Support vector machine with weight constraints or no bias

Man Yi Yim
manyi.yim@gmail.com
January 9, 2020

Introduction

This note presents the numerical approach to implement the non-standard support vector machine (SVM) with weight constraints or no bias, which arises for some constrained learning and classification problems. I will start with an introduction to the standard SVM and then describe the formalism of the non-standard SVMs.

Formalism of standard SVM

Support vector machine (SVM) is a binary classifier with optimal hyperplane that separates the two classes of linear separable patterns. Support vectors are the patterns that lie closest to the hyperplane and are thus most difficult to classify. Maximum margin κ is defined as two times the shortest distance from the closest patterns to the optimal hyperplane. Let $\mathbf{x}_i \in \mathbb{R}^{\mathbb{N}}, i = 1, \ldots, P$, be the input patterns and $\mathbf{y} \in [-1, 1]^P$ be the labels for the P patterns. Also, let $\mathbf{w}$ and b be the optimal weights and bias to achieve the maximum margin κ. For any pattern i, we have, for some $\epsilon \in \mathbb{R}$,

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq \epsilon. \tag{1}$$

Let $\mathbf{x}_+$ and $\mathbf{x}_-$ be the support vectors on the opposite sides of the hyperplane. Since Eq. 1 is linear, without loss of generality, we set ϵ to 1. The margin κ is given by

$$\begin{aligned} \kappa &= ||\mathbf{e}_w \cdot \mathbf{x}_+ - \mathbf{e}_w \cdot \mathbf{x}_-|| \\ &= \frac{||(1-b) - (-1-b)||}{||\mathbf{w}||} \\ &= \frac{2}{||\mathbf{w}||} \end{aligned} \tag{2}$$

where $\mathbf{e}_w = \mathbf{w}/||\mathbf{w}||$ is the unit vector of $\mathbf{w}$. The margin can be maximized by minimizing $||\mathbf{w}||$ using quadratic programming. We minimize

$$\frac{1}{2}||\mathbf{w}||^2 \tag{3}$$

subject to the P constraints

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \geq 0, \forall i = 1, \ldots, P. \tag{4}$$

This is a constrained optimization problem that can be solved using the Lagrangian multipler method. Because it is quadratic, the surface is a paraboloid with a single global minimum. The Langrangian is given by

$$L(\mathbf{w}, b) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{P} \alpha_i [y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1]. \tag{5}$$

Differentiating the Langrangian with respect to $\mathbf{w}$ and b yields

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{P} \alpha_i y_i \mathbf{x}_i \quad \Rightarrow \quad \mathbf{w} = \sum_{i=1}^{P} \alpha_i y_i \mathbf{x}_i \tag{6}$$

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{P} \alpha_i y_i \quad \Rightarrow \quad \sum_{i=1}^{P} \alpha_i y_i = 0. \tag{7}$$

The Lagrangian dual problem can be applied here by rewriting the Langrangian as

$$L(\alpha_i) = \sum_{i=1}^{P} \alpha_i - \frac{1}{2} \sum_{i=1}^{P} \sum_{j=1}^{P} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j. \tag{8}$$

Differentiating the Langrangian with respect to α_j gives

$$\frac{\partial L}{\partial \alpha_j} = 1 - \sum_{i=1}^{P} \alpha_i y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \quad \Rightarrow \quad \sum_{i=1}^{P} \alpha_i y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j = 1. \tag{9}$$

We can solve for all $\alpha_j, j = 1, \ldots, P$, and thus $\mathbf{w}$ and b.

Standard SVM using sklearn

For the above implementation of SVM using sklearn, the full documentation can be found below: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

```
def svm(X,Y):
        w = np.zeros(X.shape[0])
        from sklearn import svm
        hyp = svm.SVC(kernel='linear',C=10000,cache_size=20000,tol=1e-5)
        hyp.fit(X.T,Y)
        b = hyp.intercept_[0]
        for j in range(hyp.support_.size):
                w += hyp.dual_coef_[0][j]*hyp.support_vectors_[j]
        if abs(np.sum(np.abs(Y-dec))) == 0:
                return w,b,2./pylab.norm(w)
```

SVM using quadratic programming

For more flexible implementation of SVM, the full documentation can be found below: https://pypi.org/project/qpsolvers/. The sections below require the use of quadratic programming.

SVM with no bias

If neurons or nodes receive no external input, the bias is zero. To implement SVM with no bias, we can simply discard all the terms with bias b in the above formulation. That is, we minimize $\frac{1}{2}||\mathbf{w}||^2$ subject to the P constraints

$$y_i(\mathbf{w} \cdot \mathbf{x}_i) - 1 \geq 0, \forall i = 1, \ldots, P. \tag{10}$$

SVM with weight constraints

We consider non-negativity constraints here only. This constraint is needed in neural models when excitatory (or inhibitory) neurons are considered. Other kinds of weight and bias contraints can be incorporated by modifying the formulation below. Again, we minimize $\frac{1}{2}||\mathbf{w}||^2$ subject to the $P + N$ constraints (P existing and N additional)

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \geq 0, \forall i = 1, \ldots, P \tag{11}$$

and

$$\mathbf{w} \geq \mathbf{0}, \forall i = 1, \ldots, N. \tag{12}$$

The general purpose implementation of SVM:

```
def svm_qp(x,y,is_bias=1,is_wconstrained=1):
        import qpsolvers
        R = x.shape[1]
        G = -(x*y).T
        if is_bias:
                N = x.shape[0] + 1
                G = np.append(G.T,-y)
                G = G.reshape(N,R)
                G = G.T
                P = np.identity(N)
                P[-1,-1] = 1e-12    # regularization
        else:
                N = x.shape[0]
                P = np.identity(N)
        if is_wconstrained:
                if is_bias:
                        G = np.append(G,-np.identity(N)[:N-1,:])
                        G = G.reshape(R+N-1,N)
                        h = np.array([-1.]*R+[0]*(N-1))
                else:
                        G = np.append(G,-np.identity(N))
                        G = G.reshape(R+N,N)
                        h = np.array([-1.]*R+[0]*N)
        else:
                h = np.array([-1.]*R)
                w = qpsolvers.solve_qp(P,np.zeros(N),G,h)
        if is_bias:
                return w[:-1],w[-1],2/pylab.norm(w[:-1])
        else:
                return w,2/pylab.norm(w)
```