Large Margin Classification

So far in the presentation, the difference between the SVC and Logistic Regression classifiers is in the Loss Function.

The SVC is also able to create a "buffer" on either side of the separating boundary.

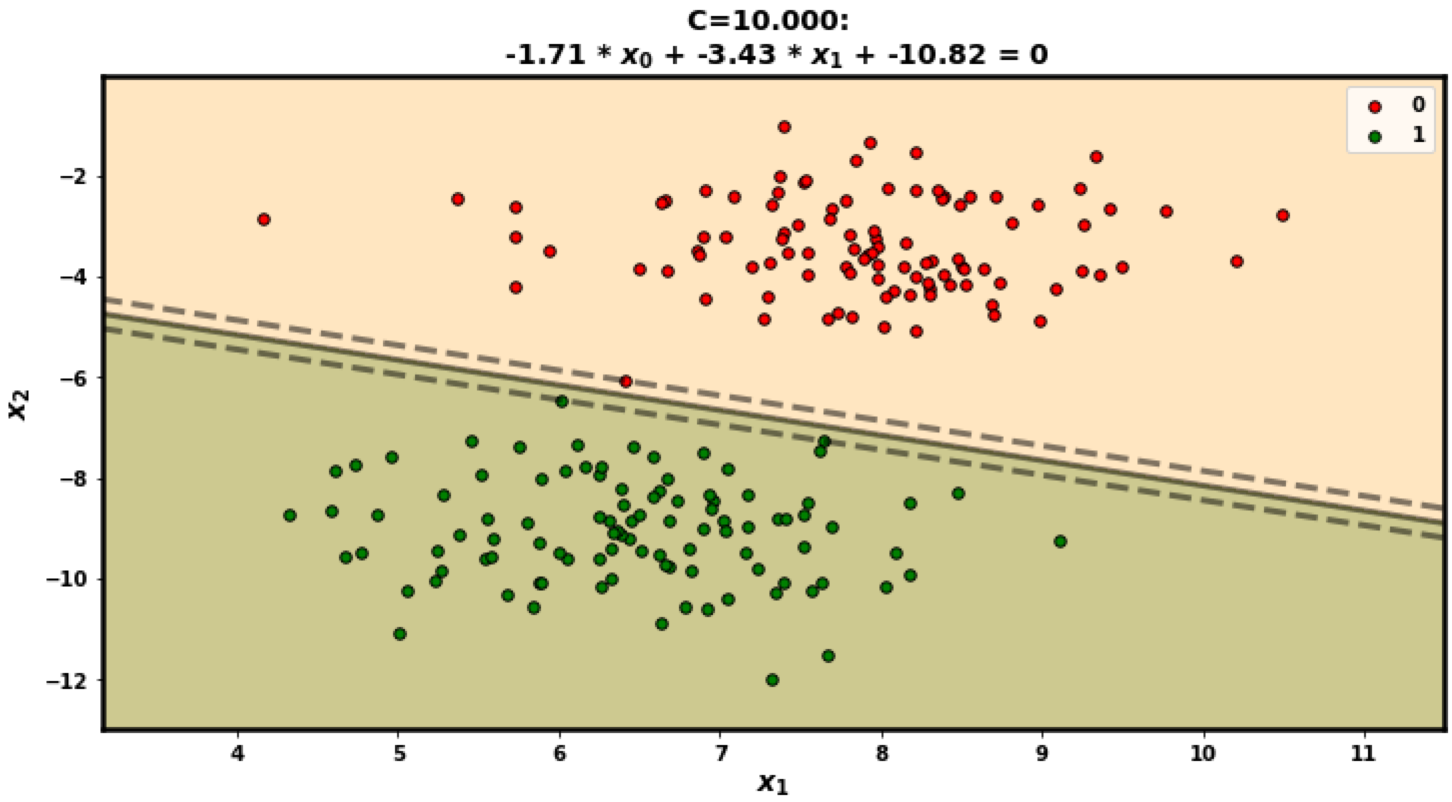
By making this buffer as wide as possible, an SVC may generalize better.

The buffer is defined by

- Two additional lines
- Parallel to separating boundary
- Same distance (the *margin*) from the separating boundary

In [4]:

```
svm_ch = svm_helper.Charts_Helper()
_= svm_ch.create_data()
fig, axs = svm_ch.create_margin(Cs=[10])
```



- The separating boundary is the solid line, whose equation is given in the title
- Each dashed line is
    - Parallel to, and at the same distance from, the separating boundary
    - The distance (measured by length of a line orthogonal to the boundary) from the separating boundary is called the *margin*

The buffer width is twice the margin

In the above plot

- All examples are correctly classified
- There are no examples in the buffer

Requiring these two properties is called *Hard Margin* Classification.

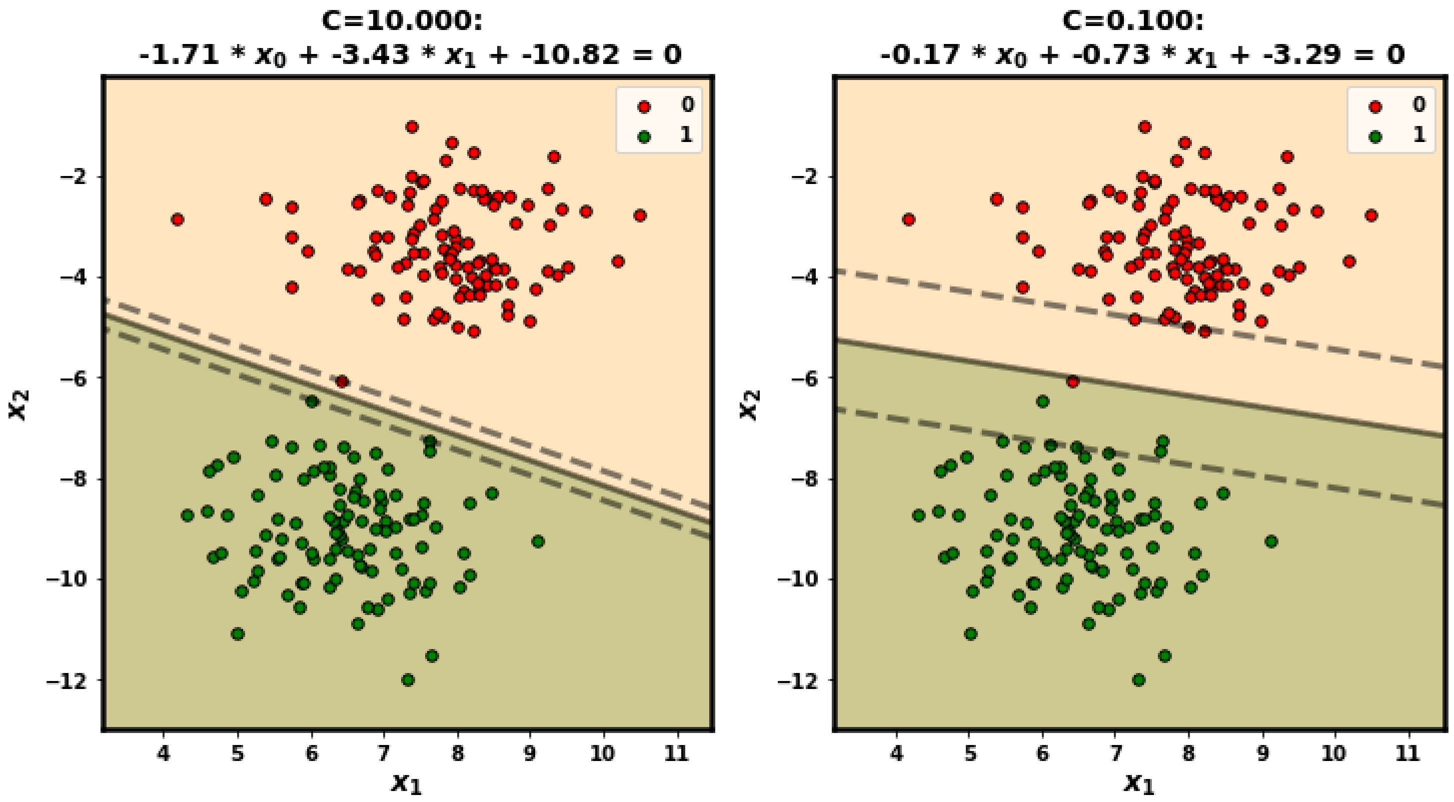
It is somewhat uncommon to be able to achieve the first property (perfect separation of classes).

A more natural Classification task is called *Soft Margin* classification which allows (but penalizes, via the Loss Function) violation of either property.

We re-run the above example with a larger margin, resulting in the presence of (correctly classified) examples in the buffer

In [5]:

```
svm_ch = svm_helper.Charts_Helper()
_= svm_ch.create_data()
fig, axs = svm_ch.create_margin(Cs=[10,.1])
```



We concentrate on Soft Margin Classification going forward.

Achieving a margin

We need to modify the per-example loss to achieve zero loss

- If the example is correctly classified (i.e., score is on correct side of separating boundary)
- **and** the example is not in the buffer (i.e., score is exceeds the margin)

This can be achieved by moving the "hinge point" of the Hinge Function

- From 0 to the margin m

This corresponds to a per-example Loss of

$$\mathcal{L}^{(\mathbf{i})} = \max\left(0, \mathrm{m} - \dot{\mathbf{y}}^{(\mathbf{i})} * s(\hat{\mathbf{x}})\right)$$

The above expression achieves zero loss when

$$
\begin{aligned}
\hat{s}\left(\mathbf{x^{(i)}}\right) &\geq \mathrm{m} && \text{Positive example, } \dot{\mathbf{y}}^{(\mathbf{i})} = +1 \\
\hat{s}\left(\mathbf{x^{(i)}}\right) &\leq -\mathrm{m} && \text{Negative example, } \dot{\mathbf{y}}^{(\mathbf{i})} = -1
\end{aligned}
$$

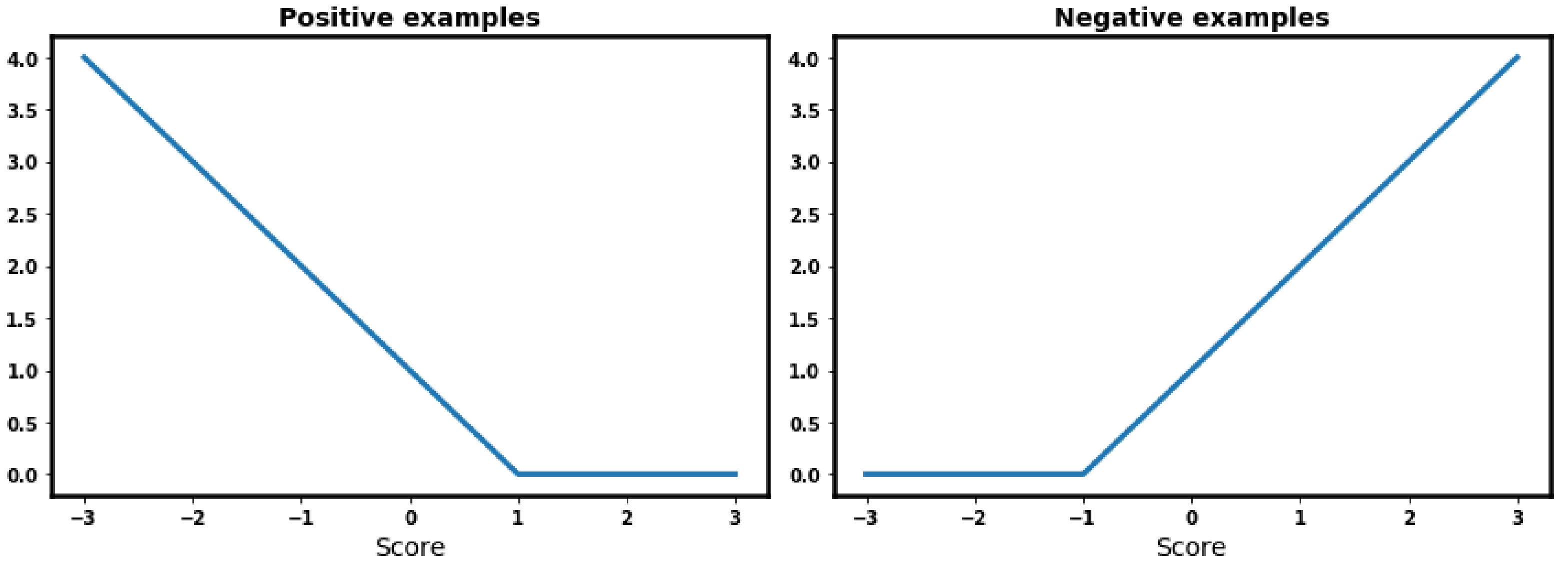
As we shall see, a margin $\mathrm{m} = 1$ will suffice resulting in

$$\mathcal{L}^{(\mathbf{i})} = \max\left(0, 1 - \dot{\mathbf{y}}^{(\mathbf{i})} * s(\hat{\mathbf{x}})\right)$$

which we shall encounter repeatedly.

Here's the plot

In [6]:
```
svmh.plot_hinges(hinge_pt=1)
```



Achieving a large margin

As we observed above, a zero loss occurs when

$$\begin{array}{ll} \hat{s}\left(\mathbf{x^{(i)}}\right) \ge \mathrm{m} & \text{Positive example, } \dot{\mathbf{y}}^{(\mathbf{i})} = +1 \\ \hat{s}\left(\mathbf{x^{(i)}}\right) \le -\mathrm{m} & \text{Negative example, } \dot{\mathbf{y}}^{(\mathbf{i})} = -1 \end{array}$$

Hence, the per-example loss above only imposes a *Classification Loss*

- Penalizing incorrect predictions
- Penalizing correct predictions that are in the buffer

It does not force m to be large.

In order to do so, we need to impose a *Margin Penalty* inversely related to the size of m.

What would happen if we divided both sides of the inequality by m ?

- Zero loss occurs when the inequality's right hand side is 1
- Θ would be rescaled by a factor of $\frac{1}{\mathrm{m}}$

This would result in a large margin m being associated with *small* Θ.

We define a Margin Penalty

$$\frac{1}{2} {\Theta^T}_{-0} \cdot \Theta_{-0}$$

as part of the Loss (that is being minimized) in order to force large m

- where Θ_{-0} is a minor variation of Θ as explained below

Notation Our convention is that each example $\mathbf{x}^{(\mathbf{i})}$ has first feature that is the constant 1:

$$\mathbf{x}^{(\mathbf{i})} = [1, \mathbf{x}^{(\mathbf{i})}_1, \ldots, \mathbf{x}^{(\mathbf{i})}_n]$$

- Design matrix $\mathbf{X}$ has been augmented with a first column of all 1's
- This allows us to write $\hat{s}\left(\mathbf{x}^{(\mathbf{i})}\right) = \Theta^T \cdot \mathbf{x}^{(\mathbf{i})}$
- Θ_0 is the intercept term

Other's (e.g., the Geron book) keep the intercept term *outside* of $\mathbf{x}$

- Resulting in $\hat{s}\left(\mathbf{x}^{(\mathbf{i})}\right) = \Theta^T \cdot \mathbf{x}^{(\mathbf{i})} + \Theta_0$, where $\mathbf{x}$ *does not* have a leading 1
- Geron changes notation from previous chapters (in the "Under the Hood" subsection, page 204)

To avoid confusion, we will write Θ_{-0} to be Θ *excluding* Θ_0

Aside

The mysterious $\frac{1}{2}$ in the Margin Penalty

- Doesn't really affect the overall cost in a significant way
- Will be useful in the mathematical derivations
    - Hint:
        - $\frac{\partial \Theta^2}{\partial \Theta} = 2\Theta$
        - The $\frac{1}{2}$ makes the derivative of the Margin Penalty with respect to Θ exactly Θ
        - The derivative will be used in the optimization of SVM Cost

SVC Loss Function

The final Average Loss Function for the SVC combines

- Classification Loss per-example (penalize incorrect or in-the-buffer predictions)
- Margin Penalty (penalize small margins)

$$\mathcal{L} = \frac{1}{2} \Theta^T_{-0} \cdot \Theta_{-0} + C * \frac{1}{m} \sum_{i=1}^{m} \max \left(0, 1 - \dot{\mathbf{y}}^{(\mathbf{i})} * s(\hat{\mathbf{x}}^{(\mathbf{i})}) \right)$$

where

$$\hat{s}\left(\mathbf{x}^{(\mathbf{i})}\right) = \mathbf{\Theta}^T \cdot \mathbf{x}^{(\mathbf{i})}$$

- The first term is the Margin Penalty
- The second term is the average of the per-example losses $\mathcal{L}_i$
    - weighted by a constant C

What is C?

- We have two loss terms: Margin Penalty and Average Classification Loss
- C allows us to express a weight for the relative importance of the two loss terms

You should recognize this form of loss function (two loss terms, with relative weight)

- It is like a loss function with a Regularization Penalty

In fact, we will provide a mathematical derivation of the Loss that makes this more apparent.

Let's consider extreme cases of C:

$$\begin{array}{ll} C = \infty & \text{No misclassification or buffer violations allowed} \\ C = 0 & \text{Misclassification and buffer violations unimportant} \end{array}$$

A high value for C

- May prevent a solution
- Encourage overfitting
  - Less importance on forcing elements of Θ to be zero

A low value for C

- Encourages underfitting
  - More importance on forcing elements of Θ to be zero

In [7]:

```
print("Done")
```

Done