Final Project

- The training dataset has been released
- HiPerGator accounts have been created for all of you that didn't already have one.
    - You were added to the 'eee4773' group.
    - You all must agree with the [HiPerGator Acceptable Use Policy (https://ufl.instructure.com/courses/404363/pages/hipergator-acceptable-use-policy)](https://ufl.instructure.com/courses/404363/pages/hipergator-acceptable-use-policy)
- On Monday, November 2, during our regular class period (11:45 AM - 12:35 PM), Matt Gitzendanner will be introducing you to the HiPerGator and how to submit jobs.

Lecture 25 - Support Vector Machine (SVM)

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('seaborn-colorblind')
from IPython.display import Image
```

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.

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

$$\begin{aligned} &\arg_{w,b} \min \frac{1}{2} \|w\|^2 \\ &\text{subject to } t_n(w^T \phi(x_n) + b) \geq 1 \end{aligned}$$

Quick Review or Illustration of what Lagrange Multipliers Are

- Reading Source: Appendix E "Lagrange Multipliers" from the Bishop textbook (https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf).

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.

Setting the derivatives of $L(w,b,a)$ with respect to w and b equal to zero, we obtain the following two conditions:

$$w = \sum_{n=1}^{N} a_n t_n \phi(x_n)$$
$$0 = \sum_{n=1}^{N} a_n t_n$$

Eliminating w and b from $L(w,b,a)$ using these conditions then gives the *dual representation* of the maximum margin problem in which we maximize

$$\tilde{L}(a) = \sum_{n=1}^{N} a_n - \sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(x_n, x_m)$$

with respect to a subject to the constraints

$$a_n \geq 0, n = 1, 2, \ldots, N$$
$$\sum_{n=1}^{N} a_n t_n = 0$$

Here the kernel function is defined by $k(x,y) = \phi(x)^T\phi(y)$.

- This takes the form of a quadratic programming problem in which we optimize a quadratic function of a subject to a set of inequality constraints.
- The solution to a quadratic programming problem in M variables in general has computational complexity that is $O(M^3)$. In going to the dual formulation we have turned the original optimization problem into the dual problem, which has N variables.
  - For a fixed set of basis functions whose number M is smaller than the number N of data points, the move to the dual problem appears disadvantageous. However, it allows the model to be reformulated using kernels, and so the maximum margin classifier can be applied efficiently to feature spaces whose dimensionality exceeds the number of data points, including infinite feature spaces.
  - The kernel formulation also makes clear the role of the constraint that the kernel function $k(x,y)$ be positive definite, because this ensures that the Lagrangian function $\tilde{L}(a)$ is bounded below, giving rise to a well defined optimization problem.

In order to classify new data points using the trained model, we evaluate the sign of $y(x) = w^T\phi(x) + b$. This can be expressed in terms of the parameters $\{a_n\}$ and the kernel function by substituting for w to give

$$y(x) = \sum_{n=1}^{N} a_n t_n k(x, x_n) + b$$

- A constrained optimization of this form satisfies the Karush-Kuhn-Tucker (KKT) conditions, which in this case require that the following three properties hold

$$\begin{aligned} &a_n \geq 0 \\ &t_n y(x_n) - 1 \geq 0 \\ &a_n\left(t_n y(x_n) - 1\right) = 0 \end{aligned}$$

Thus for every data point, either $a_n = 0$ or $t_n y(x_n) = 1$.

- Any data point for which $a_n = 0$ will not appear in the sum of $y(x)$ and hence plays no role in making predictions for new data points.
- The remaining data points are called **support vectors**, and because they satisfy $t_n y(x_n) = 1$, they correspond to points that lie on the maximum margin hyperplanes in feature space.
- Once the model is trained, a significant proportion of the data points can be discarded and only the support vectors retained!

Having solved the quadratic programming problem and found a value for a, we can then determine the value of the threshold parameter b by noting that any support vector x_n satisfies $t_n y(x_n) = 1$.

$$t_n\left(\sum_{m\in S} a_m t_m k(x_n, x_m) + b\right) = 1$$

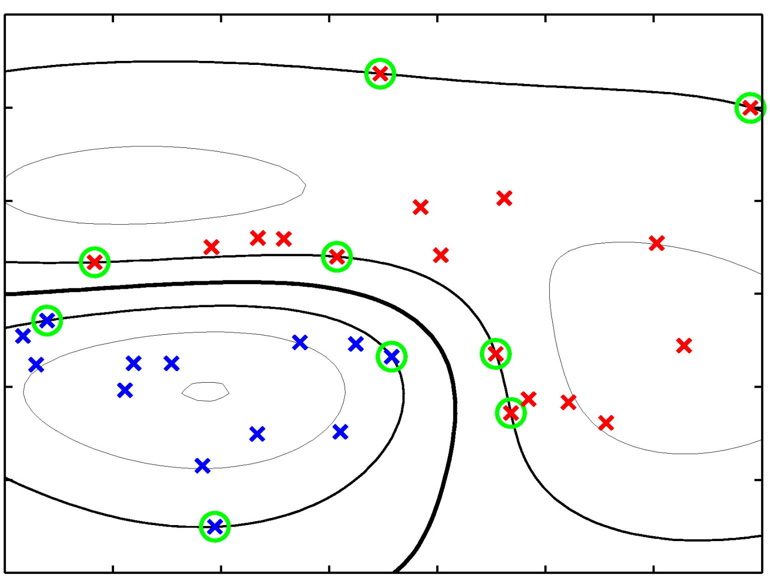
where S denotes the set of indices of the support vectors. Although we can solve this equation for b using an arbitrarily chosen support vector x_n, a numerically more stable solution is obtained by first multiplying through by t_n, making use of $t_n^2 = 1$, and then averaging these equations over all support vectors and solving for b to give

$$b = \frac{1}{N_S}\sum_{n\in S}\left(t_n - \sum_{m\in S} a_m t_m k(x_n, x_m)\right)$$

where N_S is the total number of support vectors.

In [7]:
```
Image("figures/Figure7.2.png", width=400)
```

Out[7]:



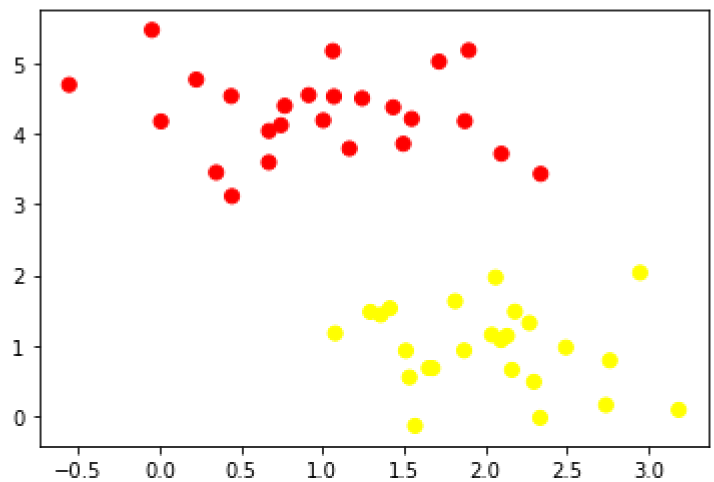
This figure (7.2 from the Bishop textbook) is an example of the classification resulting from training a support vector machine on a simple synthetic data set using a Gaussian kernel.

Although the data set is not linearly separable in the two-dimensional data space x, it is linearly separable in the nonlinear feature space defined implicitly by the nonlinear kernel function. Thus the training data points are perfectly separated in the original data space.

This example also provides a geometrical insight into the origin of sparsity in the SVM. The maximum margin hyperplane is defined by the location of the support vectors. Other data points can be moved around freely (so long as they remain outside the margin region) without changing the decision boundary, and so the solution will be independent of such data points.

Example

In [5]:
```
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');
```

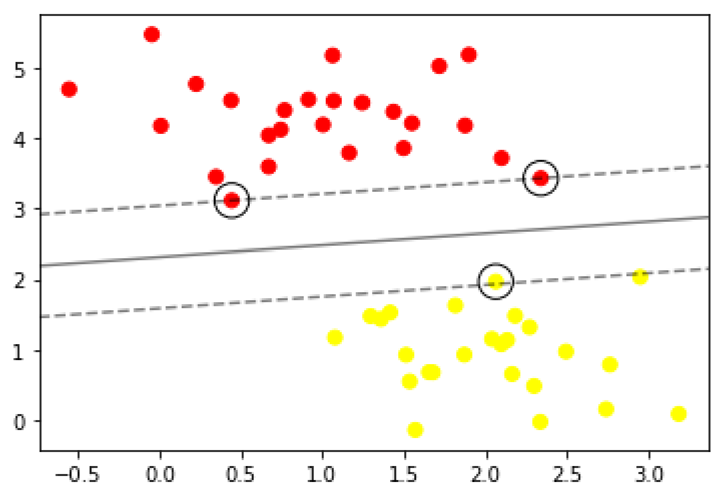


In [7]:
```
from sklearn.svm import SVC # "Support vector classifier"
model = SVC(kernel='linear')
model.fit(X, y)
```

Out[7]:
```
SVC(kernel='linear')
```

In [8]:
```
def plot_svc_decision_function(model, ax=None, plot_support=True):
    """Plot the decision function for a 2D SVC"""
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # create grid to evaluate model
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    # plot decision boundary and margins
    ax.contour(X, Y, P, colors='k',
               levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])

    # plot support vectors
    if plot_support:
        ax.scatter(model.support_vectors_[:, 0],
                   model.support_vectors_[:, 1],
                   s=300, linewidth=1, edgecolors='black',facecolors='none');
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
```

In [9]:
```
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(model);
```

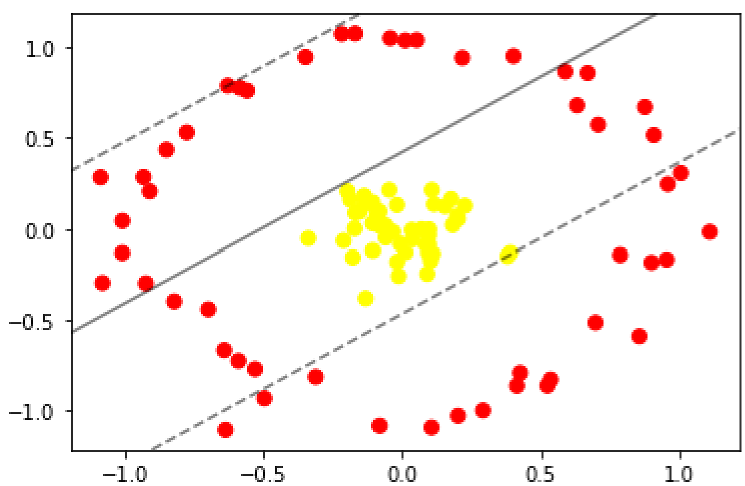


In [10]:
```
model.support_vectors_
```

Out[10]:
```
array([[0.44359863, 3.11530945],
       [2.33812285, 3.43116792],
       [2.06156753, 1.96918596]])
```

In [11]:
```python
def plot_svm(N=10, ax=None):
    X, y = make_blobs(n_samples=200, centers=2,
                      random_state=0, cluster_std=0.60)
    X = X[:N]
    y = y[:N]
    model = SVC(kernel='linear', C=1E10)
    model.fit(X, y)

    ax = ax or plt.gca()
    ax.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
    ax.set_xlim(-1, 4)
    ax.set_ylim(-1, 6)
    plot_svc_decision_function(model, ax)

from ipywidgets import interact, fixed
interact(plot_svm, N=[10, 30, 60, 100, 200], ax=fixed(None));
```

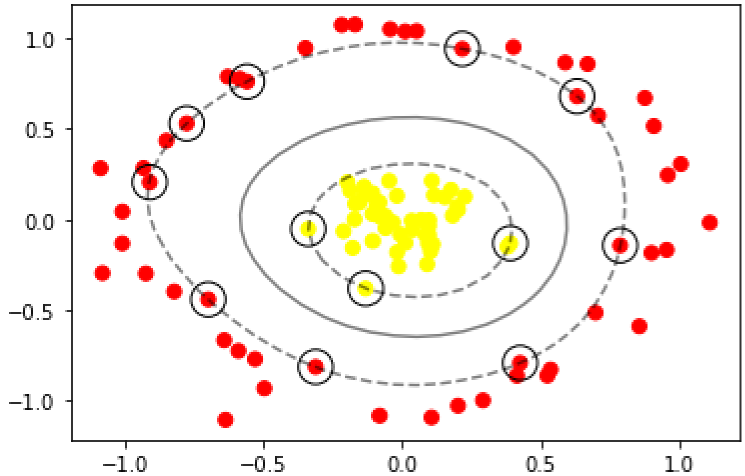
In [13]:
```python
from sklearn.datasets import make_circles
X, y = make_circles(100, factor=.1, noise=.1)

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

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



In [14]:
```python
clf = SVC(kernel='rbf', C=1E6)
clf.fit(X, y)
```

Out[14]:
```
SVC(C=1000000.0)
```

In [15]:
```
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(clf)
plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1],
            s=300, lw=1, facecolors='none');
```



In [ ]: