Sparse Support Vector Machines

Prasoon Goyal

December 15, 2014

Abstract

Support Vector Machines (SVMs) are state-of-the-art algorithms for classification in machine learning. However, the SVM formulation does not directly seek to find sparse solutions. In this work, we propose an alternate formulation that explicitly imposes sparsity. We show that the proposed technique is related to the standard SVM formulation and therefore shares similar theoretical guarantees. We further show that proposed formulation performs comparable to the standard SVM formulation on several synthetic datasets.

1 Introduction and Motivation

Support Vector Machines (SVMs) are one of the most widely used maximum margin classifiers in machine learning. Given a training set $S \in X \times \{-1, 1\}$ where $X \subseteq R^n$ with m samples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, the soft-margin SVM primal problem is the following:

$$\min_{\mathbf{w}, b, \boldsymbol{\xi}} \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{m} \xi_i$$
$$\text{s.t.}$$
$$\xi_i \geq 1 - y_i[\mathbf{w}^T\phi(\mathbf{x}_i) + b]$$
$$\xi_i \geq 0$$

Here, $\mathbf{w}$ is the weight vector, b is an offset parameter, ξ_i is the margin violation of the i^{th} training point and $\phi : R^n \to R^N$ is a (possibly non-linear) feature mapping that maps a point in R^n to a higher dimensional space R^N. The decision function is given by $f(\mathbf{x}) = sign\{\mathbf{w}^T\phi(\mathbf{x}_i) + b\}$. It can be shown that the confidence margin of the decision boundary is inversely proportional to the L_2-norm of the weight vector $\mathbf{w}$. Thus, the first term in the above problem seeks to maximize the margin, while the second term corresponds to empirical risk minimization. C is a hyperparameter that controls the trade-off between maximizing the margin and minimizing the training error.

The corresponding dual formulation for soft-margin SVMs is the following:

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i\alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$
$$\text{s.t.}$$
$$0 \leq \alpha_i \leq C$$
$$\sum_{i=1}^{m} \alpha_i y_i = 0$$

Here, α_i is the dual variable associated with the i^{th} primal constraint, and $K(\mathbf{x}, \mathbf{y})$ defined as $\phi(\mathbf{x})^T\phi(\mathbf{y})$ is known as the kernel function, which captures the similarity between points $\mathbf{x}$ and $\mathbf{y}$.

The primal variable $\mathbf{w}$ is related to the dual variable $\boldsymbol{\alpha}$ as

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \phi(\mathbf{x}_i)$$

Thus, the decision function can be represented as

$$f(\mathbf{x}) = sign\{\sum_{i=1}^{m} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x})\}$$

This can be interpreted as follows: each training point $\mathbf{x}_i$ in the training set is assigned an "importance weight" α_i. The label of a test point $\mathbf{x}$ is decided by a linear combination of labels of important points similar to $\mathbf{x}$. Note that this interpretation is consistent with the fact that non-support vectors are assigned $\alpha_i = 0$, and they are indeed unimportant for the decision, since their removal does not affect the decision boundary.

However, we see that neither the SVM primal nor the SVM dual prefers sparse solutions for $\mathbf{w}$ or $\boldsymbol{\alpha}$ respectively. And sparsity is desirable in many contexts because first, fewer non-zero learnt parameters will require smaller amount of memory, and more importantly, the prediction time for SVMs is proportional to the number of non-zero dual variables. These factors may be particularly important for machine learning algorithms implemented in devices with low memory and low computational power. In the next section, we propose an alternate formulation that is motivated from the above interpretation, and which directly attempts to impose sparsity restrictions on α.

2 Formulation

It is well-known that minimizing L_1-norm tends to impose sparsity. Therefore, we try to find vector $\boldsymbol{\alpha}$ with minimum L_1-norm that achieves low empirical error on the training set using the same decision function as that of SVMs. The precise formulation is as follows:

$$\min_{\boldsymbol{\alpha}, b, \boldsymbol{\xi}} \|\boldsymbol{\alpha}\|_1 + C\sum_{i=1}^{m} \xi_i$$
$$\text{s.t.}$$
$$\xi_i \geq 1 - y_i[\tfrac{1}{m}\sum_{j=1}^{m} \alpha_j y_j K(\mathbf{x}_i, \mathbf{x}_j) + b]$$
$$\xi_i \geq 0$$

Note that in the original SVM formulation, $\boldsymbol{\alpha}$ were Lagrange multipliers corresponding to inequality constraints, and were therefore constrained to be non-negative. However, here, we do not need to impose non-negativity constraints on $\boldsymbol{\alpha}$.
It is easy to see that the proposed formulation is convex, since the objective function is a linear combination of L_1-norm which is convex and linear functions which are again convex, and the constraints are affine, similar to the standard SVM formulation.

It is also worth noting that while in the standard SVM formulation, the kernel function must be positive definite symmetric to guarantee the existence of a feature mapping $\phi(\mathbf{x})$, here we do not need the kernel function to be positive definite symmetric, since there is no notion of feature mapping.

3 Theoretical Results

We first show that the proposed formulation is closely related to an instance of the standard SVM formulation. In particular, given m samples $(\mathbf{x}_1, y_1)$, ..., $(\mathbf{x}_m, y_m)$, and a kernel function $K : X \times X \to R$, we can write the optimization problem for the proposed formulation as

$$\min_{\boldsymbol{\alpha}, b, \boldsymbol{\xi}} \|\boldsymbol{\alpha}\|_1 + C\sum_{i=1}^{m} \xi_i$$
$$\text{s.t.}$$
$$\xi_i \geq 1 - y_i[\tfrac{1}{m}\sum_{j=1}^{m} \alpha_j y_j K(\mathbf{x}_i, \mathbf{x}_j) + b]$$
$$\xi_i \geq 0$$

Now, taking $\phi(\mathbf{x}) = \frac{1}{m}\begin{bmatrix} y_1 K(\mathbf{x}, \mathbf{x}_1) \\ \vdots \\ y_m K(\mathbf{x}, \mathbf{x}_m) \end{bmatrix}$ in the standard SVM formulation, we get

$$\min_{\mathbf{w}, b, \boldsymbol{\xi}} \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{m} \xi_i$$
$$\text{s.t.}$$

$$\xi_i \geq 1 - y_i[\tfrac{1}{m}\sum_{j=1}^{m} w_j y_j K(\mathbf{x}_i, \mathbf{x}_j) + b]$$
$$\xi_i \geq 0$$

Thus, we see that the two formulations are equivalent, modulo the fact that the standard SVM formulation minimizes the square of the L_2-norm, while the proposed formulation minimizes the L_1-norm of the weight vector.
Next, we show that extending the ideas of margin theory for the standard SVM formulation can be used to come up with theoretical guarantees for the proposed formulation. From [1], we know that for standard SVM formulation, the following holds:
Theorem: Let $H = \{\mathbf{x} \mapsto \mathbf{w}\cdot\mathbf{x} : \|\mathbf{w}\| \leq \Lambda\}$ and assume that $X \subseteq \{\mathbf{x} : \|\mathbf{x}\| \leq r\}$. Fix ρ >0, then, for any δ >0, with probability at least $1-\delta$, for any $h \in H$,

$$R(h) \leq \widehat{R}_\rho(h) + 2\sqrt{\frac{r^2\Lambda^2/\rho^2}{m}} + \sqrt{\frac{\log\frac{1}{\delta}}{2m}}$$

Thus, if H containts a hypothesis h that is able to achieve small empirical error, then as the sample size increases, the generalization error goes to zero. Note that in this case, the hypothesis set H remains unchanged as m changes.

The proposed formulation is equivalent to mapping the training data to an m-dimensional space using the feature-mapping ϕ defined above, and then finding a weight vector $\boldsymbol{\alpha}$ in the transformed space. So, if the training data lies in a ball of radius r, then given that $K(\mathbf{x},\mathbf{y})$ is bounded for all $\mathbf{x},\mathbf{y} \in X$, $\|\phi(\mathbf{x})\| \leq r'$ for all $\mathbf{x}$, where r' is a suitable constant.

Proceeding as in [1], setting $\rho = 1$, and upper-bounding the margin loss by hinge loss, we get that with probability at least $1-\delta$

$$R(h) \leq \tfrac{1}{m}\sum_{i=1}^{m}\xi_i + 2\sqrt{\frac{r'^2\Lambda^2}{m}} + \sqrt{\frac{\log\frac{1}{\delta}}{2m}}$$

where $\|\boldsymbol{\alpha}\| \leq \Lambda$. As in the case of SVM, the objective function minimized by the proposed formulation has precisely the form of the above bound: the first term in the above bound directly corresponds to the second term of the objective function, and the second term in the above bound corresponds to minimizing the L_1-norm of $\boldsymbol{\alpha}$.

However, there is an important difference between the standard SVM formulation and the proposed formulation: as m increases, the dimension of $\boldsymbol{\alpha}$ increases. Thus, for $m \gg \Lambda$, the second and the third terms on the right hand side of the bound go to zero, but at the same time, H is restricted to the set of hypotheses where almost all components of $\boldsymbol{\alpha}$ are close to zero, or only a finite number of components of $\boldsymbol{\alpha}$ are non-zero. Hence, if the problem at hand admits a sparse solution, then some hypothesis $h \in H$ will be able to achieve a small empirical error, and the model will be able to generalize. On the other hand, if the problem does not admit a sparse solution, then none of the hypotheses in H will have a small empirical error, and the bound will become uninformative. This agrees well with intuition: a sparse formulation is likely to work only when the true function has a sparse representation.

In order to ameliorate the above problem, it might look natural to allow Λ to be a function of m, to have a richer hypothesis set H as m increases. The two obvious choices for the functional dependence of Λ on m are $\Lambda \propto m$ (since L_1-norm is proportional to the dimension of the vector) and $\Lambda \propto \sqrt{m}$ (since L_2-norm is proportional to the square root of the dimension of the vector). However, it is easy to see that any of these will result in the second term not inversely related to m, and therefore the bound will no longer be informative.

4 Experiments and Results

The following steps were performed as part of preliminary experimentation.

4.1 MATLAB implementation : Linear program

The proposed formulation can be written as a linear program by writing the vector $\boldsymbol{\alpha}$ as the difference of two non-negative vectors: $\boldsymbol{\alpha} = \boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-$, with $\boldsymbol{\alpha}^+ \geq 0$ and $\boldsymbol{\alpha}^- \geq 0$.

The resulting linear program was implemented in MATLAB using `linprog`. However, since `linprog` requires the entire constraint matrix to be supplied, which is not sparse in our case, the implementation cannot be used to solve larger problems.

4.2 C++ implementation : Stochastic gradient descent

Since the proposed formulation is convex, we can solve the problem using stochastic gradient descent, which scales well to large datasets. The first set of experiments involved computing the gradient at every step with respect to only on one training point. However, the results were seen to have a large variance unless the number of iterations was increased significantly and the step size was carefully adapted. This suggested that the estimates of the gradients based on only one training point were perhaps not accurate. The algorithm was modified to approximate the gradient by taking the mean of the gradients with respect to P points at every step. Taking $P = \log_2 m$ resulted in much lower variances.

The above implementation used subgradients with the equivalent unconstrained minimization problem:

$$\min_{\boldsymbol{\alpha}, b, \boldsymbol{\xi}} \sum_{i=1}^{m} |\alpha_i| + C \sum_{i=1}^{m} \max\{0, 1 - y_i f(\mathbf{x}_i)\}$$

where

$$f(\mathbf{x}_i) = sign\{\sum_{j=1}^{m} \alpha_j y_j K(\mathbf{x}_j, \mathbf{x}_i)\}$$

One of the primary reasons for using the subgradients with unconstrained problem instead of solving the constrained problem directly was the following:
(1) Most off-the-shelf implementations to solve such problems require the entire constraint matrix to be supplied at once, which would have resulted in the same issue as the MATLAB implementation.
(2) A naive approach to handle the non-negativity constraints was to take the gradient step, and then if any of the variables that were constrained to be non-negative were found to be negative, setting them to zero. This was implemented at first, but it did not work well.
(3) Finally, the most common way to handle constraints in optimization literature is to use barrier functions. However, this did not look like a very promising method for the problem at hand, because we want most components of $\boldsymbol{\alpha}$ to be zero, while the barrier functions are undefined at the boundary, and may be ill-conditioned close to the boundary.

The above C++ code was found to be reasonably fast, and found solutions that performed almost as well as `LibSVM`. However, the solutions were not sparse. One potential reason for this was that the subgradients are not unique at non-differentiable points, where we want most of the components of $\boldsymbol{\alpha}$ to lie.

4.3 Merging the two implementations

To generate the final sets of results, the following procedure was adopted:
(1) Since the C++ implementation was reasonably fast and accurate, it was used to perform a coarse search for hyperparameters using cross-validation.
(2) The MATLAB implementation was used to fine-tune the hyperparameters and to generate a sparse solution.
(3) The resulting sparse solution was compared with the solution generated by `LibSVM` in terms of accuracy and sparsity.

The above steps are summarized schematically in the figure below.

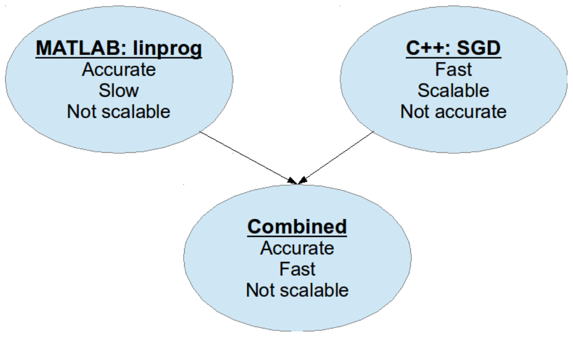

Figure 1: Schematic diagram of the implementation steps

In the preliminary experimentation, the datasets were restricted to have about 1000 training instances, since the MATLAB implementation above cannot handle much larger sized problems. Furthermore, the datasets were restricted to synthetic data in 2 dimensions, since the sparsity of such datasets is easy to establish. Gaussian kernel was used for `LibSVM` and for the proposed formulation.

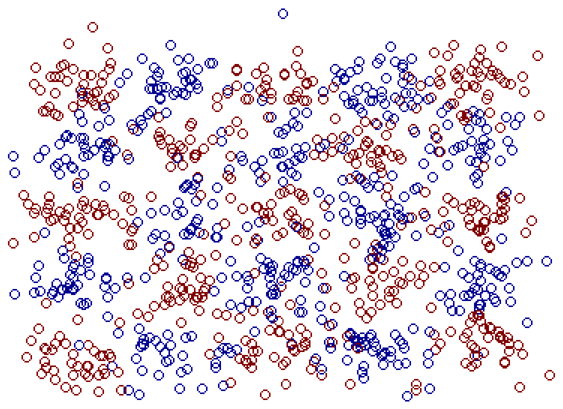

Figure 2: Dataset 1

The first dataset consisted of 25 clusters, each with 40 points, as shown in Figure 2. A sparse solution to this problem would have non-zero α_i corresponding to the points at the cluster centers, while all other points will have zero α_i. Thus, the problem admits a sparse solution. It is seen that the proposed formulation performs as well as the standard SVM formulation on this dataset, and produces a solution that is much sparser.

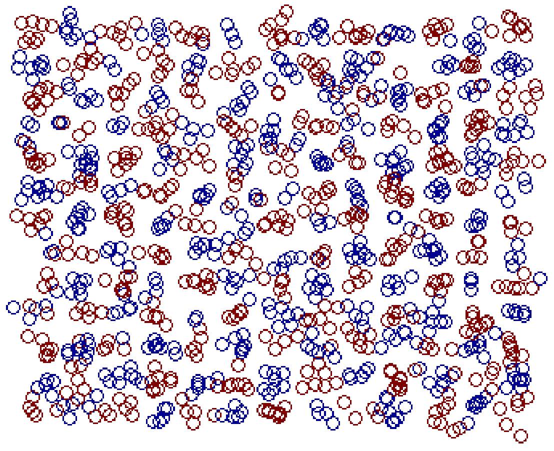

Figure 3: Dataset 2

The second dataset consisted of 169 clusters, each with 6 points, as shown in Figure 3. As above, a solution to this problem would have 169 non-zero α_i, which is not quite sparse. It is observed that the proposed formulation outperforms the standard SVM formulation on this dataset. However, while the solution produced has fewer non-zero α_i than that produced by `LibSVM`, it is far from being sparse. Enforcing sparsity by reducing C results in loss of accuracy, which agrees with the theoretical results.

The following table summarizes the datasets used:

Dataset	**N**	**m_train**	**m_test**
D1	2	1000	1000
D2	2	1014	1014

The results of the two formulations are summarized below:

	LibSVM		**Sparse_SVM**	
Dataset	**Error %**	**nSV**	**Error %**	**nSV**
D1	16.2000	394	16.1000	75
D2	13.4122	799	10.7495	366

5 Conclusion and Future Work

We proposed a new sparse formulation for support vector machines. We showed that the proposed formulation is closely related to the standard SVM formulation. Extending the margin theory of standard SVMs, we developed theory for the proposed formulation which shows that it can generalize well for problems that admit a sparse solution.

Preliminary experiments on small synthetic datasets show that the formulation does as well as `LibSVM` in terms of accuracy, while producing much sparser solutions when a sparse solution exists.

Future work aims at running experiments on larger benchmark datasets, and with kernels other than RBF kernel. Also, we can look at extending theoretical results for emsemble methods for our formulation.

6 References

[1] Mohri, Mehryar, Afshin Rostamizadeh, and Ameet Talwalkar. "Foundations of machine learning." MIT press, 2012.

[2] Cortes, Corinna, and Vladimir Vapnik. "Support-vector networks." Machine learning 20.3 (1995): 273-297.

[3] Chang, Chih-Chung, and Chih-Jen Lin. "LIBSVM: a library for support vector machines." ACM Transactions on Intelligent Systems and Technology (TIST) 2.3 (2011): 27.