Training Support Vector Machines with Weighted Reduced Convex Hulls

Mohammed Modan
Department of Mathematics, Statistics, and Computer Science
Macalester College
St. Paul, MN 55105

December 20, 2017

Abstract

Support Vector Machines have long been inaccessible for beginners beyond the linearly separable case. The geometric interpretation of SVM trainers affords a conceptually simple introduction to the subject, while still maintaining competitive accuracy and speed. In this paper, I describe the application and implementation of weighted reduced convex hulls in support vector machine classifiers.

1 Introduction

Machine learning has come to run the world. From recommender systems to predictive analytics, machine learning is integrated into almost every interaction we make with technology. Machine learning can be either supervised or unsupervised. In the latter case, the algorithm is not told what to look for and discovers patterns in the data. For the former, data is preclassified and the algorithm can find patterns associated with the class provided. These algorithms, broadly, fall into either numeric predictors or classifiers. At a basic level, they operate as follows:

- Train a model on input data with a known output
- The model can now predict an output from a new input

Algorithms can range from the relatively simple, like Naive Bayes, to the complicated and cutting edge, like neural networks. Each algorithm has its place and application in which the model will excel, and engineers/data scientists will capitalize on this fact to build their software accordingly to exploit the strengths of these algorithms. For example, Naive Bayes is frequently used for sentiment analysis or spam filtering. Support Vector Machines (SVMs) are commonly used in hand writing recognition and protein classification due to its versatility.

SVMs typically perform very well while simultaneously avoiding overfitting data. Unfortunately, SVMs come with their drawbacks as well. They may not perform as well with very noisy data, and along with other complex machine learning algorithms such as neural networks, SVMs can be difficult to interpret and they are often seen as a "black box." When explaining the algorithm is necessary for a client, simpler algorithms such as decision trees or logistic regression are typically used which are simple to interpret.

2 What is an SVM?

A Support Vector Machine (SVM) is indeed a type of machine learning algorithm. In order to understand their method of function, however, it is necessary to understand the format of the data. Columns of a dataset can be considered variables, and rows as observations. Each row additionally has an associated class. Each row can be viewed as a point with an associated class in $\mathbb{R}^v$ where v equals the number of variables. For example consider the `iris` dataset in R (Table 1). Each row is a point in $\mathbb{R}^2$.

Petal Length	Petal Width	Species
1.4	0.2	setosa
1.4	0.3	setosa
1.3	0.2	setosa
4.7	1.4	versicolor
4.5	1.5	versicolor
4.9	1.5	versicolor

Table 1: The `iris` dataset in $\mathbb{R}^2$

Notice that these points can be plotted in $\mathbb{R}^2$, coloring them by an arbitrarily assigned positive or negative class. Observe that this may give rise to class separation in the Cartesian plane. As illustrated in Figure 1, a vector w can then span between convex hulls of the positive and negative classes. The margin is defined as the norm of w, and the hyperplane separating the hulls is the perpendicular bisector of w. Note that an infinite number of w can be drawn to separate the data, however there exists one unique hyperplane which maximizes the margin and maximizes separation. This occurs when w spans the closest points between the convex hulls, as it does in Figure 1. This w defines the maximum margin and therefore maximally separating hyperplane.

Therefore we can define an SVM as follows:

Definition 2.1. *An* SVM *is a classifier defined by a maximally separating hyperplane.*

Additionally, note that once the hyperplane is found, predictions are quite simple. $w \cdot x + b < 0$ defines the negative class, $w \cdot x + b > 0$ defines the positive class, and $w \cdot x + b = 0$ defines the hyperplane itself.

3 Defining the Optimization Problem

While there are several approaches in finding the maximum margin hyperplane, we will focus on the entirely geometric approach, which relies on an iterative, interpretable algorithm. Other methods involve solve quadratic programs, which lose the geometric elegance of the solution.

First, let us mathematically outline the problem we are trying to solve. Given a set of points $P = \{x_1, x_2, ..., x_n | x_i \in \mathbb{R}^d\}$ and their associated labels $Y = \{y_1, y_2, ..., y_n | y_i \in \{1, -1\}\}$, we can write the equation of the convex hull as:

$$CH(P) = \left\{ \sum_{i=1}^{n} \alpha_i x_i \;\middle|\; x_i \in P,\ \sum_{i=1}^{n} \alpha_i = 1,\ \ 0 \leq \alpha_i \leq 1 \right\} \tag{3.1}$$

Where finding the maximum margin is equivalent to solving the minimization task given by 3.2 and 3.3 [1].

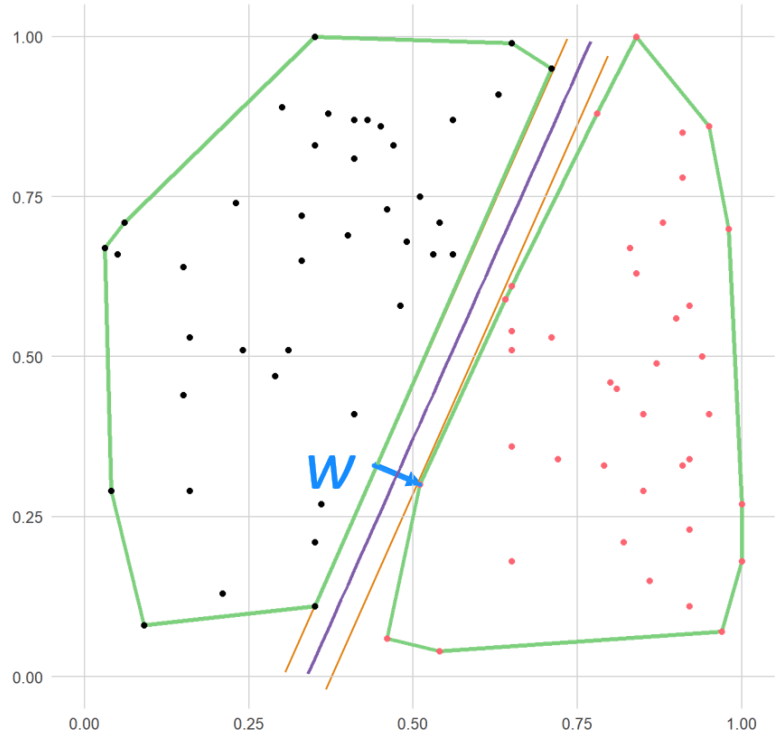

Figure 1: A dataset in $\mathbb{R}^v$ with associated positive (salmon) and negative (black) classes. w indicates the vector spanning the shortest distance between convex hulls of the classes (green), pointing to the positive class, defining the maximum margin (orange) hyperplane (purple) as its perpendicular bisector

$$\text{minimize} \quad \sum_{i,j} y_i y_j \alpha_i \alpha_j \mathbf{x_i^T x_j}, \tag{3.2}$$

$$\text{subject to} \quad \sum_{i=1}^{n} \alpha_i y_i = 0, \quad \sum_{i=1}^{n} \alpha_i = 2, \quad \alpha_i \geq 0 \tag{3.3}$$

An iterative algorithm to find this optimal solution was found by Kozinec, and later improved by Schlesinger to converge at an ϵ-optimal solution for time efficiency [2].

4 The Schlesinger-Kozinec Algorithm

The Schlesinger-Kozinec (SK) algorithm maintains the geometric view of SVMs, converges very fast, and scales incredibly well [3]. With this iterative approach, the algorithm essentially needs to "check" points on the convex hulls, but does not actually have to compute the entire hull. This allows the algorithm to rapidly scale to higher dimensions without increasing time complexity. This efficient aspect of the algorithm relies on a `findVertex` function, as outlined in Algorithm 1 and in Figure 2. This algorithm will find the point on the CH of P in the direction of $\mathbf{n}$.

Now, given a positive and negative point set, we can work with two candidate nearest points, $\mathbf{p_{pos}}$ and $\mathbf{p_{neg}}$, we can find w as equation 4.1.

Algorithm 1 Finding a CH vertex

1: **function** FINDVERTEX($P, \mathbf{n}$)
2: $a \leftarrow \arg\min_i\{\mathbf{n} \cdot \mathbf{x_i}\}$
3: **return** $\mathbf{x_i}$

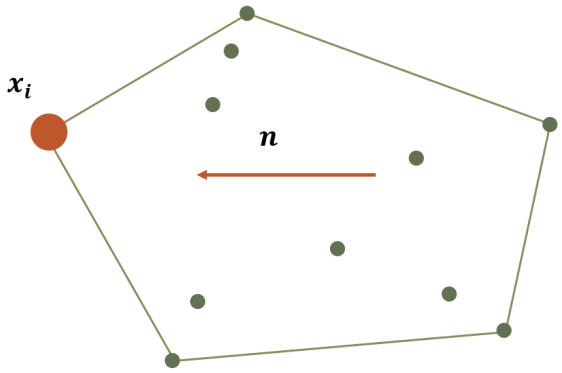

Figure 2: the findVertex routine will find the vertex x_i on the convex hull of P in the direction of $\mathbf{n}$

$$w = p_{pos} - p_{neg} \tag{4.1}$$

With each iteration of the algorithm, the candidate points will move closer and closer to the true closest points of the convex hulls, eventually defining the optimal $\mathbf{w}$ and hyperplane.

4.1 Initialization

Thanks to Goodrich [3], a thorough understanding of this algorithm has been made possible. While Goodrich initializes p_{pos} and p_{neg} as any random point on their respective hulls, I found this to lead to wildly inconsistent training times. Instead, I propose initialization based on the centroids of each point set, which can be calculated by averaging all points. Thereafter, p_{pos} can be initialized as `findVertex`$(center_{neg} - center_{pos})$ and p_{neg} can be initialized as `findVertex`$(center_{pos} - center_{neg})$. This ensures that the starting points will at least initialize on at least the correct half of the convex hulls.

Additionally, update vertices v_{pos} and v_{neg} are initialized as `findVertex`$(-w)$ and `findVertex`(w) respectively. These will be used to updated p_{pos} and p_{neg}.

4.2 Stopping Conditions

If the new v_{pos} and v_{neg} are within ϵ to p_{pos} and p_{neg}, the algorithm is terminated. This is checked confirming that both 4.2 and 4.3 are satisfied.

$$1 - \frac{w \cdot (v_{pos} - p_{neg})}{||w||^2} < \epsilon \tag{4.2}$$

$$\texttt{and} \quad 1 - \frac{w \cdot (v_{neg} - p_{pos})}{||w||^2} < \epsilon \tag{4.3}$$

Essentially, ϵ optimality is reached with the w created by the update vertex and the old candidate is close enough to w created by the candidate points.

4.3 Adaptation

First, choose to either update the positive or negative class. If $w \cdot (p_{pos} - v_{pos}) > w \cdot (p_{neg} - v_{neg})$, then there is more room for potential progress to be made on the positive hull so p_{pos} is updated. Otherwise, p_{neg} is updated.

p_{pos} is updated by finding the point on the line segment created by p_{pos} and v_{pos} that is closest to p_{neg}. This happens to be at the point where w is orthogonal to $p_{pos} - v_{pos}$ and connects via p_{neg}. This is graphically illustrated in Figure 3. p_{pos}^{new} can then be found from Equation 4.4.

$$p_{pos}^{new} = (1-q)p_{pos} + qv_{pos}\ , \quad q \in [0,1] \tag{4.4}$$

Where q is found such that $w^{new} \cdot (p_{pos} - v_{pos}) = 0$ to ensure they are orthogonal. q is clamped between 0 and 1 to ensure the new point lies between the candidate and update points. Substituting 4.4 and p_{neg} for w^{new} gives rise to Equation 4.5 where q can be solved for to yield 4.6.

$$0 = \Big(p_{neg} - ((1-q)p_{pos} + qv_{pos})\Big) \cdot \Big(p_{pos} - v_{pos}\Big) \tag{4.5}$$

$$q = \frac{(p_{pos} - p_{neg}) \cdot (p_{pos} - v_{pos})}{(p_{pos} - v_{pos})^2} \tag{4.6}$$

p_{neg} is updated similarly, with Equations 4.7 and 4.8.

$$p_{neg}^{new} = (1-q)p_{neg} + qv_{neg}\ , \quad q \in [0,1] \tag{4.7}$$

$$q = -\frac{(p_{pos} - p_{neg}) \cdot (p_{neg} - v_{neg})}{(p_{neg} - v_{neg})^2} \tag{4.8}$$

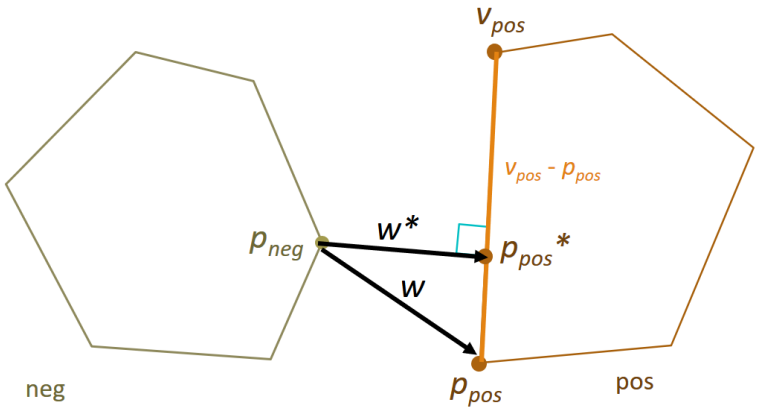

Figure 3: A new p_{pos} is found such that it minimizes distance to p_{neg}. This requires that the new w be orthogonal to $p_{pos} - v_{pos}$.

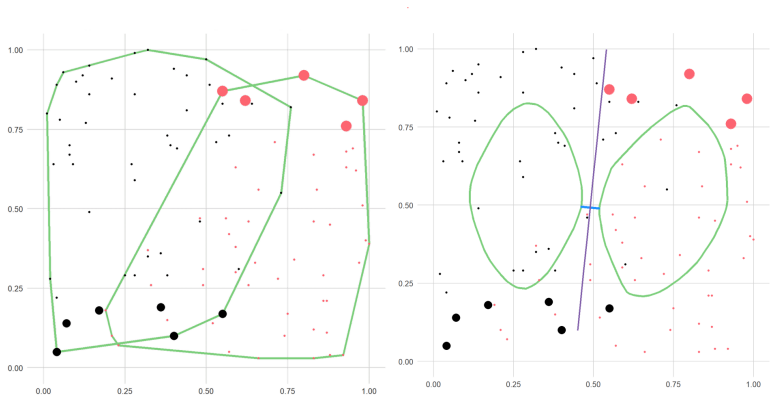

Figure 4: Left: $\mu = 1$, non-linearly separable hulls. Right: Weighted reduction to $\mu = 1/30$. Linearly separable hulls, SK proceeds as normal

5 Weighted Convex Hull Reduction

Notice that up to now, only the linearly separable case has been considered for points set. However, most data cannot be linearly separated. To maintain the geometric beauty of the algorithm, consider a reduction of the convex hull for each class. In addition, consider that it would be possible to reduce the convex hull based on the weight of each point.

Franc and Hlavac [2] explored this idea and adapted the SK algorithm accordingly. In addition, Goodrich [3] introduced individual point weights to the SK algorithm. In fact, the algorithm remains the same, only the `findVertex` function changes to account for the new type of hull, taking in two new parameters as well: μ and s, corresponding to the reduction factor and weights respectively. The new `findVertex` is discussed further in [3]. For my implementation, the centroids of the hulls are now weighted averages of each point.

For a weighted reduced convex hull (WRCH), the definition changes only slightly to yield Equation 5.1

$$WRCH(P, s, \mu) = \left\{ \sum_{i=1}^{n} \alpha_i x_i \;\middle|\; x_i \in P,\ \sum_{i=1}^{n} \alpha_i = 1,\ \ 0 \leq \alpha_i \leq s_i \mu \right\} \tag{5.1}$$

Here, $s \in \mathbb{R}^n$, a vector of weights for each point. All that has changed is the upper bound on α_i to be reduced by a constant and weight. The minimization problem then becomes:

$$\text{minimize} \quad \sum_{i,j} y_i y_j \alpha_i \alpha_j \mathbf{x_i^T x_j}, \tag{5.2}$$

$$\text{subject to} \quad \sum_{i=1}^{n} \alpha_i y_i = 0,\ \ \sum_{i=1}^{n} \alpha_i = 2,\ \ 0 \leq \alpha_i \leq s_i \mu \tag{5.3}$$

This new minimization problem allows for linear separability that can also account for weighted points, as demonstrated in Figure 4.

6 Tests

After implementing the weighted SK algorithm described by Goodrich [3], I tested the data on two datasets from the UCI machine learning repository after normalization - `wine` and `heart`. Classifying wine 1 from wines 2 and 3 from alcohol content, flavinoid content, and magnesium content yielded an astonishing 94.3% accuracy, completing in just 5 iterations, very quick considering each loop runs with O(n) time complexity.

A preliminary test on the `heart` dataset, considering all variables, yielded 77.4% accuracy in predicting which patients had heart disease, but 13% of cases were false negatives. To remedy this, weighing the positive for disease class 5x more increased accuracy to 78.5% and decreased false negatives to only 3%.

It is important to note that for all tests,ϵ can vary depending on the required accuracy, but as per instructions in Goodrich [3], $\mu = \frac{1}{0.9\kappa}$ where $\kappa =$ the sum of weights in the smallest by weight class. This leads to a 90% hull reduction in the smallest class, guaranteeing separability. Goodrich did not elaborate on why or how this guarantees separability and I have not been able to reach a conclusion.

7 Conclusion and Future

A conceptually simple, geometric interpretation of SVM training allows for more accessibility. We have seen how the geometric algorithm for convex hulls can be adapted for reduced convex hulls as well, with high accuracy.

For future work, I hope to transfer computations to Java for speed, expand to a multiclass SVM, and implement kernels to afford greater accuracy and adaptability.

References

[1] S. Theodoridis and M. Mavroforakis, "Reduced Convex Hulls: A Geometric Approach to Support Vector Machines [Lecture Notes]," *IEEE Signal Processing Magazine*, vol. 24, pp. 119–122, May 2007.

[2] V. Franc and V. Hlav, "An iterative algorithm learning the maximal margin classifier," *Pattern Recognition*, vol. 36, pp. 1985–1996, Sept. 2003.

[3] B. Goodrich, D. Albrecht, and P. Tischer, "Training weighted SVMs using a generalized Schlesinger-Kozinec algorithm," pp. 2435–2440, IEEE, Nov. 2011.