Data Analysis and Machine Learning: Support Vector Machines

Morten Hjorth-Jensen[1,2]

Department of Physics, University of Oslo[1]

Department of Physics and Astronomy and National Superconducting Cyclotron Laboratory, Michigan State University[2]

Nov 5, 2018

Support Vector Machines, overarching aims

A Support Vector Machine (SVM) is a very powerful and versatile Machine Learning model, capable of performing linear or nonlinear classification, regression, and even outlier detection. It is one of the most popular models in Machine Learning, and anyone interested in Machine Learning should have it in their toolbox. SVMs are particularly well suited for classification of complex but small-sized or medium-sized datasets.

The case with two well-separated classes only can be understood in an intuitive way in terms of lines in a two-dimensional space separating the two classes (see figure below).

The basic mathematics behind the SVM is however less familiar to most of us. It relies on the definition of hyperplanes and the definition of a **margin** which separates classes (in case of classification problems) of variables. It is also used for regression problems.

With SVMs we distinguish between hard margin and soft margins. The latter introduces a so-called softening parameter to be discussed below. We distinguish also between linear and non-linear approaches. The latter are the most frequent ones since it is rather

Hyperplanes and all that

The theory behind support vector machines (SVM hereafter) is based on the mathematical description of so-called hyperplanes. Let us start with a two-dimensional case. This will also allow us to introduce our first SVM examples. These will be tailored to the case of two specific classes, as displayed in the figure here.

We assume here that our data set can be well separated into two domains, where a straight line does the job in the separating the two classes. Here the two classes are represented by either crosses or circles.

What is a hyperplane?

The aim of the SVM algorithm is to find a hyperplane in an p-dimensional space, where p is the number of features that distinctly classifies the data points.

In a p-dimensional space, a hyperplane is what we call an affine subspace of dimension of $p-1$. As an example, in two dimension, a hyperplane is simply as straight line while in three dimensions it is a two-dimensional subspace, or stated simply, a plane.

In two dimensions, with the variables x_1 and x_2, the hyperplane is defined as

$$b + w_1 x_1 + w_2 x_2 = 0,$$

where b is the intercept and w_1 and w_2 define the elements of a vector orthogonal to the line $b + w_1 x_1 + w_2 x_2 = 0$. In two dimensions we define the vectors $\hat{x} = [x1, x2]$ and $\hat{w} = [w1, w2]$. We can then rewrite the above equation as

$$\hat{w}^T \hat{x} + b = 0.$$

A *p*-dimensional space of features

We limit ourselves to two classes of outputs y_i and assign these classes the values $y_i = \pm 1$. In a *p*-dimensional space of say *p* features we have a hyperplane defines as

$$b + wx_1 + w_2x_2 + \cdots + w_px_p = 0.$$

If we define a matrix $\hat{X} = [\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_p]$ of dimension $n \times p$, where *n* represents the observations for each feature and each vector x_i is a column vector of the matrix $\hat{X}$,

$$\hat{x}_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ \ldots \\ \ldots \\ x_{ip} \end{bmatrix}.$$

If the above condition is not met for a given vector $\hat{x}_i$ we have

$$b + w_1x_{i1} + w_2x_i2 + \cdots + w_px_{ip} > 0,$$

if our output $y_i = 1$. In this case we say that $\hat{x}_i$ lies on one of the sides of the hyperplane and if

The two-dimensional case

Let us try to develop our intuition about SVMs by limiting ourselves to a two-dimensional plane. To separate the two classes of data points, there are many possible lines (hyperplanes if you prefer a more strict naming) that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

What a linear classifier attempts to accomplish is to split the feature space into two half spaces by placing a hyperplane between the data points. This hyperplane will be our decision boundary. All points on one side of the plane will belong to class one and all points on the other side of the plane will belong to the second class two.

Unfortunately there are many ways in which we can place a hyperplane to divide the data. Below is an example of two candidate hyperplanes for our data sample.

Getting into the details

Let us define the function

$$f(x) = \hat{w}^T \hat{x} + b = 0,$$

as the function that determines the line L that separates two classes (our two features), see the figure here.

Any point defined by $\hat{x}_i$ and $\hat{x}_2$ on the line L will satisfy $\hat{w}^T(\hat{x}_1 - \hat{x}_2) = 0$.

The signed distance δ from any point defined by a vector $\hat{x}$ and a point $\hat{x}_0$ on the line L is then

$$\delta = \frac{1}{||\hat{w}||}(\hat{w}^T \hat{x} + b).$$

First attempt at a minimization approach

How do we find the parameter b and the vector $\hat{w}$? What we could do is to define a cost function which now contains the set of all misclassified points M and attempt to minimize this function

$$C(\hat{w}, b) = - \sum_{i \in M} y_i(\hat{w}^T \hat{x}_i + b).$$

We could now for example define all values $y_i = 1$ as misclassified in case we have $\hat{w}^T \hat{x}_i + b < 0$ and the opposite if we have $y_i = -1$. Taking the derivatives gives us

$$\frac{\partial C}{\partial b} = - \sum_{i \in M} y_i,$$

and

$$\frac{\partial C}{\partial \hat{w}} = - \sum_{i \in M} y_i x_i.$$

Solving the equations

We can now use the Newton-Raphson method or gradient descent to solve the equations

$$b \leftarrow b + \eta \frac{\partial C}{\partial b},$$

and

$$\hat{w} \leftarrow \hat{w} + \eta \frac{\partial C}{\partial \hat{w}},$$

where η is our by now well-known learning rate.

There are however problems with this approach, although it looks pretty straightforward to implement. In case we separate our data into two distinct classes, we may up with many possible lines, as indicated in the figure and shown by running the following program. For small gaps between the entries, we may also end up needing many iterations before the solutions converge and if the data cannot be separated properly into two distinct classes, we may not experience a converge at all.

A better approach

A better approach is rather to try to define a large margin between the two classes (if they are well separated from the beginning). Thus, we wish to find a margin M with $\hat{w}$ normalized to $||\hat{w}|| = 1$ subject to the condition

$$y_i(\hat{w}^T \hat{x}_i + b) \geq M \forall i = 1, 2, \ldots, p.$$

All points are thus at a signed distance from the decision boundary defined by the line L. The parameters b and w_1 and w_2 define this line.

We seek thus the largest value M defined by

$$\frac{1}{||\hat{w}||} y_i(\hat{w}^T \hat{x}_i + b) \geq M \forall i = 1, 2, \ldots, n,$$

or just

$$y_i(\hat{w}^T \hat{x}_i + b) \geq M ||\hat{w}|| \forall i.$$

If we scale the equation so that $||\hat{w}|| = 1/M$, we have to find the minimum of $\hat{w}^T \hat{w} = ||\hat{w}||$ (the norm) subject to the condition

$$y_i(\hat{w}^T \hat{x}_i + b) \geq 1 \forall i.$$

A quick reminder on Lagrangian multipliers

Consider a function of three independent variables $f(x, y, z)$. For the function f to be an extreme we have

$$df = 0.$$

A necessary and sufficient condition is

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} = \frac{\partial f}{\partial z} = 0,$$

due to

$$df = \frac{\partial f}{\partial x} dx + \frac{\partial f}{\partial y} dy + \frac{\partial f}{\partial z} dz.$$

In many problems the variables x, y, z are often subject to constraints (such as those above for the margin) so that they are no longer all independent. It is possible at least in principle to use each constraint to eliminate one variable and to proceed with a new and smaller set of independent varables.

The use of so-called Lagrangian multipliers is an alternative technique when the elimination of variables is incovenient or undesirable. Assume that we have an equation of constraint on the variables x, y, z

Adding the muliplier

However, we can add to

$$df = \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy + \frac{\partial f}{\partial z}dz,$$

a multiplum of $d\phi$, viz. $\lambda d\phi$, resulting in

$$df + \lambda d\phi = (\frac{\partial f}{\partial z} + \lambda\frac{\partial \phi}{\partial x})dx + (\frac{\partial f}{\partial y} + \lambda\frac{\partial \phi}{\partial y})dy + (\frac{\partial f}{\partial z} + \lambda\frac{\partial \phi}{\partial z})dz = 0.$$

Our multiplier is chosen so that

$$\frac{\partial f}{\partial z} + \lambda\frac{\partial \phi}{\partial z} = 0.$$

We need to remember that we took dx and dy to be arbitrary and thus we must have

$$\frac{\partial f}{\partial x} + \lambda\frac{\partial \phi}{\partial x} = 0,$$

and

$$\frac{\partial f}{\partial y} + \lambda\frac{\partial \phi}{\partial y} = 0.$$

When all these equations are satisfied, $df = 0$. We have four unknowns, x, y, z and λ. Actually we want only x, y, z, λ needs not

Setting up the problem

In order to solve the above problem, we define the following Lagrangian function to be minimized

$$\mathcal{L}(\lambda, b, \hat{w}) = \frac{1}{2}\hat{w}^T\hat{w} - \sum_{i=1}^{n} \lambda_i \left[y_i(\hat{w}^T\hat{x}_i + b) - 1 \right],$$

where λ_i is a so-called Lagrange multiplier subject to the condition $\lambda_i \geq 0$.

Taking the derivatives with respect to b and $\hat{w}$ we obtain

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum_i \lambda_i y_i = 0,$$

and

$$\frac{\partial \mathcal{L}}{\partial \hat{w}} = 0 = \hat{w} - \sum_i \lambda_i y_i \hat{x}_i.$$

Inserting these constraints into the equation for $\mathcal{L}$ we obtain

$$\mathcal{L} = \sum_i \lambda_i - \frac{1}{2} \sum_{ij}^{n} \lambda_i \lambda_j y_i y_j \hat{x}_i^T \hat{x}_j,$$

subject to the constraints $\lambda \geq 0$ and $\sum_i \lambda_i y_i = 0$. We must in

The problem to solve

We can rewrite

$$\mathcal{L} = \sum_i \lambda_i - \frac{1}{2} \sum_{ij}^n \lambda_i \lambda_j y_i y_j \hat{x}_i^T \hat{x}_j,$$

and its constraints in terms of a matrix-vector problem where we minimize w.r.t. λ the following problem

$$\frac{1}{2} \hat{\lambda}^T \begin{bmatrix} y_1 y_1 \hat{x}_1^T \hat{x}_1 & y_1 y_2 \hat{x}_1^T \hat{x}_2 & \dots & \dots & y_1 y_n \hat{x}_1^T \hat{x}_n \\ y_2 y_1 \hat{x}_2^T \hat{x}_1 & y_2 y_2 \hat{x}_2^T \hat{x}_2 & \dots & \dots & y_1 y_n \hat{x}_2^T \hat{x}_n \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ y_n y_1 \hat{x}_n^T \hat{x}_1 & y_n y_2 \hat{x}_n^T \hat{x}_2 & \dots & \dots & y_n y_n \hat{x}_n^T \hat{x}_n \end{bmatrix} \hat{\lambda} - \not{\Bbbk} \hat{\lambda},$$

subject to $\hat{y}^T \hat{\lambda} = 0$.

Examples with kernels

```python
from IPython.display import set_matplotlib_formats, display
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import mglearn
from cycler import cycler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.datasets import make_blobs


X, y = make_blobs(centers=4, random_state=8)
y = y % 2

mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
plt.show()

from sklearn.svm import LinearSVC
linear_svm = LinearSVC().fit(X, y)

mglearn.plots.plot_2d_separator(linear_svm, X)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
# add the squared first feature
X_new = np.hstack([X, X[:, 1:] ** 2])
```