svm_professor

October 21, 2019

1 Support Vector Machines

Authors: Jesús Cid Sueiro (jcid@tsc.uc3m.es)
Jerónimo Arenas García (jarenas@tsc.uc3m.es)

This notebook is a compilation of material taken from several sources:

- The sklearn documentation
- A notebook by Jake Vanderplas
- Wikipedia

Notebook version: 1.0 (Oct 28, 2015)
1.1 (Oct 27, 2016)
2.0 (Nov 2, 2017)
2.1 (Oct 20, 2018)
2.2 (Oct 20, 2019)

Changes:
v.1.0 - First version
v.1.1 - Typo correction and illustrative figures for linear SVM
v.2.0 - Compatibility with Python 3 (backcompatible with Python 2.7)
v.2.1 - Minor corrections on the notation
v.2.2 - Minor equation errors. Reformatted hyperlinks. Restoring broken visualization of image
v.2.3 - Notation revision

In [1]:
```
from __future__ import print_function

# To visualize plots in the notebook
%matplotlib inline

# Imported libraries
#import csv
#import random
#import matplotlib
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
#import pylab
```

```
import numpy as np
#from sklearn.preprocessing import PolynomialFeatures
from sklearn import svm
from sklearn.datasets.samples_generator import make_blobs
from sklearn.datasets.samples_generator import make_circles

from ipywidgets import interact
```

1.1 1. Introduction

[Source: sklearn documentation]

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function.

The disadvantages of support vector machines include:

- SVMs do not directly provide probability estimates.

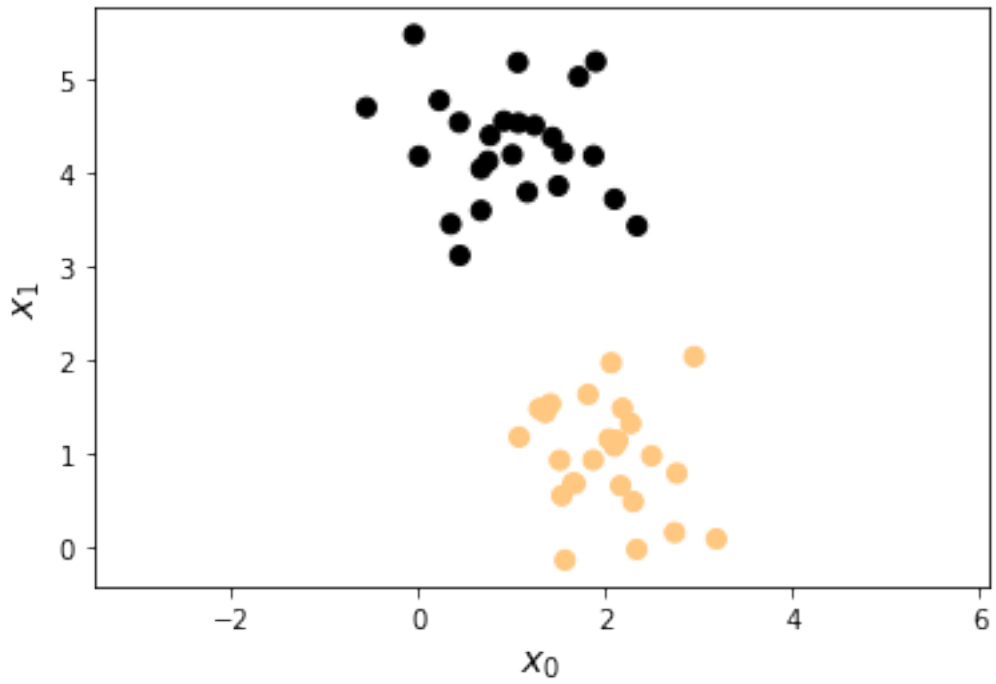
1.2 2. Motivating Support Vector Machines

[Source: A notebook by Jake Vanderplas]

Support Vector Machines (SVMs) are a kind of ***discriminative*** classifiers: that is, they draw a boundary between clusters of data without making any explicit assumption about the probability model underlying the data generation process.

Let's show a quick example of support vector classification. First we need to create a dataset:

```
In [2]: 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='copper')
        plt.xlabel("$x_0$", fontsize=14)
        plt.ylabel("$x_1$", fontsize=14)
        plt.axis('equal')
        plt.show()
```

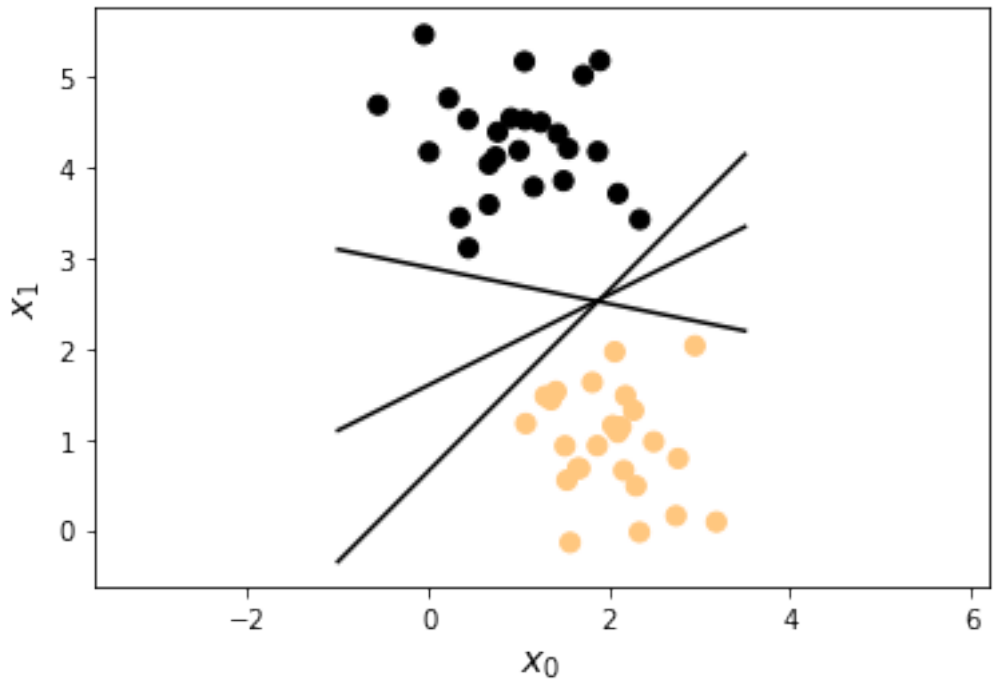


A discriminative classifier attempts to draw a line between the two sets of data. Immediately we see an inconvenience: such problem is ill-posed! For example, we could come up with several possibilities which perfectly discriminate between the classes in this example:

```
In [3]: xfit = np.linspace(-1, 3.5)
        plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='copper')

        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);
        plt.xlabel("$x_0$", fontsize=14)
        plt.ylabel("$x_1$", fontsize=14)
        plt.axis('equal')
        plt.show()
```



These are three very different separators which perfectly discriminate between these samples. Depending on which you choose, a new data point will be classified almost entirely differently! How can we improve on this?

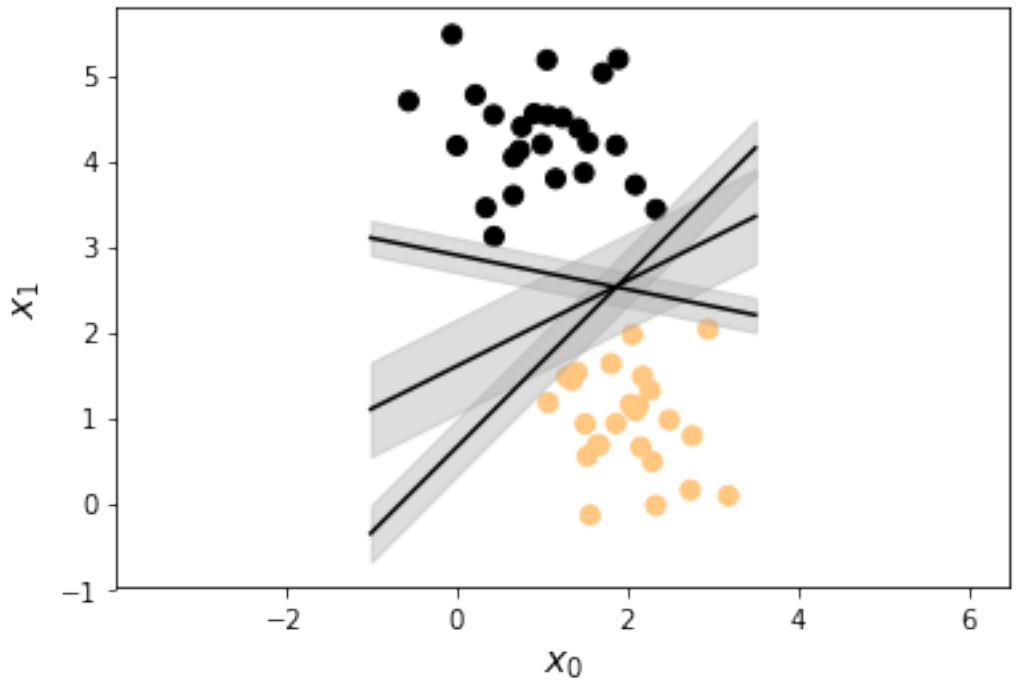
Support Vector Machines (SVM) select the boundary decision maximizing the ***margin***. The margin of a classifier is defined as twice the maximum signed distance between the decision boundary and the training data. By *signed* we mean that the distance to misclassified samples is counted negatively. Thus, if the classification problem is "separable" (i.e. if there exist a decision boundary with zero errors in the training set), the SVM will choose the zero-error decision boundary that is "as far as possible" from the training data.

In summary, what an SVM does is to not only draw a line, but consider the "sample free" region about the line. Here's an example of what it might look like:

```
In [4]: xfit = np.linspace(-1, 3.5)
        plt.scatter(X[:,0], X[:,1], c=y, s=50, cmap='copper')

        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)
        plt.xlabel("$x_0$", fontsize=14)
        plt.ylabel("$x_1$", fontsize=14)
        plt.axis('equal')
        plt.show()
```



Notice here that if we want to maximize this width, the middle fit is clearly the best. This is the intuition of the SVM, which optimizes a linear discriminant model in conjunction with a margin representing the perpendicular distance between the datasets.

1.3 3. Linear SVM

[Source: adapted from Wikipedia]

In order to present the SVM in a formal way, consider a training dataset $\mathcal{D} = \{(\mathbf{x}_k, y_k) \mid \mathbf{x}_k \in \Re^M, y_k \in \{-1, 1\}, k = 0, \ldots, K-1\}$, where the binary symmetric label $y_k \in \{-1, 1\}$ indicates the class to which the point $\mathbf{x}_k$ belongs. Each $\mathbf{x}_k$ is an M-dimensional real vector. We want to find the maximum-margin hyperplane that divides the points having $y_k = 1$ from those having $y_k = -1$.

Any hyperplane can be written as the set of points $\mathbf{x}$ satisfying

$$\mathbf{w}^\intercal \mathbf{x} - b = 0,$$

where $\mathbf{w}$ denotes the (not necessarily normalized) normal vector to the hyperplane. The parameter $\frac{b}{\|\mathbf{w}\|}$ determines the offset of the hyperplane from the origin along the normal vector $\mathbf{w}$.

If the training data are linearly separable, we can select two parallel hyperplanes in a way that they separate the data and there are no points between them, and then try to maximize their distance. The region bounded by them is called "the margin". These hyperplanes can be described by the equations

$$\mathbf{w}^\intercal \mathbf{x} - b = 1$$

and

$$\mathbf{w}^\intercal \mathbf{x} - b = -1.$$

Note that the two equations above can represent any two parallel hyperplanes in $\Re^M$. Essentially, the direction of vector $\mathbf{w}$ determines the orientation of the hyperplanes, whereas parameter b and the norm of $\mathbf{w}$ can be used to select their exact location.

To compute the distance between the hyperplanes, we can obtain the projection of vector $\mathbf{x}_1 - \mathbf{x}_2$, where $\mathbf{x}_1$ and $\mathbf{x}_2$ are points from each of the hyperplanes, onto a unitary vector orthonormal to the hyperplanes:

$$\text{Distance between hyperplanes} = \left[\frac{\mathbf{w}}{\|\mathbf{w}\|}\right]^\top (\mathbf{x}_1 - \mathbf{x}_2) = \frac{\mathbf{w}^\top \mathbf{x}_1 - \mathbf{w}^\top \mathbf{x}_2}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|}.$$

Therefore, to maximize the distance between the planes we want to minimize $\|\mathbf{w}\|$.

As we also have to prevent data points from falling into the margin, we add the following constraints: for each k either

$$\mathbf{w}^\top \mathbf{x}_k - b \geq +1, \qquad \text{if} \quad y_k = 1, \qquad \text{or} \tag{1}$$

$$\mathbf{w}^\top \mathbf{x}_k - b \leq -1, \qquad \text{if} \quad y_k = -1. \tag{2}$$

This can be rewritten as:

$$y_k(\mathbf{w}^\top \mathbf{x}_k - b) \geq 1, \quad \text{for all } 0 \leq k \leq K-1.$$

We can put this together to get the optimization problem:

$$(\mathbf{w}^*, b^*) = \arg\min_{(\mathbf{w},b)} \|\mathbf{w}\| \text{subject to: } y_k(\mathbf{w}^\top \mathbf{x}_k - b) \geq 1, \text{ for any } k = 0, \ldots, K-1$$

This optimization problem is difficult to solve because it depends on $\|\mathbf{w}\|$, the norm of $\mathbf{w}$, which involves a square root. Fortunately it is possible to alter the minimization objective $\|\mathbf{w}\|$ by substituting it with $\frac{1}{2}\|\mathbf{w}\|^2$ (the factor of $\frac{1}{2}$ being used for mathematical convenience) without changing the solution (the minimum of the original and the modified equation have the same $\mathbf{w}$ and b):

$$(\mathbf{w}^*, b^*) = \arg\min_{(\mathbf{w},b)} \frac{1}{2}\|\mathbf{w}\|^2 \text{subject to: } y_k(\mathbf{w}^\top \mathbf{x}_k - b) \geq 1, \text{ for any } k = 0, \ldots, K-1$$

This is a particular case of a *quadratic programming* problem.

1.3.1 3.1. Primal form

The optimization problem stated in the preceding section can be solved by means of a generalization of the Lagrange method of multipliers for inequality constraints, using the so called Karush–Kuhn–Tucker (KKT) multipliers $\boldsymbol{\alpha}$. According to it, the constrained problem can be expressed as

$$(\mathbf{w}^*, b^*, \boldsymbol{\alpha}^*) = \arg\min_{\mathbf{w},b} \max_{\boldsymbol{\alpha} \geq 0} \left\{ \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{k=0}^{K-1} \alpha_k \left[y_k(\mathbf{w}^\top \mathbf{x}_k - b) - 1 \right] \right\}$$

that is, we look for a *saddle point*.