SVM Classifier

Bouh Ivan

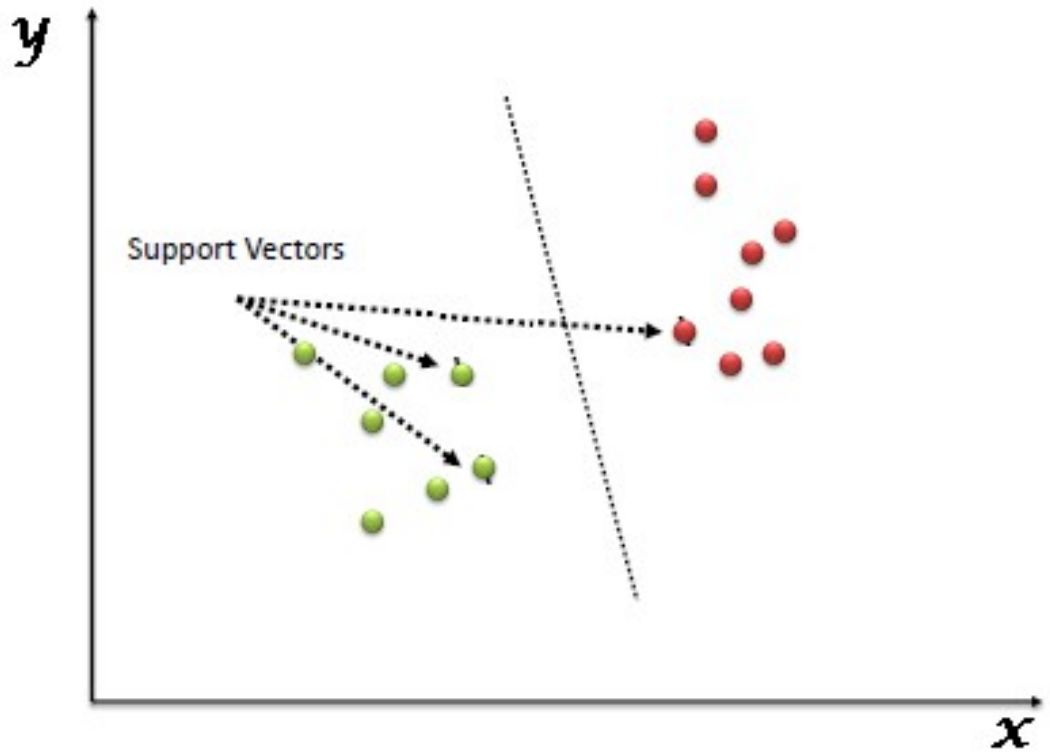
January 10, 2022

Contents

1 Introduction

.

In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is a number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well.



The hyperplane is our decision boundary. Everything on one side belongs to one class, and everything on the other side belongs to a different class.

2 Formulation

We'd like to find the point closest to the separating hyperplane and make sure this is as far away from the separating line as possible. This is known as margin. We want to have the greatest possible margin, because if we made a mistake or trained our classifier on limited data, we'd want it to be as robust as possible. The equation of the hyperplane is of the form

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

Where $\phi(x)$ denotes a feature-space transformation function and b bias parameter We want to find the best hyperplane (that maximises the distance between the closest data points and the hyperplane) that 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.

The perpendicular distance of a point x from a hyperplane defined by $y(x) = 0$ where $y(x)$ is given by $|y(x)|/||w||$ Furthermore, we are only interested in solutions for which all data points are correctly classified, so that $t_n y(x_n) > 0$ for all n. Thus the distance

of a point x_n to the decision surface is given by

$$\frac{t_n y(x_n)}{||w||} = \frac{t_n(w^T(x_n) + b)}{||w||}$$

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

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

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 $w \to kw$ and $b \to kb$, then the distance from any point x_n to the decision surface, given by $t_n y(x_n)/||w||$, 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. In this case, all data points will satisfy the constraints

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

In the case of data points for which the equality holds, 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

$$arg\, min_{w,b} \frac{1}{2}||w||^2$$

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

$$L(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{n=1}^{N} \alpha_n [t_n(w^T X_n + b) - 1]$$

where

$$\alpha = (\alpha_1, ..., \alpha_N)^T$$

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

$$w = \sum_{n=1}^{N} \alpha_n t_n x_n$$

$$\sum_{n=1}^{N} \alpha_n t_n = 0$$

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

$$\widetilde{L}(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m t_n t_m k(X_n, X_m)$$

with respect to a subject to the constraints

$$\alpha_n \geq 0, n = 1, ..., N,$$

$$\sum_{n=1}^{N} \alpha_n t_n = 0.$$

Here k is the kernel function representing the dot product in the target dimesion, in our case the original dimension so $k(x, x') = \phi(x)^T \phi(x')$. The KKT conditions can be used to check for convergence to the optimal point. For this problem the KKT conditions are

$$\alpha_i = 0 \Rightarrow y^{(i)}(w^T x^{(i)} + b) \geq 1$$

$$\alpha_i = C \Rightarrow y^{(i)}(w^T x^{(i)} + b) \leq 1$$

$$0 < \alpha_i < C \Rightarrow y^{(i)}(w^T x^{(i)} + b) = 1$$

In other words, any α's that satisfy these properties for all i will be an optimal solution to the optimization problem given above.

3 Algorithm

To solve the above optimisation problem multiple optimisers exist but a principal algorithm SMO introduced as a faster optimization algorithm for SVM that works by optimizing pairs of alpha's ans updating bs accordingly until no further optimization is possible.

3.1 The Simplified SMO Algorithm

the SMO algorithm selects two α parameters, α_i and α_j and optimizes the objective value jointly for both these α's. Finally it adjusts the b parameter based on the new α's. This process is repeated until the α's converge. We now describe these three steps in greater detail.

3.1.1 Selecting α Parameters

Much of the full SMO algorithm is dedicated to heuristics for choosing which α_i and α_j to optimize so as to maximize the objective function as much as possible. For large data sets, this is critical for the speed of the algorithm, since there are $m(m-1)$ possible choices for α_i and α_j, and some will result in much less improvement than others.

However, for our simplified version of SMO, we employ a much simpler heuristic. We simply iterate over all α_i, i = 1, . . . m. If α_i does not fulfill the KKT conditions to within some numerical tolerance, we select α_j at random from the remaining $m - 1\alpha_i$'s and attempt to jointly optimize α_i and α_j. If none of the α's are changed after a few iteration over all the α_i's, then the algorithm terminates. It is important to realize that by employing this simplification, the algorithm is no longer guaranteed to converge to the global optimum (since we are not attempting to optimize all possible α_i, α_j pairs, there exists the possibility that some pair could be optimized which we do not consider).

3.1.2 Optimizing α_i and $alpha_j$

Having chosen the Lagrange multipliers α_i and α_j to optimize, we first compute constraints on the values of these parameters, then we solve the constrained maximization problem. First we want to find bounds L and H such that $L \leq \alpha_j \leq H$ must hold in order for α_j to satisfy the constraint that $0 \leq \alpha_j \leq C$. It can be shown that these are given by the following:

- If $y^{(i)} \neq y^{(j)}, L = max(0, \alpha_j - \alpha_i), \quad H = min(C, C + \alpha_j - \alpha_i)$ (1)
- If $y^{(i)} = y^{(j)}, L = max(0, \alpha_i + \alpha_j - C), \quad H = min(C, \alpha_i + \alpha_j)$ (2)

Now we want to find α_j so as to maximize the objective function. If this value ends up lying outside the bounds L and H, we simply clip the value of α_j to lie within this range. It can be shown that the optimal α_j is given by:

$$\alpha_j := \alpha_j - \frac{y(j)(E_i - E_j)}{\eta} \quad (3)$$

where

$$E_k = f(x^{(k)}) - y^{(k)} \quad (4)$$

$$\eta = 2k(x^{(i)}, x^{(j)}) - k(x^{(i)}, x^{(i)}) - k(x^{(j)}, x^{(j)}) \quad (5)$$

You can think of E_k as the error between the SVM output on the kth example and the true label $y^{(k)}$. Next we clip α_j to lie within the range [L, H]

$$\alpha_j := \begin{cases} H \; if \; \alpha_j > H \\ \alpha_j \; if \; L \leq \alpha_j \leq H \\ L \; if \; \alpha_j < L \end{cases} \quad (6)$$

Finally, having solved for $alpha_j$ we want to find the value for α_i. This is given by

$$\alpha_i := \alpha_i + y^{(i)}y^{(j)}(\alpha_j^{(old)} - \alpha_j) \quad (7)$$

3.1.3 Computing the b threshold

After optimizing α_i and α_j, we select the threshold b such that the KKT conditions are satisfied for the ith and jth examples. If, after optimization, α_i is not at the bounds

(i.e., $0 < \alpha_i < C$), then the following threshold b1 is valid, since it forces the SVM to output $y^{(i)}$ when the input is $x^{(i)}$

$$b_1 = b - E_i - y^{(i)}(\alpha_i - \alpha_i^{(old)})k(x^{(i)}, x^{(j)}) - y^{(j)}(\alpha_j - \alpha_j^{(old)})k(x^{(i)}, x^{(j)}) \quad (8)$$

Similarly, the following threshold b_2 is valid if $0 < \alpha_j < C$

$$b_2 = b - E_j - y^{(i)}(\alpha_i - \alpha_i^{(old)})k(x^{(i)}, x^{(j)}) - y^{(j)}(\alpha_j - \alpha_j^{(old)})k(x^{(i)}, x^{(j)}) \quad (9)$$

If both $0 < \alpha_i < C$ and $0 < \alpha_j < C$ then both these thresholds are valid, and they will be equal. If both new α's are at the bounds (i.e., $\alpha_i = 0$ or $\alpha_i = C$ and $\alpha_j = 0$ or $\alpha_j = C$) then all the thresholds between b_1 and b_2 satisfy the KKT conditions, we let $b := (b_1 + b_2)/2$.

3.1.4 Pseudo-Code for Simplified SMO

In this section we present pseudo-code for the simplified SMO algorithm.
Algorithm: Simplified SMO
Input:
C: regularization parameter
tol: numerical tolerance
max_passes: max # of times to iterate over α's without changing
$(x^{(1)}, y^{(1)}), ..., (x^{(m)}, y^{(m)})$: training data
Output:
$\alpha \in \mathbb{R}^m$ Lagrange multipliers for solution
$b \in \mathbb{R}$ threshold for solution

```
Initialize αᵢ = 0, ∀i, b = 0.
Initialize passes = 0.
while(passes < max_passes)
        num changed alphas = 0.
        for i = 1, . . . m,
                Calculate E_i = f(x^(i)) − y^(i)
                if((y^(i)E_i < −tol  &&  α_i < C)  ||  (y^(i)E_i > tol  &&  α_i > 0))
                        Select j ≠ i randomly.
                        Calculate E_j = f(x^(j)) − y^(j)
                        Save old α's: (old)_i = α_i, (old)_j = α_j.
                        Compute L and H
                        if(L == H)
                                continue to next i.
                        Compute η
                        if(η >= 0)
                                continue to next i.
                        Compute and clip new value for α_j
                        if(|α_j − α_j^(old)| < 10^−5)
                                continue to next i.
                        Determine value for α_i.
                        Compute b_1 and b_2
                        Compute b
                        num_changed_alphas := num_changed_alphas + 1.
                endif
        endfor
        if (num_changed_alphas == 0)
                passes := passes + 1
        else
                passes := 0
endwhile
```