Lecture 31 - Kernel Machines; Support Vector Machines

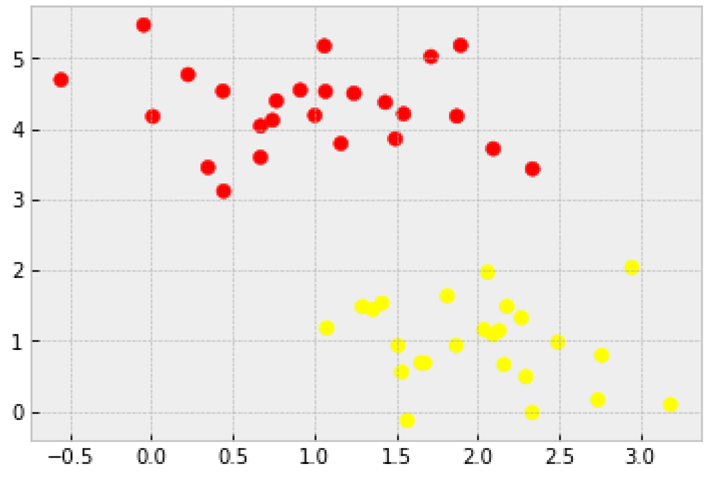
In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('bmh')
```

Kernel Machine

In [2]:

```
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=50, centers=2,
                  random_state=0, cluster_std=0.60)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn');
```



A linear discriminative classifier would attempt to draw a straight line separating the two sets of data, and thereby create a model for classification. For two dimensional data like that shown here, this is a task we could do by hand. But immediately we see a problem: there is more than one possible dividing line that can perfectly discriminate between the two classes!

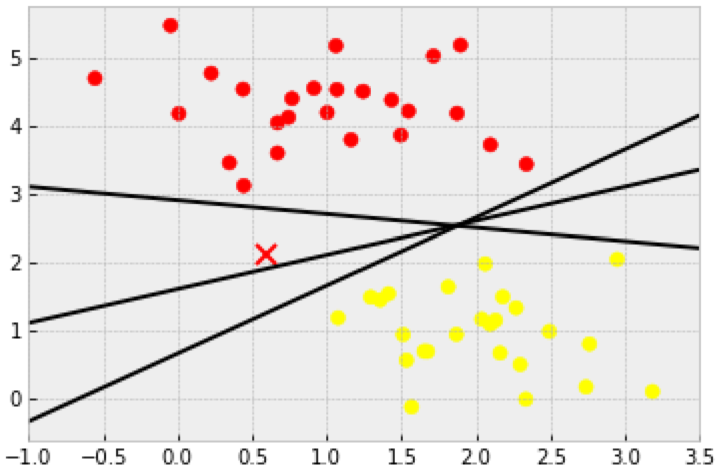
We can draw them as follows:

In [3]:

```
xfit = np.linspace(-1, 3.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plt.plot([0.6], [2.1], 'x', color='red', markeredgewidth=2, markersize=10)

for m, b in [(1, 0.65), (0.5, 1.6), (-0.2, 2.9)]:
    plt.plot(xfit, m * xfit + b, '-k')

plt.xlim(-1, 3.5);
```

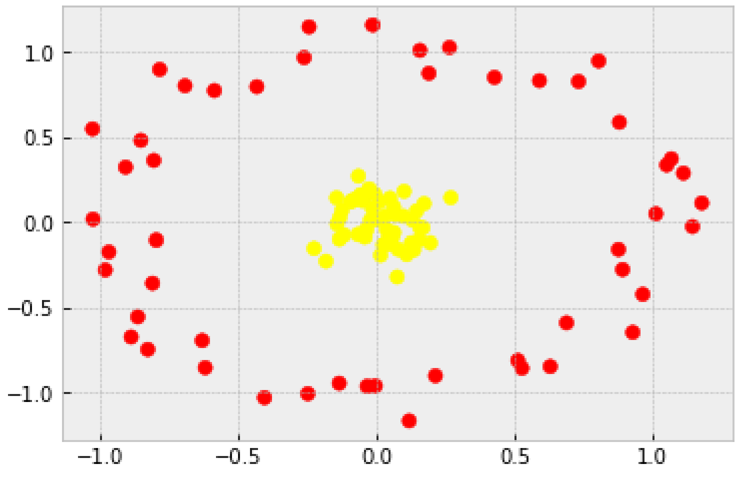


These are three very different separators which, nevertheless, perfectly discriminate between these samples. Depending on which you choose, a new data point (e.g., the one marked by the "X" in this plot) will be assigned a different label! Evidently our simple intuition of "drawing a line between classes" is not enough, and we need to think a bit deeper.

Kernel Machine

Consider the following data set:

In [4]:
```
from sklearn.datasets import make_circles
X, y = make_circles(100, factor=.1, noise=.1)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn');
```



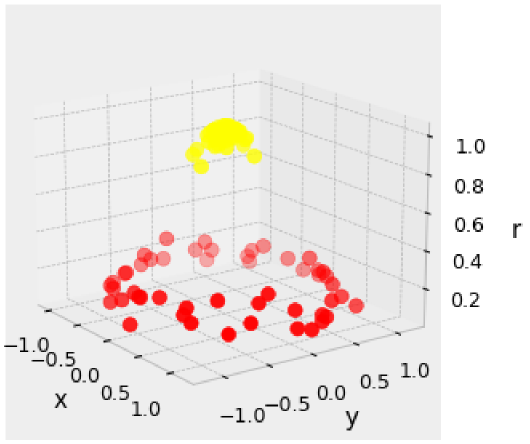
The two classes are clearly not linearly separable. What would you do?

In [5]:
```
r = np.exp(-((X - X.mean(axis=0))** 2/(2*X.std(axis=0, ddof=1)**2)).sum(axis=1)) #RBF (
```

In [6]:
```
from mpl_toolkits import mplot3d

def plot_3D(elev=30, azim=30, X=X, y=y):
    ax = plt.subplot(projection='3d')
    ax.scatter3D(X[:, 0], X[:, 1], r, c=y, s=50, cmap='autumn')
    ax.view_init(elev=elev, azim=azim)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('r')

%matplotlib notebook
plot_3D();
```



Support Vector Machines: Maximizing the Margin

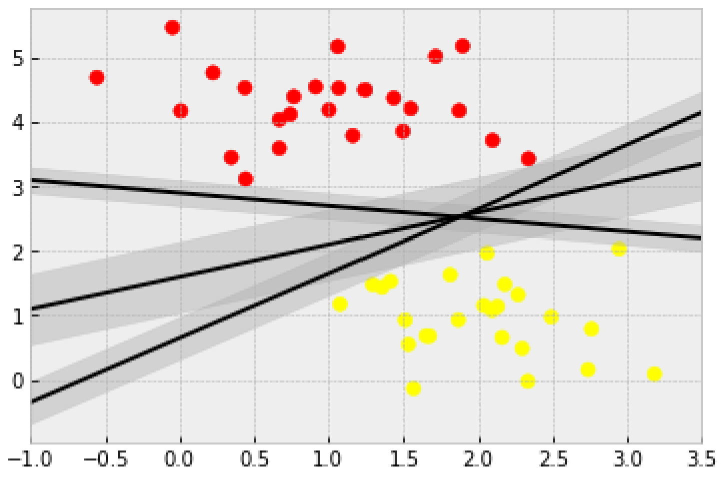
SVMs offer one way to improve on this. The intuition is this: rather than simply drawing a zero-width line between the classes, we can draw around each line a margin of some width, up to the nearest point. Here is an example of how this might look:

In [7]:

```
%matplotlib inline
X, y = make_blobs(n_samples=50, centers=2,
                  random_state=0, cluster_std=0.60)
xfit = np.linspace(-1, 3.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')

for m, b, d in [(1, 0.65, 0.33), (0.5, 1.6, 0.55), (-0.2, 2.9, 0.2)]:
    yfit = m * xfit + b
    plt.plot(xfit, yfit, '-k')
    plt.fill_between(xfit, yfit - d, yfit + d, edgecolor='none',
                     color='#AAAAAA', alpha=0.4)

plt.xlim(-1, 3.5);
```



In support vector machines, the line that maximizes this margin is the one we will choose as the optimal model. Support vector machines are an example of such a maximum margin estimator.

Support Vector Machine: Separable Classes

Let's start with the two-class linearly separable task and then we will extend the method to more general cases where data are not separable. Let $\phi(x_i)$, $i = 1, 2, \ldots, N$, be the feature vectors of the training set, X, and corresponding target values $t_1, t_2, \cdots, t_N$ where $t_n \in \{-1, 1\}$. These belong to either of two classes, C_1, C_2, which are *assumed to be linearly separable*.

The goal, once more, is to design a hyperplane

$$y(x) = w^T \phi(x) + b = 0$$

that classifies correctly all the training vectors.

Because the training data is linearly separable in the feature space, by definition there exists at least one choice of the parameters w and b such that $y(x)$ satisfies $y(x_n) > 0$ for points having $t_n = +1$ and $y(x_n) < 0$ for points having $t_n = -1$, so that $t_n y(x_n) > 0$ for all training data points.

Such a hyperplane is not unique. The perceptron algorithm may converge to any one of the possible solutions. Having gained in experience, this time we will be more demanding.

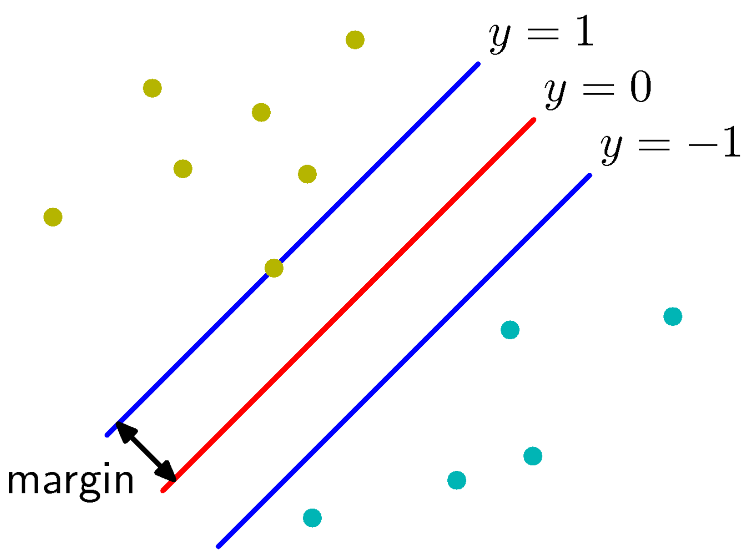
- Which hyperplane would any sensible engineer choose as the classifier for operation in practice, where data outside the training set will be fed to it?
- Once again, the hyperplane that leaves more “room” on either side, so that data in both classes can move a bit more freely, with less risk of causing an error.
- A sensible choice for the hyperplane classifier would be the one that leaves the maximum margin from both classes.
- Thus such a hyperplane can be trusted more, when it is faced with the challenge of operating with unknown data.
    - It has a higher generalizarion performance.

Quantifying the "Margin"

Let us now quantify the term "margin" that a hyperplane leaves from both classes. Every hyperplane is characterized by its direction (determined by w) and its exact position in space (determined by b). Since we want to give no preference to either of the classes, then it is reasonable for each direction to select that hyperplane which has the same distance from the respective nearest points in c_1 and c_2.

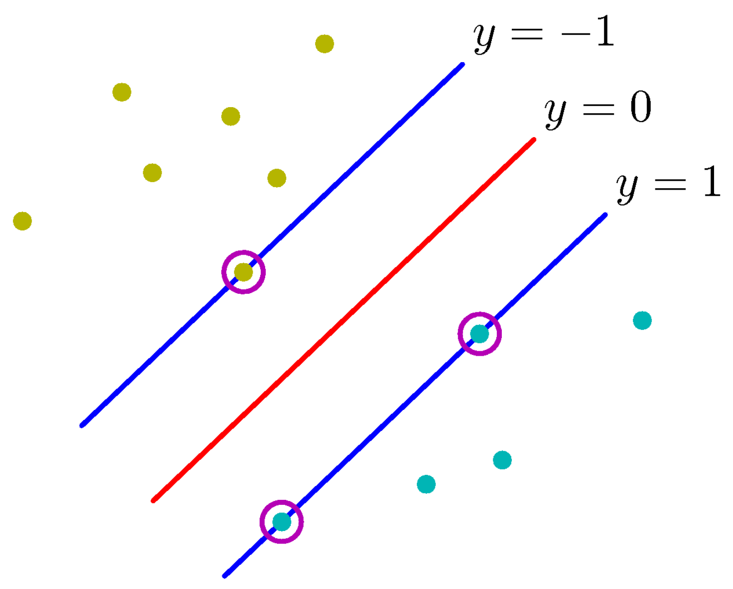
In [8]:
```
from IPython.display import Image
Image('figures/Figure7.1a.png', width=400)
```

Out[8]:



In [9]:
```
Image('figures/Figure7.1b.png', width=400)
```

Out[9]:



Recall that the perpendicular distance of a point x from a hyperplane defined by $y(x) = 0$ where $y(x)$ takes the form $y(x) = w^T\phi(x) + b$ is given by $\frac{|y(x)|}{\|w\|}$.

Furthermore, we are only interested in solutions for which all data points are correctly classified, so that $t_n y(x_n) > 0, \forall n$. Thus the distance of a point x_n to the decision surface is given by

$$\frac{t_n y(x_n)}{\|w\|} = \frac{t_n\left(w^t\phi(x_n) + b\right)}{\|w\|}$$

The margin is given by the perpendicular distance to the closest point x_n from the data set, and we wish to optimize the parameters w and b in order to maximize this distance. Thus the maximum margin solution is found by solving

$$\arg_{w,b}\max\left\{\frac{1}{\|w\|}\min_n\left[t_n(w^T\phi(x_n) + b)\right]\right\}$$

where the factor $\frac{1}{\|w\|}$ is taken outside the optimization over n because w does not depend on n.

- Direct solution of this optimization problem would be very complex, and so we shall convert it into an equivalent problem that is much easier to solve.
- To do this we note that if we make the rescaling $w \to \kappa w$ and $b \to \kappa b$, then the distance from any point x_n to the decision surface, given by $\frac{t_n y(x_n)}{\|w\|}$, is unchanged. We can use this freedom to set

$$t_n(w^T\phi(x_n) + b) = 1$$

for the point that is closest to the surface, called the **support vectors**. In this case, all data points will satisfy the constraints

$$t_n(w^T\phi(x_n) + b) \geq 1, n = 1, 2, \ldots, N$$

- This is known as the canonical representation of the decision hyperplane.
- In the case of data points for which the equality holds (support vectors), the constraints are said to be active, whereas for the remainder they are said to be inactive.
- By definition, there will always be at least one active constraint, because there will always be a closest point, and once the margin has been maximized there will be at least two active constraints.

The optimization problem then simply requires that we maximize $\|w\|^{-1}$, which is equivalent to minimizing $\|w\|^2$, and so we have to solve the optimization problem

$$\text{arg}_{w,b}\min \frac{1}{2}\|w\|^2 \tag{1}$$
$$\text{subject to } t_n(w^T\phi(x_n)+b) \geq 1 \tag{2}$$

In order to solve this constrained optimization problem, we introduce *Lagrange multipliers* $a_n \geq 0$, with one multiplier an for each of the constraints, giving the Lagrangian function

$$L(w,b,a) = \frac{1}{2}\|w\|^2 - \sum_{n=1}^{N} a_n \left(t_n(w^T\phi(x_n)+b)-1\right)$$

- Note the minus sign in front of the Lagrange multiplier term, because we are minimizing with respect to w and b, and maximizing with respect to a.

To be continued...