)$ in our training examples to the hyperplane is:

$$\text{dist}(\mathbf{x}_n, h) = \frac{|\mathbf{w}^T\mathbf{x}_n + b|}{\|\mathbf{w}\|} = \frac{y_n(\mathbf{w}^T\mathbf{x}_n + b)}{\|\mathbf{w}\|}$$

Fun Time: What is the distance of the data point nearest to the canonical representation of the hyperplane (**hint**: take a look of Equation (4))?

(1) $\frac{1}{\|\mathbf{w}\|}$ (2) $\frac{2}{\|\mathbf{w}\|}$ (3) $\frac{1}{\|\mathbf{w}\|^2}$ (4) cannot be determined.

Slido 073374

Remark: This simple expression for the distance of the nearest data point to the hyperplane is the entire reason why we chose to normalize $(b, \mathbf{w})$ as we did in (4). For any separating hyperplane satisfying (4), the margin is $1/\|\mathbf{w}\|$. If you hold on a little longer, you are about to reap the full benefit, namely a simple algorithm for finding the optimal hyperplane.

Maximum Margin of Separating Hyperplane

The maximum-margin separating hyperplane $(b^*, \mathbf{w}^*)$ is the one that satisfies the separation condition (4) with minimum weight-norm (since the margin is the inverse of the weight-norm). Instead of minimizing the weight-norm, we can equivalently minimize $\frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$, which is analytically more friendly. Therefore, to find this optimal hyperplane, we need to solve the following optimization problem.

(6) $\underset{b,\mathbf{w}}{\text{minimize:}} \quad \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$

subject to: $\min_{n=1,\cdots,N} y_n(\mathbf{w}^T\mathbf{x}_n + b) = 1$

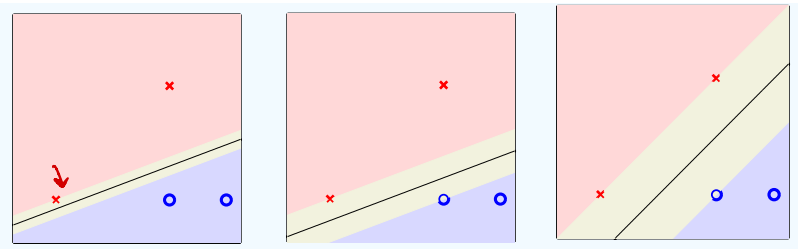
The constraint ensures that the hyperplane separates the data as per (4). Observe that the bias b does not appear in the quantity being minimized, but it is involved in the constraint. To make the optimization problem easier to solve, we can replace the single constraint $\min_{n=1,\cdots,N} y_n(\mathbf{w}^T\mathbf{x}_n + b) = 1$ with N 'looser' constraints $y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1$ for $n = 1, \cdots, N$ and solve the optimization problem:

(7) $\underset{b,\mathbf{w}}{\text{minimize:}} \quad \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$

subject to: $y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 \qquad (n = 1, \cdots, N)$

The constraint in (6) implies the constraints in (7), which means that the constraints in (7) are looser. Fortunately, at the optimal solution, the constraints in (7) become equivalent to the constraint in (6).

Example: We will use a toy example to solve (7). In two dimensions, a hyperplane is specified by the parameters (b, w_1, w_2). Let us consider a data set that was the basis for the figure below. There are many possibilities of hyperplanes to separate the data points. Our task is to use (7) to find the **optimal separating hyperplane** with the largest margin.



The data matrix and target values, together with the separability constraints from (7) are summarized below. The inequality on a particular row is the separability constraint for the corresponding data point in that row.

$$\mathrm{X} = \begin{bmatrix} 0 & 0 \\ 2 & 2 \\ 2 & 0 \\ 3 & 0 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} -1 \\ -1 \\ +1 \\ +1 \end{bmatrix}$$

$$\begin{aligned} -b &\geq 1 && \text{(i)} \\ -(2w_1 + 2w_2 + b) &\geq 1 && \text{(ii)} \\ 2w_1 + b &\geq 1 && \text{(iii)} \\ 3w_1 + b &\geq 1 && \text{(iv)} \end{aligned}$$

Q: What happens when we combine (i) and (iii)?

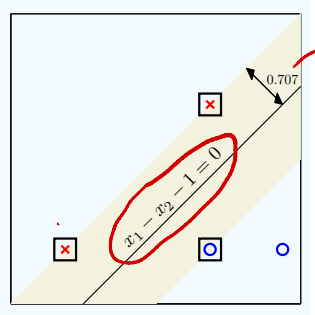
A:

$w_1 \geq 1$

Q: What happens when we combine (ii) and (iii)?

$w_2 \leq -1$

This means that $\frac{1}{2}(w_1^2 + w_2^2) \geq 1$ with the equality $\frac{1}{2}(w_1^2 + w_2^2) = 1$ when $w_1 = 1$ and $w_2 = -1$. We can then substitute these values into (i) – (iv) and verify that $(b^* = -1, w_1^* = 1, w_2^* = -1)$ satisfies all four constraints, minimizes $\frac{1}{2}(w_1^2 + w_2^2)$, and therefore gives the optimal hyperplane. The optimal hyperplane is shown in the following figure.



Q: What is our final hypothesis $g(\mathbf{x})$?

A:

$g(\mathbf{x}) = \text{sign}(x_1 - x_2 - 1)$

Q: How to solve the maximum margin?

A:

The maximum margin is $\frac{1}{\|\mathbf{w}^*\|} = \frac{1}{\sqrt{2}} = 0.707$

Remark: Data points (i), (ii) and (iii) are boxed because their separation constraints are exactly met: $y_n(\mathbf{w}^T\mathbf{x}_n + b) = 1$.

Quadratic Programming (QP): For bigger data sets, manually solving the optimization problem in (7) as we did in Example is no longer feasible. Fortunately, Equation (7) belongs to a well-studied family of optimization problems known as **quadratic programming (QP)**. Whenever you minimize a (convex) quadratic function, subject to linear inequality constraints, you can use quadratic programming. Quadratic programming is such a well-studied area that excellent, publicly available solvers exist for many numerical computing platforms.

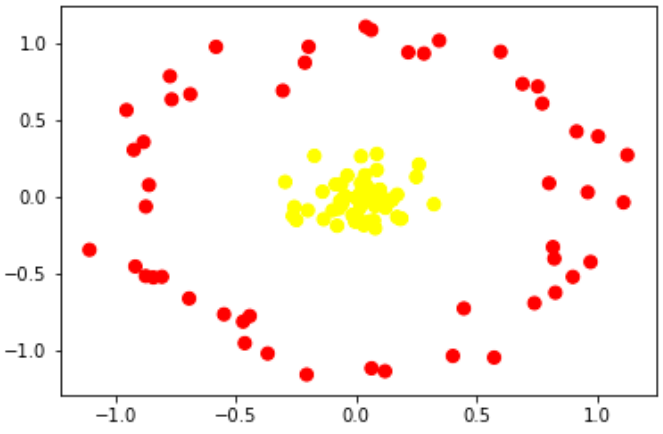
3. Beyond linear boundaries: Kernel SVM

Where SVM becomes extremely powerful is when it is combined with nonlinear **kernels**. To motivate the need for kernels, let's look at some data that is not linearly separable:

```
from sklearn.datasets.samples_generator import make_circles
X, y = make_circles(100, factor=.1, noise=.1)

clf = SVC(kernel='linear').fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
```



It is clear that no linear discrimination will ever be able to separate this data. But we can think about how we might project the data into a higher dimension such that a linear separator would be sufficient.

Q: Any potential basis function we can use to project the data so a linear separator would be sufficient?

A:

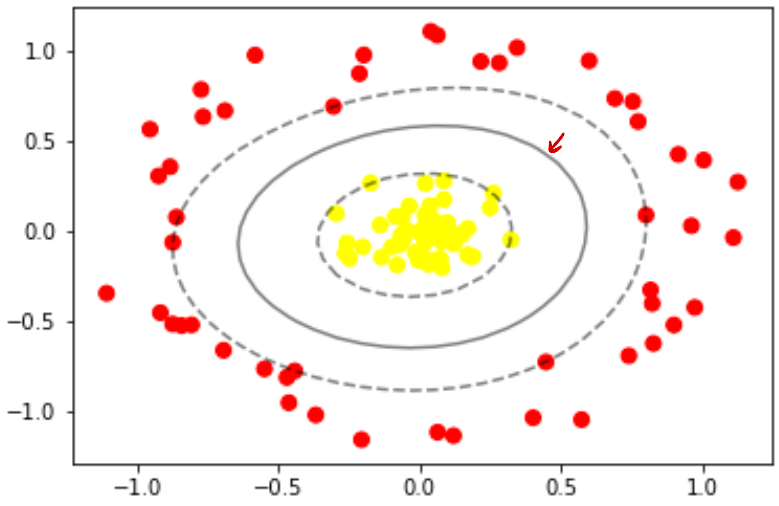
One simple projection we could use would be to compute **a radial basis function** centered on the middle clump. A radial basis function is a real-valued function whose value depends only on the distance from the origin. For example, the RBF in `Scikit-Learn` uses the form $e^{-\gamma\|x-x'\|}$ where $\gamma > 0$.

In `Scikit-Learn`, we can apply kernelized SVM simply by changing our linear kernel to an RBF (radial basis function) kernel:

```
clf = SVC(kernel='rbf', C=1E6)
clf.fit(X, y)
```

```
SVC(C=1000000.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

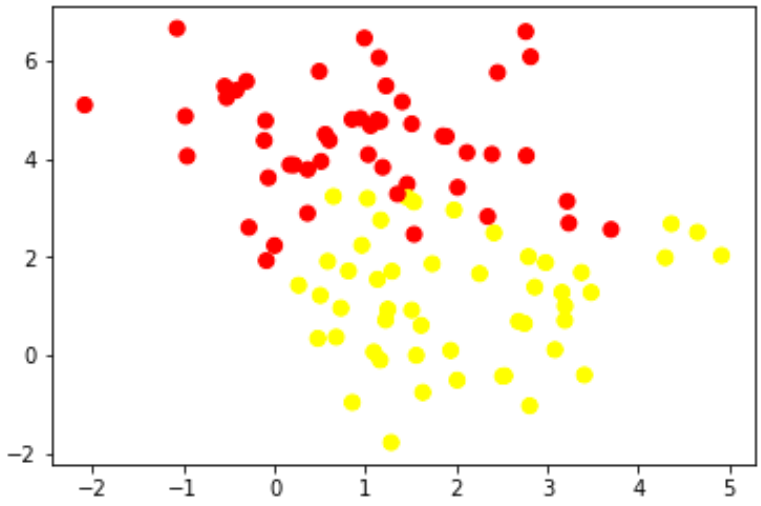
```
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(clf)
```



Using this kernelized support vector machine, we learn a suitable nonlinear decision boundary. This kernel transformation strategy is used often in machine learning to **turn fast linear methods into fast nonlinear methods**, especially for models in which the *kernel trick* can be used.

4. Softening Margins

Our discussion thus far has centered around very clean datasets, in which a perfect decision boundary exists. *But what if your data has some amount of overlap?* For example, you may have data like this:



To handle this case, the SVM implementation has a bit of a *fudge-factor*[2] which "softens" the margin: that is, it allows some of the points to creep into the margin if that allows a better fit. The hardness of the margin is controlled by a tuning parameter, most often known as C. For very large C, the margin is hard, and points cannot lie in it. For smaller C, the margin is softer, and can grow to encompass some points.

= hyperparameter

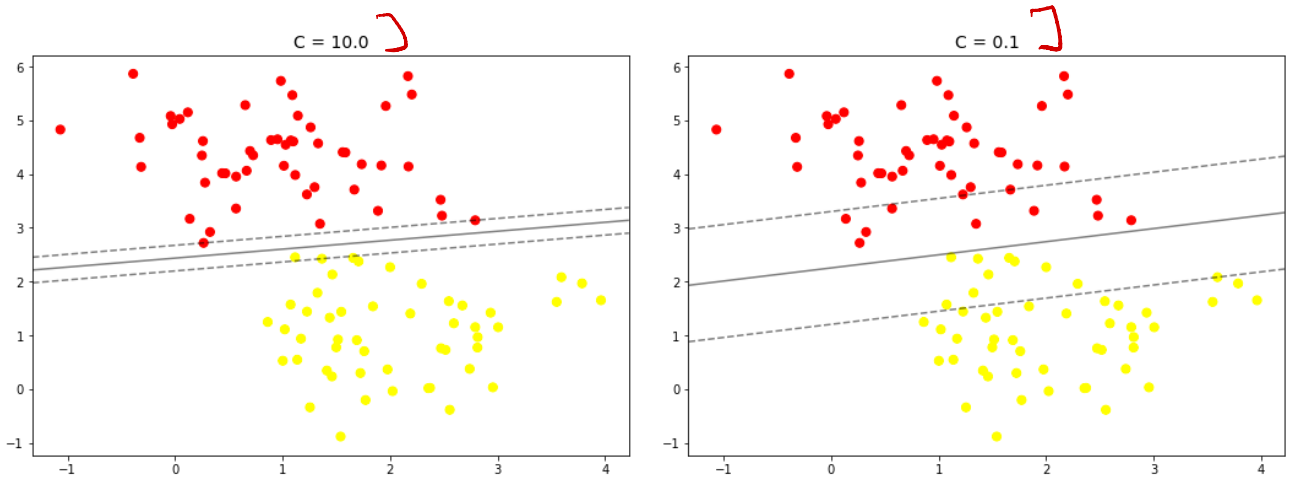
The plot shown below gives a visual picture of how a changing C parameter affects the final fit, via the softening of the margin:

```
X, y = make_blobs(n_samples=100, centers=2,
                  random_state=0, cluster_std=0.8)

fig, ax = plt.subplots(1, 2, figsize=(16, 6))
fig.subplots_adjust(left=0.0625, right=0.95, wspace=0.1)

for axi, C in zip(ax, [10.0, 0.1]):
    model = SVC(kernel='linear', C=C).fit(X, y)
    axi.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
    plot_svc_decision_function(model, axi)
    axi.scatter(model.support_vectors_[:, 0],
                model.support_vectors_[:, 1],
                s=300, lw=1, facecolors='none');
    axi.set_title('C = {0:.1f}'.format(C), size=14)
```

[2] A *fudge-factor* is an ad hoc quantity or element introduced into a calculation, formula or model in order to make it fit observations or expectations. Examples include Einstein's Cosmological Constant, dark energy, the initial proposals of dark matter and inflation.



The optimal value of the `C` parameter will depend on your dataset, and should be tuned using **cross-validation**.

You can download the above Python codes `SVM_tutorial.ipynb` from the course website.

5. Summary: Support Vector Machine

We have seen a brief introduction to the principles behind support vector machines. These methods are a powerful classification method for a number of reasons:

- Their dependence on relatively few support vectors means that they are very compact models, and take up very little memory.
- Once the model is trained, the prediction phase is very fast.
- Because they are affected only by points near the margin, they work well with high-dimensional data—even data with more dimensions than samples, which is a challenging regime for other algorithms.
- Their integration with kernel methods makes them very versatile, able to adapt to many types of data.

However, SVMs have several disadvantages as well:

- The scaling with the number of samples N is $O(N^3)$ at worst, or $O(N^2)$ for efficient implementations. For large number of training samples, this computational cost can be prohibitive.
- The results are strongly dependent on a suitable choice for the softening parameter C. This must be carefully chosen via **cross-validation**, which can be expensive as datasets grow in size.
- The results do not have a direct probabilistic interpretation. This can be estimated via an internal cross-validation (see the probability parameter of SVC), but this extra estimation is costly.

With those traits in mind, we generally only turn to SVMs once other simpler, faster, and less tuning-intensive methods have been shown to be insufficient for our needs. Nevertheless, if you have the CPU cycles to commit to training and cross-validating an SVM on your data, the method can lead to excellent results.