MULTICLASS SUPPORT VECTOR MACHINES WITH GENSVM

P.J.F. Groenen, G.J.J. van den Burg
Econometric Institute, Erasmus University Rotterdam
Rotterdam, THE NETHERLANDS
e-mail: groenen@ese.eur.nl

Abstract

Binary support vector machines (SVM) have become a standard tool for supervised machine learning. Attractive features of the SVM are that its solution only depends on badly fitting observations, it combats overfitting through regularization, can handle high dimensionality (thus many predictors), allows for nonlinear predictions, and is robust against outliers. Much less attention has been given to classification problems with more than two classes. In the machine learning literature, such multiclass problems tend to be solved by repeatedly applying binary SVMs, for example, through one-versus-one (OvO) or one-versus-all (OvA). Although such approaches are generally fast, they can lead to regions that are inconclusive in their prediction. As an alternative, the present authors have proposed a single machine classifier, called GenSVM (see, [1]). In this paper, we present its main properties and discuss examples of its implementations in the R and Python packages.

Keywords: data science, support vector machine, multiclass classification

1 Introduction

Multiclass classification is abundant in many areas of empirical science. In medicine, the task could be to predict cancer stages out of blood values, in epidemiology to predict college degree out of genetic variables, and in marketing to predict buyers from nonbuyers out of previous shopping behavior. All these cases are examples of the multiclass classification problem that has K (two or more) response categories, with usually one or more predictor variables. Often, the goal is to find a linear combination of the predictor variables that separates the predictor space into K polyhedral sets. Then the decision rule is to assign the observation to the class of the polyhedral set it falls in. Some multiclass classification techniques include: Fisher discriminant analysis, multinomial regression, classification trees, and neural networks.

Here, we focus on the supervised learning technique of support vector machines (SVMs). The binary SVM is quite popular in computer science. The advantage of the binary SVM is that it only depends on observations that are not perfectly predicted, it is robust against outliers, and has a provision against overfitting by using regularisation term. Multiclass classification is often performed through repeated binary SVMs, either through one-versus-one (OvO, doing binary SVMs between all pairs of classes, or through one-versus-all (OvA, for each class doing a binary SVMs between itself against the remaining classes). OvO has the advantage that each binary SVM solves a small subproblem, whereas OvA only needs to solve K problems. Both approaches can make

use of standard binary SVM implementations. However, there are also disadvantages of OvO and OvA. For example, OvO needs to solve $K(K-1)/2$ problems which can become large as K grows. OvA can be expected to be slower than OvO because for each class the binary SVM problem is as large as the full data set. More importantly, the binary SVM is not designed for the multiclass problem. Both OvO and OvA can have ambiguity in prediction regions.

To overcome these problems, Van den Burg and Groenen (2016) [2] proposed a novel single machine multiclass SVM called GenSVM. It is specifically designed for the multiclass problem, it is a direct extension of the binary SVM, it is flexible through different weightings and hyperparameters, is fast and accurate. In this paper, we explain the main properties of GenSVM and show an example from the R package (see [3]) and Python package of GenSVM.

2 GenSVM

The basic idea of GenSVM is as follows. Let the $n \times m$ matrix $\mathbf{X}$ contain m predictor variables from the training set. In addition, let $\mathbf{y}$ be an n vector with class labels $\{1, 2, \ldots, K\}$ for each of the i objects. Also, let the $K \times (K-1)$ matrix $\mathbf{U}$ contain coordinates of a regular simplex in $K-1$ dimensions. For example, for $K = 3$ classes, the simplex is an equilateral triangle in two dimensions. Then, the goal of GenSVM is to (non)linearly map the n objects in the p dimensional space given by $\mathbf{X}$ to the $K-1$ dimensional space of the simplex such that each object i is in or as close as possible to the prediction region of its class y_i according to some missclassification error. Figure 1 gives an example of a multiclass classification problem with $K = 3$ classes and $m = 2$ predictor variables. The left panel shows the n objects as points labeled by their class in the space of the predictor variables in $\mathbf{X}$. The middle panel show the $K - 1 = 2$ dimensional simplex space with the boundaries (solid lines) of the predictor regions. The right panel shows the nonlinear boundaries of the prediction regions in the original space of $\mathbf{X}$.

Important aspects of GenSVM are (a) how exactly the mapping is being done from the original space of $\mathbf{X}$ to the simplex space $\mathbf{S}$ and (b) how the errors are being defined. We will discuss linear mappings only. For an explanation on how nonlinear mappings can be obtained through kernels, we refer to the appendix of [2]. The linear mapping of $\mathbf{X}$ to the simplex space $\mathbf{S}$ is obtained by

$$\mathbf{S} = \mathbf{X}\mathbf{W} + \mathbf{1}\mathbf{c}^\top$$

with $\mathbf{W}$ the $m \times (K-1)$ matrix with unknown weights and $\mathbf{c}$ the $(K-1) \times 1$ translation vector.

Once the mapping is obtained, GenSVM needs to determine how good or bad an object is placed. Consider Figure 4 where object A is shown as a point in the simplex space corresponding to one of the rows of $\mathbf{S}$. Assume that A has $y_A = 2$ so that ideally A should be located in the shaded area around vertex $\mathbf{u}_2$.

To measure the error in classification, GenSVM uses a measure of distance to the shaded area. In particular, consider the projection q_A^{21} of point A onto difference vector

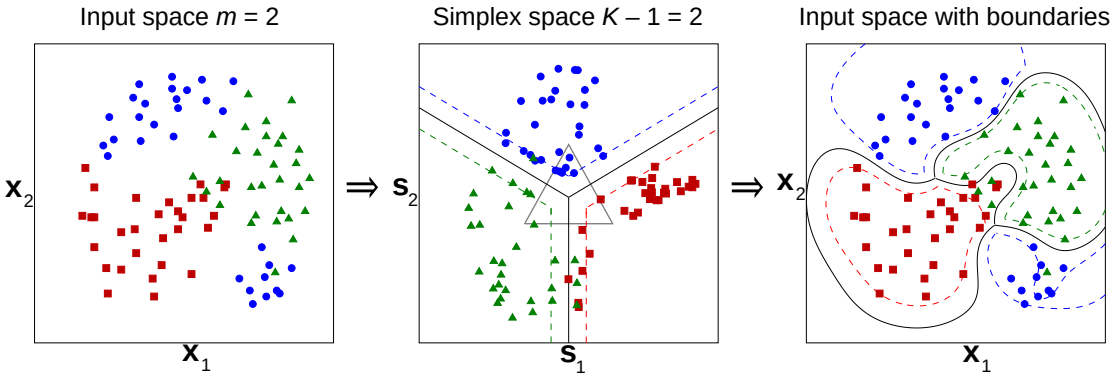

Figure 1: Example of objects in $K = 3$ classes in the original $p = 2$ dimensional space of $\mathbf{X}$ (left panel), their mapping in the $K - 1 = 2$ dimensional simplex space, the simplex, and the prediction regions (middle panel), and the resulting nonlinear prediction regions for the classes in the original space of $\mathbf{X}$ (right panel). This figure is reproduced from [2].

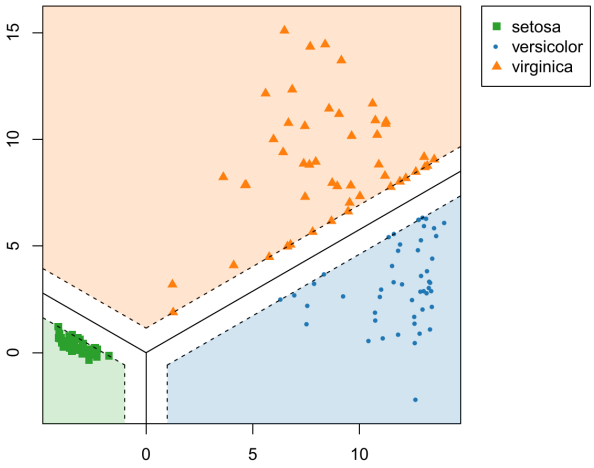

Figure 2: The simplex space with misclassified object A that ideally should have been positioned in the shaded prediction region of its class 2. The projections q_A^{21} and q_A^{23} show how far A is from the boundaries separating vertices $\mathbf{u}_2$ and $\mathbf{u}_1$ and vertices $\mathbf{u}_2$ and $\mathbf{u}_3$. This figure is reproduced from [2].

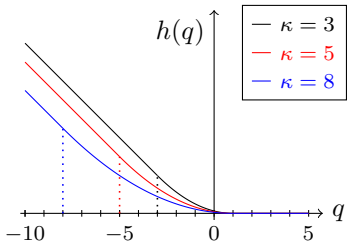

Figure 3: The huberized hinge error $h(q)$.

$\mathbf{u}_2 - \mathbf{u}_1$ that measures the distance of A to the boundary separating class 2 from 1. Here, this boundary coincides with the vertical axis, thus all points with $s_1 = 0$. Let the values on the predictor variables for A be in the vector $\mathbf{x}_A^\top$. Then, the position of A in the simplex space is given by $\mathbf{s}_A^\top = \mathbf{x}_A^\top \mathbf{W} + \mathbf{c}^\top$ and its projections onto the difference vectors separating class 2 from 1 and 3 by

$$\begin{aligned} q_A^{(21)} &= \mathbf{s}_A^\top(\mathbf{u}_2 - \mathbf{u}_1) \qquad (< 0 \text{ when misclassified}) \\ q_A^{(23)} &= \mathbf{s}_A^\top(\mathbf{u}_2 - \mathbf{u}_3) \qquad (< 0 \text{ when misclassified}). \end{aligned}$$

The seriousness of the misclassification is given by the Huberized hinge error

$$h(q) = \begin{cases} 1 - q - (\kappa + 1)/2 & \text{if } q \leq -\kappa \\ (1 - q)^2/(2(\kappa + 1)) & \text{if } q \in (-\kappa, 1] \\ 0 & \text{if } q > 1 \end{cases}$$

as shown in Figure 3. Finally, a rule is needed to combine the misclassification errors. For this, GenSVM uses the L_p norm

$$h\left(q_i^{(y_i,j)}\right) = \left(\sum_{j \neq y_i} h^p\left(q_i^{(y_i,j)}\right)\right)^{1/p}$$

where $1 \leq p \leq 2$. The effect for this error function is shown in Figure 4 for $p = 2$ and $\kappa = -0.95$. In this case, one can easily see that this error is a function of the Euclidean distance of the point to the boundary. As κ grows larger, the sharp bend becomes more smooth (quadratic). If $p = 1$, the L_1 norm is used, implying that misclassification with respect to multiple classes receives a higher error than misclassification with respect to a single class only.

With these definitions, the GenSVM loss function can be defined as

$$L_{\text{MSVM}}(\mathbf{W}, \mathbf{c}) = \frac{1}{n}\sum_{i=1}^{n} \rho_i \left(\sum_{j \neq y_i} h^p\left(q_i^{(y_i,j)}\right)\right)^{1/p} + \lambda \text{tr } \mathbf{W}'\mathbf{W} \tag{1}$$

with $\rho_i \geq 0$ prespecified object weights and $\lambda > 0$ a given penalty strength parameter. Note that the term tr $\mathbf{W}'\mathbf{W}$ is quadratic in $\mathbf{W}$ and penalizes nonzero w_{kl}. The GenSVM

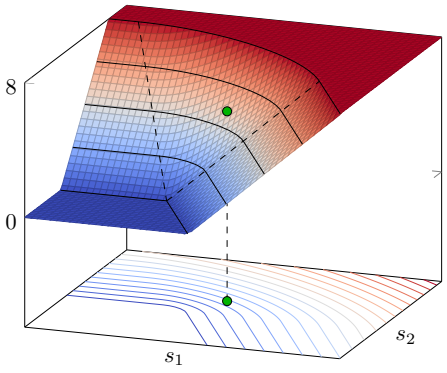

Figure 4: The combined hinge error by $h\left(q_i^{(y_i,j)}\right)$ for $p = 2$ and $\kappa = -0.95$.

loss function has several nice properties. First, it is a convex function in $\mathbf{W}$ and $\mathbf{c}$ as it is a sum of convex functions. Consequently, its minimum is global. Secondly, (1) simplifies into standard binary SVM if $K = 2$ and $\kappa = -1$. Finding optimal values for λ, p, and κ can be done through K-fold cross validation.

The GenSVM algorithm developed in [2] and implemented in [3] that optimizes $L_{\text{MSVM}}(\mathbf{W}, \mathbf{c})$ is based on the MM (minimization by majorization principle, see [1]). One of the main advantages of MM is that for a given λ, p, and κ the function value decreases until a minimum is reached. In K-fold cross validation, (1) needs to minimized for many combinations of λ, p, and κ. The advantage of the MM algorithm is that warm starts can be used, for example, if two subsequent λ values are only slightly different, then the $\mathbf{W}$ and $\mathbf{c}$ obtained can be used as start values for the run of the next λ. Often only a very few iterations are needed thereby greatly reducing the computational efforts.

3 The GenSVM package in R

The GenSVM package in R implements the loss function and minimization procedure discussed above.[1] The core of the package is written in `C` so that is tends to run fast. The linear algebra routines used by GenSVM use the optimized BLAS and LAPACK libraries shipped with R, and therefore automatically takes advantage of multi-core architectures.

Below is a simple example of a nonlinear GenSVM applied to the Fisher Iris data using the radial basis function (RBF) kernel. Because the Iris data only has $K = 3$ classes, the simplex space can be visualized in 2D in Figure 5.

```
R> library(gensvm)
R> x <- iris[, -5]
R> y <- iris[, 5]
R> #  Fit a nonlinear GenSVM through the RBF kernel
R> fit <- gensvm(scale(x), y, kernel='rbf', max.iter=10000)
```

[1] The `gensvm` package is available on CRAN.

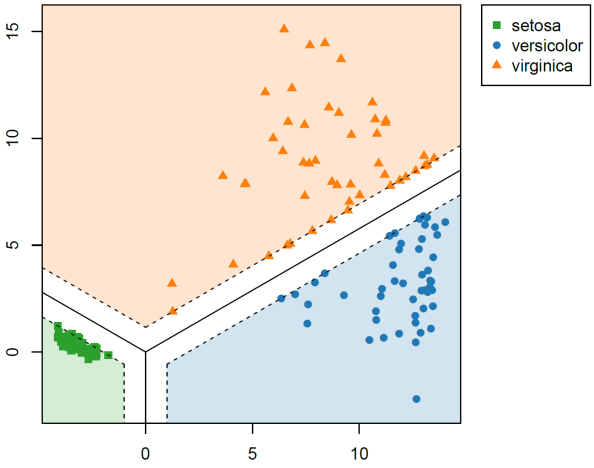

Figure 5: Simplex space of the GenSVM solution on the iris data set.

```
R> plot(fit)
```

The R package includes code for efficiently running cross validation and grid search over the hyperparameters. This code ensures that warm starts are used throughout, thereby increasing the efficiency of the MM algorithm. For GenSVM and SVMs in general, optimizing the hyperparameters is an important step for obtaining high classification accuracy. GenSVM is flexible through three different hyperparameters (κ, p, and λ) and optimizing these through grid search can be a time-consuming task for the user. To facilitate this, the GenSVM R package contains three pre-defined parameter grids in the function `gensvm.grid` that are designed to obtain high out-of-the-box classification accuracy. These parameter grids were constructed from the best hyperparameter configurations found in the large experimental study of [2]. For instance, the following code illustrates running a grid search with a very small hyperparameter grid:

```
R> fit <- gensvm.grid(x, y, param.grid = 'tiny')
```

A convenient feature of this grid search function is that the resulting object can directly be used with common R functions such as `plot` and `predict`, for which the best performing model will be used. Moreover, the object contains a `cv.results` attribute that holds a data frame with the complete results of the grid search.

Another common task in fitting SVMs is scaling the data and creating a training and test dataset. Scaling the features is necessary to ensure that the regularization term works equally on all features. To facilitate this, the GenSVM R package contains the `gensvm.maxabs.scale` function that scales the features to the interval $[-1, 1]$ while preserving sparsity. Furthermore, the R package also includes a function for creating a train and test dataset called `gensvm.train.test.split`. These functions can be combined as follows:

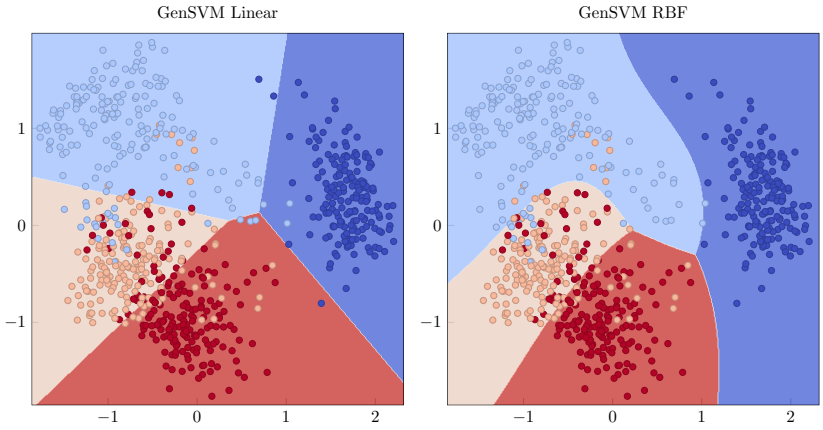

Figure 6: Illustration of linear and nonlinear GenSVM for a subset of the handwritten digits dataset using the Python package for GenSVM. Dimensionality reduction was applied to obtain two input dimensions for easy visualization, but this is not a requirement for GenSVM.

```
R> split <- gensvm.train.test.split(x, y, random.state=123)
R> x.train <- split$x.train; y.train <- split$y.train;
R> x.test <- split$x.test; y.test <- split$y.test;
R> scaled <- gensvm.maxabs.scale(x.train, x.test)
R> fit <- gensvm(scaled$x, y.train, kernel='rbf', max.iter=1000, random.seed=123)
R> gensvm.accuracy(predict(fit, scaled$x.test), y.test)

[1] 0.921
```

4 The GenSVM package in Python

The Python package for GenSVM has similar functionality to the R package, but is based on the object-oriented framework for machine learning methods used in Scikit-Learn [4].[2] This allows for straightforward interoperability with existing code that uses methods from the Scikit-Learn package. The computational routines in the Python package are again implemented in `C` (in fact, both packages share the same `C` library) and are linked to Python through Cython. An example of fitting and predicting a GenSVM model with the Python package is as follows:

```
from gensvm import GenSVM

clf = GenSVM(kernel='rbf', verbose=1)
clf.fit(x, y)
```

Note that due to the object-oriented nature of the code, parameters that determine the kernel and affect the loss function are provided in the constructor of the `GenSVM` object.

[2]The Python package for GenSVM is available on PyPI.

While previously we have illustrated the *simplex space* of a GenSVM solution, we now illustrate the decision boundaries in the *input space*. We use a subset of the handwritten digits dataset from the UCI repository [5] and illustrate the decision boundaries in the original space in Figure 6. Notice that the dataset is not separable with either a linear or nonlinear kernel. However, GenSVM with the RBF kernel achieves better separation of the classes.

References

[1] Hunter, D.R. & Lange, K. (2004). A tutorial on MM algorithms. *The American Statistician*, *58* (1):30-–37.

[2] Van den Burg, G.J.J. & Groenen, P.J.F. (2016). GenSVM: A generalized multiclass support vector machine. *Journal of Machine Learning Research*, *17*, 1-42.

[3] Van den Burg, G.J.J. & Groenen, P.J.F. (2018). *GenSVM: A generalized multiclass support vector machine.* `https://cran.r-project.org/package=gensvm`, *R-package*.

[4] Pedregosa, F. et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12* (Oct):2825–2830.

[5] Bache, K. & Lichman, M. (2013). UCI Machine Learning Repository. *University of California, Irvine*. `http://archive.ics.uci.edu/ml`