The Stata Journal (2016)
16, Number 4, pp. 917–937

Support vector machines

Nick Guenther
University of Waterloo
Waterloo, Canada
nguenthe@uwaterloo.ca

Matthias Schonlau
University of Waterloo
Waterloo, Canada
schonlau@uwaterloo.ca

Abstract. Support vector machines are statistical- and machine-learning techniques with the primary goal of prediction. They can be applied to continuous, binary, and categorical outcomes analogous to Gaussian, logistic, and multinomial regression. We introduce a new command for this purpose, `svmachines`. This package is a thin wrapper for the widely deployed `libsvm` (Chang and Lin, 2011, *ACM Transactions on Intelligent Systems and Technology* 2(3): Article 27). We illustrate `svmachines` with two examples.

Keywords: st0461, svmachines, svm, statistical learning, machine learning, support vector machines

1 Introduction

The primary purpose of Gaussian linear regression is explanation. The linear model makes it possible to explain how an individual x variable affects the outcome y because changing x_i by one unit leads to an average change in y by β_i units. In contrast, the primary purpose of statistical or machine learning is prediction. Without the constraint of providing an easy explanation, models are free to be more complicated, capturing extra detail in the relationship between x and y that simpler models may miss. Sometimes, such models are called "black box" models.

To our knowledge, Stata has little representation in the machine-learning community. In this article, we introduce a new command, `svmachines`, that offers access to a classic machine-learning algorithm—the support vector machine (SVM)—in hopes of opening up this field to Stata users. The software is available for Windows (64-bit), Mac OS X (64-bit), and Linux (64-bit and 32-bit).

The article is outlined as follows: Section 2 outlines the syntax of the new command. Section 3 introduces SVMs for classification and for regression. It also includes a discussion about how probabilities are computed and how two-class classification is extended to multiclass classification. Section 4 gives an example of SVM classification, and section 5 gives an example of SVM regression. Section 6 concludes with a discussion.

2 Syntax

The full syntax of the command to fit a SVM model is as follows:

```
svmachines depvar indepvars [if] [in] [, type(type) kernel(kernel) c(#)
   epsilon(#) nu(#) gamma(#) coef0(#) degree(#) shrinking probability
   sv(newvar) tolerance(#) verbose cache_size(#)]
```

The most interesting thing a fitted machine-learning model can do is predict response values. To that end, the standard `predict` command may be used during postestimation as follows:

```
predict newvar [if] [in] [, probability scores verbose]
```

We explain the options in the SVM section below.

3 Support vector machines (SVMs)

SVMs (Cortes and Vapnik 1995; Vapnik 2000) can be used for regression of three different types of outcomes: Bernoulli (binary), multinomial, or continuous. Regression with Bernoulli outcomes is often referred to as classification in statistical learning. Regression with multinomial outcomes is called multiclass classification. Finally, regression with continuous outcomes is simply called regression.

3.1 Support vector classification

Analogous to logistic regression, SVMs were initially conceived for classification with two classes. This approach was later extended to address continuous outcomes and classification with more than two classes. In `svmachines`, this mode is available through the `type(svc)` option. We now introduce the mathematics of SVMs with two-class classification.

Figure 1 gives an example with two x variables. The outcome, class membership ($y = 1$ or $y = -1$), is indicated by the plotting symbol. We can draw many lines that fully separate the two classes, two of which are shown in figure 1. Which of them is "best"? One sensible choice is to choose a separating line such that the margin—the minimum distance between the line and the closest point—is maximized. This leads to the following optimization problem,

$$\underset{\beta_j, j=1,\ldots,p}{\text{maximize}}\, M \tag{1}$$

$$\text{subject to} \begin{cases} \sum_{j=1}^{p} \beta_j^2 = 1 \\ y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \geq M \end{cases} \tag{2}$$

where M is the margin, p is the number of x variables, and $y_i \in \{-1, 1\}, i = 1, \ldots, n$. The second line in (2) represents a hyperplane. In two dimensions, as in figure 1, the hyperplane is a line. Observations above the hyperplane satisfy

$$(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) > 0$$

and observations below the hyperplane satisfy

$$(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) < 0$$

Observations below the hyperplane have a class label of -1, so

$$y(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip})$$

is always positive as long as the two classes are separable. This mathematical convenience is the reason that the class labels are $\{-1, 1\}$ rather than the typical labels, $\{0, 1\}$, used in logistic regression. The constraint on the β_j in (2) is not necessary but has the advantage that the distance of any observation i to the hyperplane equals

$$y(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip})$$

when the β_j are scaled such that the constraint holds.

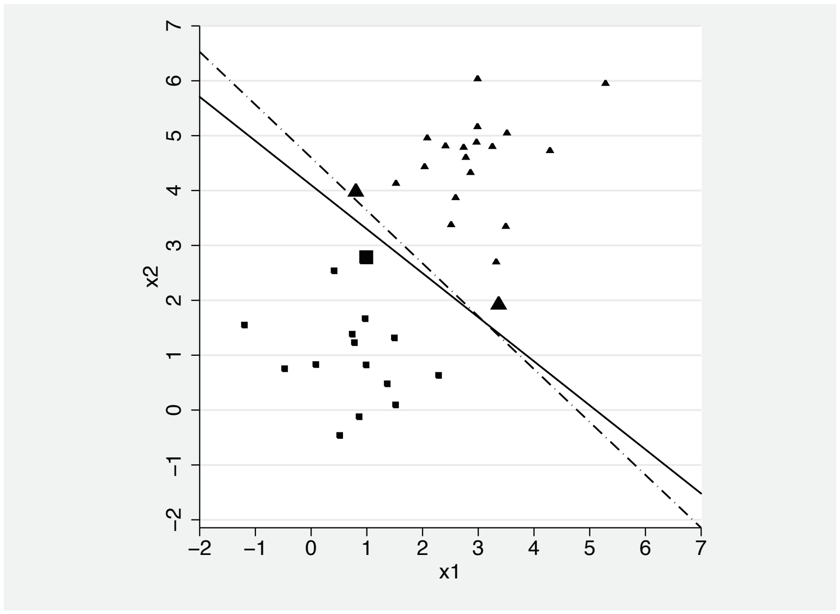

Figure 1. Two separable classes of observations with the maximum margin separating line (solid) and a second separating line (dashed). Triangles and squares denote the two classes (y values). Larger triangles and squares denote support vectors.

The optimization problem in (1)–(2) can be solved with a quadratic program solver. Importantly, the solution depends only on observations that lie on the margin. This greatly reduces computation because only a fraction of the data is necessary to compute an optimal solution. That subset of observations is called the support vectors. The example in figure 1 has three support vectors.

In most applications, the two classes are not separable, and the optimization problem has no solution. This can be addressed by allowing a certain amount of error:

$$\underset{\beta_{ij},\epsilon_i,i=1,\ldots,n;j=1,\ldots,p}{\text{maximize}} \quad M \tag{3}$$

$$\text{subject to} \begin{cases} \sum_{j=1}^{p} \beta_j^2 = 1 \\ y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \geq M(1-\epsilon_i) \\ \epsilon_i \geq 0; \ \sum_{i=1}^{n} \epsilon_i \leq C \end{cases} \tag{4}$$

The introduction of so-called slack variables, ϵ_i, allows for observations to lie on the wrong side of the margin. However, the sum of such errors cannot exceed a total error budget, C. C is specified by passing `c(#)` to the `svmachines` command.

Again, the solution to the optimization problem depends only on the support vectors: observations that lie on the margin or observations that violate the margin. Consider the example in figure 2. The solid line separates most of the triangles and squares, except for one square above the line and one triangle below the line (both are circled). The upper dashed line is the margin for the triangles, and the lower dashed line is the margin for the squares. As before, there are some observations directly on the margin. Unlike before, there are also observations that violate the margins. Observations that violate the margin are not misclassified as long as they lie on the correct side of the decision boundary. Note one of the misclassified observations, a square, appears outside the two margins. It is nonetheless a support vector because it violates the lower margin. The violation is so large that it falls beyond the upper margin.

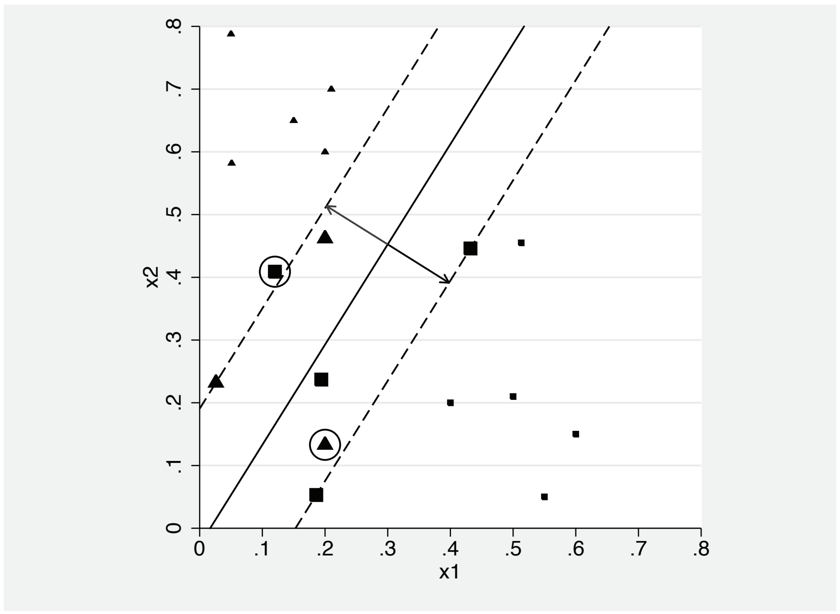

Figure 2. Two nonseparable classes of observations. The solid line represents the decision boundary; the dashed lines represent the upper and lower margins. Triangles and squares denote the two classes (y values). Larger triangles and squares denote support vectors. Misclassified observations on the wrong side of the decision boundary are circled.

Large C values increase the penalty for misclassifications on the training data. This corresponds to tighter margins. Large C values also correspond to fewer support vectors because fewer observations are misclassified and because fewer observations lie within the tighter margins. Figure 2 demonstrates an SVM with a large C value (10,000). The number of misclassified observations (in the training data) is minimized. Minimizing the number of misclassifications in the training may result in overfitting to the training data with poor performance on the test data. We call C a tuning parameter and will discuss choosing it in section 3.6.

Mathematically, the classification for the two classes of y is based on sign$\{\widehat{f}(x)\}$ (that is, which side of the hyperplane the observation falls on) with

$$\begin{aligned}\widehat{f}(x) &= \widehat{\beta}_0 + \widehat{\beta}_1 x_{i1} + \cdots + \widehat{\beta}_p x_{ip} \\ &= \widehat{\beta}_0 + \sum_{i \in S} \widehat{\alpha}_i y_i \langle x, x_i \rangle \qquad (5)\end{aligned}$$

where $\widehat{\alpha}_i > 0$ are estimated coefficients arising during optimization (not further explained here), S is the set of support vectors, and $\langle ., . \rangle$ is the inner (dot) product. The dot product is a measure of similarity between the two observations $i \in \{1, \ldots, n\}$ and $i' \in \{1, \ldots, n\}$. We will call this inner product K:

$$K(x_i, x_{i'}) = \langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j} \tag{6}$$

This is also called a linear kernel.

We sometimes want to add additional variables as a function of original variables, such as polynomial variables. Polynomial variables increase the number of variables available for regression, but the number of underlying x variables remains unchanged. Such variables can also be added in SVMs. Adding variables does not change the optimization problem in (3)–(4).

Let us define $h(x_i) = \{h_1(x_i), h_2(x_i), \ldots, h_{p'}(x_i)\}$ for $i = 1, \ldots, n$. For example, a quadratic term for x_1 of observation i might be added by specifying $h(x_i) = (x_{i1}, x_{i2}, \ldots, x_{ip}, x_{i1}^2)$. Equation (5) turns into

$$\begin{aligned}\widehat{f}(x) &= \widehat{\beta}_0 + \widehat{\beta}_1 h(x_{i1}) + \cdots + \widehat{\beta}_p h(x_{ip}) \\ &= \beta_0 + \sum_{i \in S} \widehat{\alpha}_i y_i \langle h(x), h(x_i) \rangle \end{aligned} \tag{7}$$

The solution to the optimization problem in (7) depends only on the inner (dot) product of the observations (not on the observations themselves).

The SVM method using the linear kernel in (6) can be specified with `kernel(linear)`, but nonlinear kernels is more flexible. Nonlinear kernels map the p and x variables into a higher dimensional space. The mapping occurs implicitly; polynomial and other variables in the higher dimensional space are not actually computed. The same number of observations is more sparse in a higher dimensional space, which makes them more likely to be linearly separable. This improves the performance of the SVM algorithm and, as a by-product, makes the decision boundaries back in the original space bendable (Hastie, Tibshirani, and Friedman 2009, chap. 12).

The most popular choice for a nonlinear kernel is the radial (or Gaussian) kernel, specified as `kernel(rbf)`,

$$K(x_i, x_{i'}) = \exp\left\{ -\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 \right\}$$

where $\gamma > 0$ is an additional tuning parameter specified by `gamma(#)`. When a test observation is very far from a training observation, the exponent becomes strongly negative, and $K(x_i, x_{i'})$ is close to zero. In this case, training observations far away from a test observation have almost no effect on the decision of how to classify the test observation. The parameter γ regulates how close observations have to be to contribute to the classification decision. This kernel is more flexible than the linear kernel and is the default choice.

The polynomial kernel of degree d, specified as `kernel(poly) degree(d)`, is

$$K(x_i, x_{i'}) = \left(\beta_0 + \gamma \sum_{j=1}^{p} x_{ij} x_{i'j}\right)^d$$

where $\gamma > 0$ and β_0 are additional tuning parameters. The parameter γ is specified as before, and β_0 is specified by `coef0(#)`. β_0 "biases" the similarity metric for all samples; often it can be left at its default, `coef0(0)`. Using this kernel is equivalent to adding polynomial powers of the x variables, as in polynomial regression. However, here the additional variables are not explicitly computed, only implied in the use of kernels. The default is `degree(3)`. Larger values for d make the model more complex, but often $d = 2$ or $d = 3$ is sufficient.

Another kernel choice is the sigmoid kernel, specified as `kernel(sigmoid)`,

$$K(x_i, x_{i'}) = \tanh\left(\beta_0 + \gamma \sum_{j=1}^{p} x_{ij} x_{i'j}\right)$$

where $\gamma > 0$ and β_0 are additional tuning parameters. This kernel is not as commonly used as the previously mentioned kernels, in part because of some undesirable theoretical properties (not positive definite for some parameter settings) (Lin and Lin 2003). We mention it here for completeness and because it is used in the context of neural networks, another important learning algorithm.

3.2 Multiclass classification

There is no SVM equivalent to multinomial regression for more than two classes. Instead, results from individual two-class SVMs are combined. There are multiple ways of doing so. The Stata implementation applies the "one-against-one" approach: If k is the number of classes, the support vector classification algorithm is run $k(k-1)/2$ times for each possible pair of classes. For each pair, the chosen class receives a point, and the class that collects the most points of all two-class SVMs is the winner. In the unlikely event of a tie, a deterministic arbitrary tie-breaker rule is used. The one-against-one approach performed well in a comparison of approaches (Hsu and Lin 2002), and the training time for one against one is shorter than that of one against all (Lin 2015).

There is no change in syntax to run a multiclass classification; `svmachines` will detect a multiclass situation automatically and invoke one against one.

3.3 Probability estimates

Two-class SVMs do not natively compute probabilities the way logistic regressions do. Instead, they output a score, with positive values meaning $y = 1$ is more likely, negative values meaning $y = -1$ is more likely, and $y = 0$ meaning both classes are equally likely. (This score is not usually of interest but is available by specifying the `scores` option of `predict`.) This score is mapped to a pseudoprobability using (regularized) logistic regression, with class membership (0/1) as the dependent variable and score as the only independent variable (Platt 2000; Lin, Lin, and Weng 2007).

In multiclass classification, the one-against-one approach compares a given class with $(k-1)$ other categories using $(k-1)$ two-class support vector classifiers. First, the pairwise class probabilities $P(y = i|y = i \textit{ or } j, x)$ are computed as before in two-class classification. Second, the pairwise class probabilities are used to calibrate the probabilities $p(y = i|x), i = 1, \ldots, k$. Details are described in Chang and Lin (2011) and in the second method of Wu, Lin, and Weng (2004).

`svmachines` will generate these probabilities if the computationally expensive option `probability` is specified. `predict` also requires the `probability` option.

`predict` uses the *newvar* passed to it as a stem and generates one variable for each class, *newvar_classname*, in addition to the normal column of predicted classes, *newvar*. For example, if the two y values are labeled 0 and 1, the command `predict p, probability` will generate the variables `p`, `p_0`, and `p_1`. All the *newvar_classname* sum to 1 because the sample must fall into one of the classes.

A model fit with `probability` can still `predict` without it, and this behaves as if `probability` was never specified. However, beware that the two will give slightly different predictions. Predicting with probabilities internally runs an entirely different algorithm: the probabilities are computed, and the class with the greatest probability is chosen, rather than the point method that declares the class with the most points the winner. One implication is that two-class predictions scores of 0 (equal chance) will not map to a probability of 0.5 but may be a little larger or a little smaller.

3.4 Support vector regression

Support vector classification has been extended to work for continuous outcomes and is called support vector regression, which is activated with `type(svr)`. Smola and Schölkopf (2004) give a good tutorial. We summarize the method below.

By way of comparison, in standard Gaussian regression, the squared error loss function is minimized—with the loss for observation i given as $L\{y_i, f(x_i)\} = \{y_i - f(x_i)\}^2$. Instead, in support vector regression, the so-called ϵ-insensitive loss function is minimized: any loss smaller than ϵ is set to zero, and beyond that bound, a simple linear loss function is used:

$$L_\epsilon = \begin{cases} 0 & \text{if } |y_i - f(x_i)| < \epsilon \\ |y_i - f(x_i)| - \epsilon & \text{otherwise} \end{cases}$$

For example, if $f(x)$ is the linear function $f(x) = \beta_0 + x_i^t\beta$, the loss function is

$$\sum_{i=1}^{n} \max(y_i - x_i^t\beta - \beta_0 - \epsilon, 0)$$

Note ϵ is a tuning parameter. This can be rewritten as a constrained optimization problem:

$$\begin{aligned} &\text{minimize } \frac{1}{2}||\beta||^2 \\ &\text{subject to } \begin{cases} y_i - x_i^t\beta - \beta_0 \leq \epsilon \\ -(y_i - x_i^t\beta - \beta_0) \leq \epsilon \end{cases} \end{aligned}$$

If not all observations lie within an ϵ band around the regression line, the problem does not have a solution. As before, the slack variables, ζ_i and ζ_i^*, are introduced, which allows for observations to lie outside the ϵ band around the regression line

$$\begin{aligned} &\text{minimize } \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{n}(\zeta_i + \zeta_i^*) \\ &\text{subject to } \begin{cases} y_i - x_i^t\beta - \beta_0 & \leq \epsilon + \zeta_i \\ -(y_i - x_i^t\beta - \beta_0) & \leq \epsilon + \zeta_i^* \\ \zeta_i, \zeta_i^* & \geq 0 \end{cases} \end{aligned} \tag{8}$$

whereas before, $C > 0$ regulated how strongly to avoid observations outside the ϵ band. Figure 3 illustrates this. All observations on and outside the ϵ band around the regression line are support vectors. The loss is shown for one of those observations. The fit line is similar, but not identical to the regression line obtained from Gaussian regression. It is not identical because Gaussian regression uses all observations and not just the support vectors and because Gaussian regression uses a quadratic loss rather than the ϵ-insensitive linear loss. ϵ is specified with `epsilon(#)`, and C is still specified by `c(#)`.

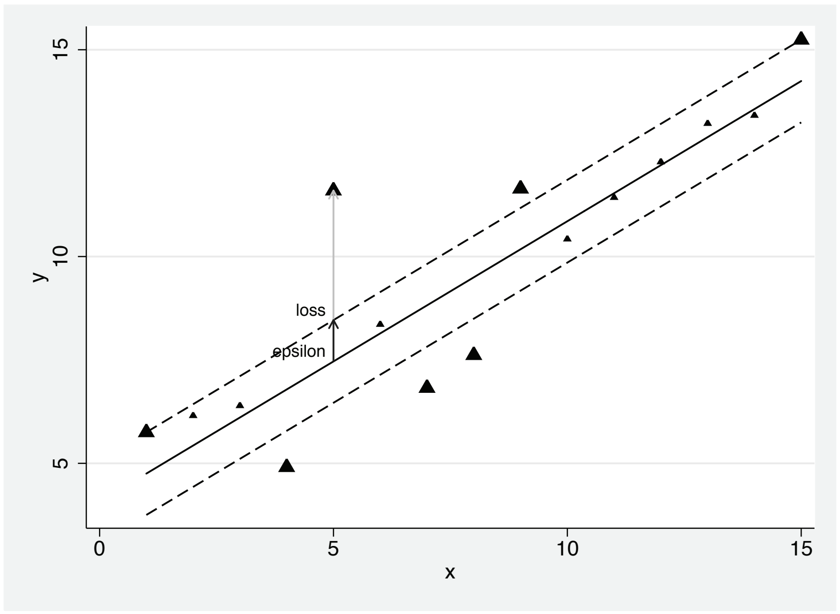

Figure 3. ϵ-insensitive loss in regression. Support vectors are marked with larger plotting symbols. The arrows show the loss for one of the observations.

Again, how to solve the optimization problem in (8) is not as important as how the solution depends only on a restricted set of observations, the support vectors. In this case, the support vectors are those that do not fall within the ϵ boundary. Using kernels greatly facilitates the computations.

Analogous to support vector classification, kernels can be used to achieve nonlinear regressions, and all the same `kernel()` options as under `type(svc)` are available under `type(svr)`.

3.5 Alternative approaches and options

Not every program option is related to the above mathematics.

There is a second kind of SVM known as the ν variant (Schölkopf et al. 2000; Chen, Lin, and Schölkopf 2005). This SVM uses a different parameterization to solve the same problem. In classic SVM, the range of tuning parameters C or ϵ is zero to infinity; in ν-SVM, ν is always in the interval $[0, 1]$. It is related to the proportion of support vectors and therefore controls the model complexity and running time. `type(nu_svc)` implements classification, and `nu(#)` should be used instead of `c(#)`. `type(nu_svr)` implements regression, and `nu(#)` should be used instead of `epsilon(#)`.

Besides classification and regression, there is also an algorithm, `type(one_class)`, that estimates the support of a distribution (Schölkopf et al. 2001) whose predictions are binary indicators of whether a point is in the distribution (1) or is an outlier (0).

The `sv(`*newvar*`)` option adds an extra column called *newvar* during fitting and fills it with a binary indicator of whether each observation was a support vector (1) or not (0).

The `shrinking` option activates a heuristic technique to try to speed up computations by temporarily eliminating variables from consideration (Joachims 1999), while `tolerance()` controls the stopping tolerance for the quadratic program solver in the rare case that it needs controlling.

`verbose` adds more detailed output during computation. `cache_size()` controls how many megabytes of RAM are used to speed up computation by caching kernel values and can generally be set as large as your system can handle to no ill effect.

The `svmachines` command will often issue a warning that `matsize` needs to be increased. A larger `matsize` is necessary to store the fitted model's coefficients of support vectors in `e(sv_coef)`. Before running `svmachines`, you may run something like

```
. set matsize 10000
```

However, most users will never need these coefficients and will not need to run this. All other results will be correct without increasing `matsize`.

3.6 Tuning parameters

Unlike traditional regression methods, machine-learning techniques typically are underconstrained, leaving so-called tuning parameters to be chosen by the user. Depending upon the application, these parameters might be chosen manually or automatically. For example, in a clustering algorithm, the number of clusters is a tuning parameter and might be fixed to five to make the clustering more interpretable; in regularized (LASSO, Ridge) Gaussian regression, the bias-variance tradeoff is controlled by the regularization parameter and would typically be chosen algorithmically. In SVM, C, ϵ, γ, and β_0 are tuning parameters, depending on the particular chosen combination of `type()` and `kernel()`. Arguably, d is also a tuning parameter when using a polynomial kernel, depending on whether it is considered a part of the model definition.

To "tune" these parameters automatically, we search for the values that give a high quality of fit. A well-fit model should not just provide a good prediction on the data it was fit to, it should also generalize to data not yet seen. For a fixed choice of tuning parameters, we can estimate this generalization accuracy with a technique called cross-validation. The simplest form of cross-validation is as follows: the data are separated into a training set, and a test set. The algorithm fits on the training set, and the accuracy (for example, the percent correctly classified or the mean squared error [MSE]) is evaluated on the test set, giving an estimate of how the fit generalizes. More accurate and computationally intensive forms of cross-validation exist (Hastie, Tibshirani, and Friedman 2009, 241).

Tuning is performed for varying values of the tuning parameters, searching for those that give the best generalization accuracy. This can be done by choosing a small number of possible values to test for each parameter and trying all possibilities on the grid of their combinations. This is known as a grid search.

We do not provide grid search or cross-validation code with the `svmachines` package. For the time being, this process will need to be manually written every time, but see the examples below for code outlining the process.

4 Example: Predicting an indicator of high income in the LISS panel

We now give an example for support vector classification. Because respondents in surveys may be reluctant to report their income, income categorizations often have a sizable proportion of missing values. As part of an imputation strategy, prediction may be useful. Here we compare predictions of SVM with logistic regression of an indicator of high income in the Longitudinal Internet Studies for the Social Sciences (LISS) panel.

The LISS panel is an open-access Internet panel based on a probability sample of households drawn from the Dutch population register in 2007. Refreshment samples were added in 2009 and again in 2010–2011. Households that could not otherwise participate were provided a computer and an Internet connection. Respondents were paid an incentive of 15 Euro per hour (and proportionally less for shorter surveys).

Here we use the July 2015 dataset of background variables, which includes income. (The data are downloadable from http://www.lissdata.nl/dataarchive/.) The y variable was an indicator of whether the respondent had a net monthly income of 2,000 Euro or more. Dutch respondents recall net monthly income more easily than gross income or yearly income. Nearly 40% of answers to gross monthly income were categorized as “I don’t know” compared with about 6% for net monthly income, and 20.4% of respondents with known net monthly income reported earning more than 2,000 Euro a month. We used the traditional encoding for y using values 0 and 1, but any other two values, such as 1 and 2 or -1 and 1, would give identical results.

We used the following x variables: female gender, age, marital status (married, separated, divorced, widowed, never been married), education (9 categories), employment status (14 categories, including paid employment, attending school, retired), number of living-at-home children, number of household members, home ownership (owned, rented, cost-free, missing), migration background (Dutch, first-generation Western, second-generation Western, first-generation non-Western, second-generation non-Western), urbanicity of home (5 levels), household head lives with a partner (indicator), domestic situation (single, cohabitation with children, cohabitation without children, single with children, other), individual’s position within the household (8 categories, including head of household, wedded partner, and unwedded partner), number of household members, and age of household head.

Indicator variables were created for all categorical variables. The two age variables (respondent and head of household) and the two count variables (household members and children in household) were standardized to have mean zero and standard deviation one. For clarity, the standardization of variables `aantalhh`, `lftdhhh`, `leeftijd`, and `aantalki` can be accomplished as follows:

```
foreach var of varlist aantalhh lftdhhh leeftijd aantalki {
   qui sum `var´
   generate `var´_sd = (`var´ - r(mean) ) / r(sd)
}
```

There were 10,243 respondents with nonmissing values for net monthly income. The training data consisted of 5,000 random observations, and the remainder was used as test data.

We first use support vector classification (`type(svc)`) with the default radial-basis-function (RBF) kernel (`kernel(rbf)`). To find the tuning parameters yielding the highest accuracy on the test data, we conducted a grid search for all combinations of $C \in \{0.001; 0.01; 1; 10; 100; 1,000; 10,000\}$ and $\gamma \in \{0.0001; 0.001; 0.01; 0.1; 1; 10\}$.

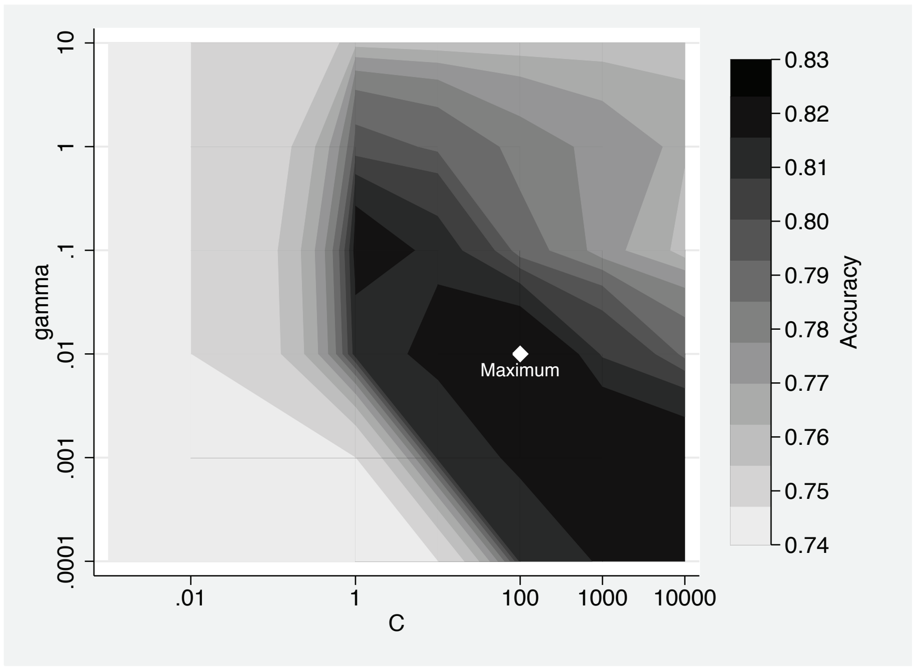

Figure 4. Contour plot of accuracy as a function of tuning parameters C and γ. Accuracy is computed on a test dataset.

Figure 4 gives a contour plot of accuracy, or percentage of correctly classified observations, as a function of C and γ. The highest accuracy on the grid, 0.823, is achieved with $\gamma = 0.01$ where $C = 100$. The SVM command with these parameters is

```
svmachines y `xvars´ if train, c(100) gamma(.1) type(svc) kernel(rbf)
predict yhat
```

where the macro `'xvars'` contained the x variables and the variable `train` is an indicator for the 5,000 observations in the training data. You can refine the grid further or use a more sophisticated method for tuning the parameters, but this probably would not yield a substantial improvement in terms of accuracy. The prediction gives predicted values, not probabilities. To get pseudoprobabilities, you must alter the command:

```
svmachines y `xvars´ if train, c(10) gamma(.1) type(svc) kernel(rbf) prob
predict yhat, prob
```

The `probability` option has to appear in `svmachines` for it to work in the `predict` statement.

For comparison, logistic regression achieves an accuracy of 0.805. The increase in accuracy for SVM, 1.8%, is modest. In our experience for binary outcomes, modest improvements relative to logistic regression are typical.

We standardized the four nonindicator variables. This did not matter much in this particular application, but it is good practice.

5 Regression example

Following Schonlau (2005), we simulated 2,000 observations from the following model,

$$y = 30(x_1 - 0.5)^2 + 2{x_2}^{-0.5} + x_3 + \epsilon$$

where $\epsilon \sim \text{Uniform}(0,1)$ and $0 \le x_i \le 1$ for $i \in \{1,2,3\}$. To keep things relatively simple, we chose the model to be additive without interactions. It is quadratic in x_1, nonlinear in x_2, and linear with a small slope in x_3. The nonlinear contribution of x_2 is stronger than the linear contribution of x_3, even though their slopes are similar. A fourth variable, x_4, is unrelated to the response but is used in the analysis to try to confuse the learning algorithm. Scatterplots of y versus x_1 through x_4 are shown in figure 5.

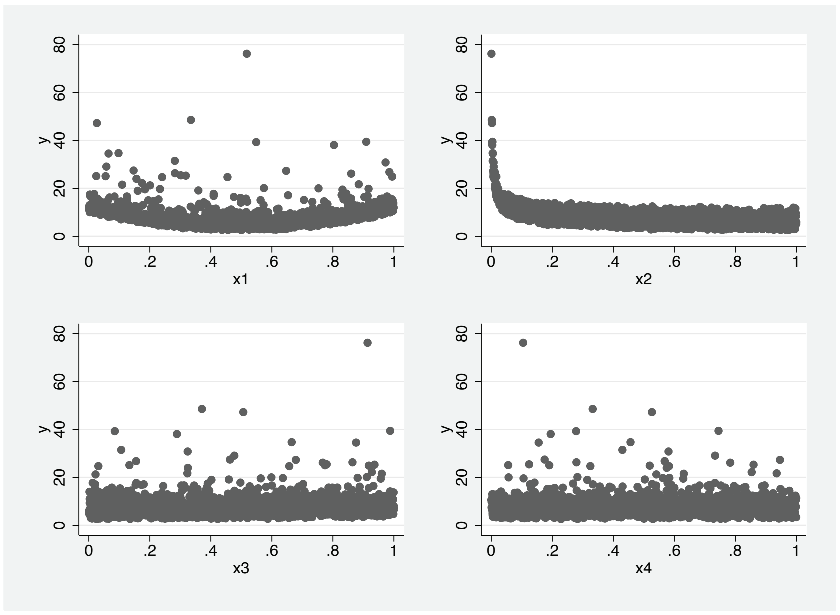

Figure 5. Scatterplots of y versus x_1 through x_4

We first conducted a grid search of tuning parameter values to find the combination of values that yields the lowest MSE on a test dataset. The test dataset consists of a random half of the 2,000 observations. The contour plot in figure 6 shows the MSE as a function of C and γ for a fixed ϵ. We found that for this dataset, the value of ϵ had little effect on the MSE.

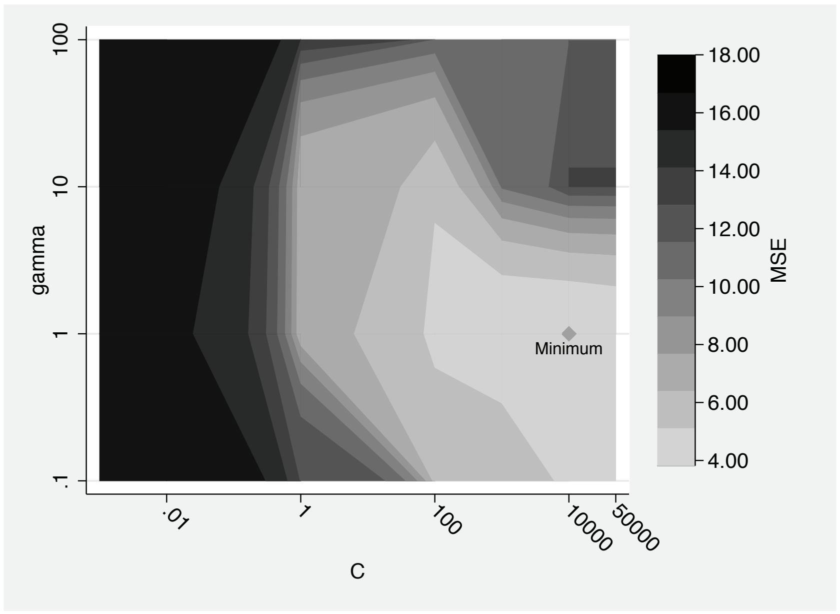

Figure 6. Contour plot of MSE as a function of tuning parameters c and γ for $\epsilon = 1$. The MSE is computed on a test dataset.

The combination producing the lowest MSE is $c = 10000$, $\gamma = 1$, and $\epsilon = 1$. For this combination, the MSE on the test data is 3.82. In comparison, the MSE of linear regression is 12.39 on the same test data. The contour plot also reveals about half the region shown has an MSE that is larger than 12.39. This underlines the importance of tuning SVM parameters before prediction. For clarity, the `svmachines` command with these parameters is

```
svmachines y x1 x2 x3 x4 if train, c(10000) gamma(1) eps(1) type(svr)
predict pred_svm
```

The MSE can be computed from the predicted and true y values,

```
egen mse=total(((y-pred_svm)^2)/`testn´) if !test
```

where `test` is an indicator variable of whether the observation is in the test data and `‘testn’` refers to the number of test observations. Figure 7 shows a calibration plot—a scatterplot of fit versus actual values—for SVM and linear regression. The SVM model is much better calibrated.

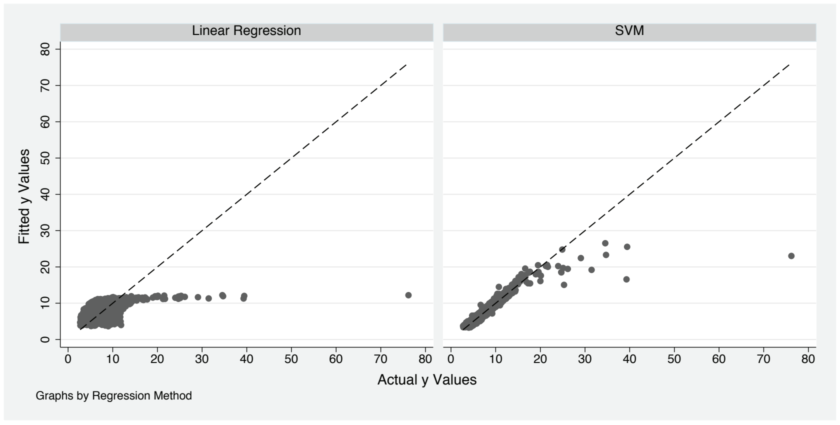

Figure 7. A calibration plot for the linear regression example: fit versus actual values for both linear regression and SVM. The SVM model is much better calibrated.

It is often useful to visualize the black-box function in some way. Conditional plots are an easy way of doing so. Figure 8 gives conditional plots—scatterplots of predicted values for x_1—while other variables are held constant at $x_2 = x_3 = x_4 = 0.5$. Conditional plots for the other x variables are constructed analogously. For clarity, code for constructing this conditional plot is in the appendix. The conditional plots visualize the effects of the individual variables well.

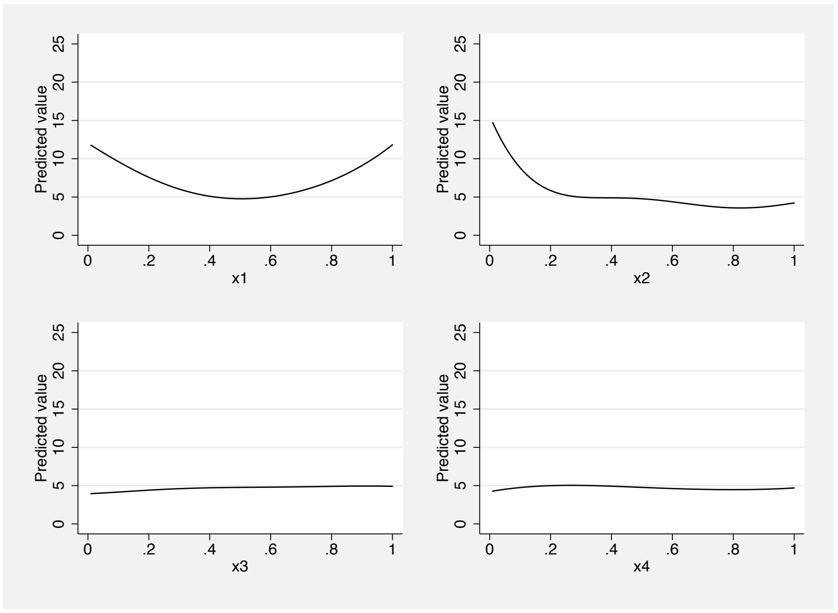

Figure 8. Conditional plots: Predictions for x_1 through x_4 while other variables are held constant at $x_i = 0.5$

6 Discussion

SVM can be a powerful tool, but with great power comes great responsibility. Several issues should be remembered: First, the tuning of parameters is very important. As we have seen in the regression example, an unfortunate choice of tuning parameters can lead to much worse MSE than linear regression. It is safer to work with Gaussian and logistic regression than with an untuned SVM model. Second, if x variables span vastly different ranges, it is safer to standardize them to have mean zero and standard deviation one (for example, Ben-Hur and Weston [2010, sec. 8]). Alternatively, computer scientists tend to standardize the variables to a range such as $[0, 1]$. Either method should work fine. We are not aware that one method is superior to the other. Third, which kernel should be chosen? Our default choice is the RBF kernel because it performs well in many situations. RBF also has fewer tuning parameters (C and γ) than the polynomial kernel (C, degree, γ, and β_0). A linear kernel might be suitable when both the number of observations and the number of variables are very large (Hsu, Chang, and Lin 2003). This problem occurs, for example, in document classification.

SVM is among the best-performing statistical-learning or machine-learning algorithms in terms of accuracy (Maroco et al. 2011), but there is no guarantee it will perform better than other learning techniques. Particularly for binary outcomes, logistic regression and SVM will tend to give similar results (Hastie 2003; Verplancke et al. 2008) because their implied loss function is similar (James et al. 2013, sec. 9.5).

We have previously implemented another important learning technique, gradient boosting (Schonlau 2005). However, statistical or machine learning has not been a primary past focus in Stata. We hope the role of statistical or machine learning in Stata will continue to expand.

7 Acknowledgments

This research was supported by the Social Sciences and Humanities Research Council of Canada (# 435-2013-0128). We are grateful to Chih-Chung Chang and Chih-Jen Lin for programming `libsvm` and making it available. In this article, we make use of data of the LISS panel administered by CentERdata (Tilburg University, The Netherlands). We are grateful for the data.

8 References

Ben-Hur, A., and J. Weston. 2010. A user's guide to support vector machines. In *Data Mining Techniques for the Life Sciences*, ed. O. Carugo and F. Eisenhaber, 223–239. New York: Humana Press.

Chang, C.-C., and C.-J. Lin. 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology* 2(3): Article 27.

Chen, P.-H., C.-J. Lin, and B. Schölkopf. 2005. A tutorial on ν-support vector machines. *Applied Stochastic Models in Business and Industry* 21: 111–136.

Cortes, C., and V. Vapnik. 1995. Support-vector networks. *Machine Learning* 20: 273–297.

Hastie, T. 2003. Support vector machines, kernel logistic regression, and boosting. http://web.stanford.edu/~hastie/Papers/svmtalk.pdf.

Hastie, T., R. Tibshirani, and J. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* 2nd ed. New York: Springer.

Hsu, C.-W., C.-C. Chang, and C.-J. Lin. 2003. A practical guide to support vector classification. http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf.

Hsu, C.-W., and C.-J. Lin. 2002. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks* 13: 415–425.

James, G., D. Witten, T. Hastie, and R. Tibshirani. 2013. *An Introduction to Statistical Learning: With Applications in R.* New York: Springer.

Joachims, T. 1999. Making large-scale support vector machine learning practical. In *Advances in Kernel Methods: Support Vector Learning*, ed. B. Schölkopf, C. J. C. Burges, and A. J. Smola, 169–184. Cambridge, MA: MIT Press.

Lin, C.-J. 2015. LIBSVM FAQ. http://www.csie.ntu.edu.tw/~cjlin/libsvm/faq.html.

Lin, H.-T., and C.-J. Lin. 2003. A study on sigmoid kernels for SVM and the training of non-PSD kernels by SMO-type methods. http://www.csie.ntu.edu.tw/~cjlin/papers/tanh.pdf.

Lin, H.-T., C.-J. Lin, and R. C. Weng. 2007. A note on Platt's probabilistic outputs for support vector machines. *Machine Learning* 68: 267–276.

Maroco, J., D. Silva, A. Rodrigues, M. Guerreiro, I. Santana, and A. de Mendonça. 2011. Data mining methods in the prediction of Dementia: A real-data comparison of the accuracy, sensitivity and specificity of linear discriminant analysis, logistic regression, neural networks, support vector machines, classification trees and random forests. *BMC Research Notes* 4: Article 299.

Platt, J. C. 2000. Probabilities for SV machines. In *Advances in Large Margin Classifiers*, ed. A. J. Smola, P. L. Bartlett, B. Schölkopf, and D. Schuurmans, 61–74. Cambridge, MA: MIT Press.

Schölkopf, B., J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. 2001. Estimating the support of a high-dimensional distribution. *Neural Computation* 13: 1443–1471.

Schölkopf, B., A. J. Smola, R. C. Williamson, and P. L. Bartlett. 2000. New support vector algorithms. *Neural Computation* 12: 1207–1245.

Schonlau, M. 2005. Boosted regression (boosting): An introductory tutorial and a Stata plugin. *Stata Journal* 5: 330–354.

Smola, A. J., and B. Schölkopf. 2004. A tutorial on support vector regression. *Statistics and Computing* 14: 199–222.

Vapnik, V. N. 2000. *The Nature of Statistical Learning Theory*. 2nd ed. New York: Springer.

Verplancke, T., S. Van Looy, D. Benoit, P. Vansteelandt, P. Depuydt, F. De Turck, and J. Decruyenaere. 2008. Support vector machine versus logistic regression modeling for prediction of hospital mortality in critically ill patients with haematological malignancies. *BMC Medical Informatics and Decision Making* 8: Article 56.

Wu, T.-F., C.-J. Lin, and R. C. Weng. 2004. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research* 5: 975–1005.

About the authors

Nick Guenther is an undergraduate in the Department of Statistics and the School of Computer Science at the University of Waterloo in Canada. His interests include feature learning, functional programming, and applying machine learning toward new technologies.

Matthias Schonlau is a professor in the Department of Statistics and Actuarial Sciences at the University of Waterloo in Canada. His interests include survey methodology and text mining of open-ended questions.

Appendix

Code for creating the conditional plot in the regression example. The code assumes the model is trained on the first 1,000 observations.

```
use toy_normal
drop if _n>1000
set obs 1400
replace x1=0.5 if _n>1000
replace x2=0.5 if _n>1000
replace x3=0.5 if _n>1000
replace x4=0.5 if _n>1000
replace x1= (_n-1000)/100 if _n>1000 & _n<=1100
replace x2= (_n-1100)/100 if _n>1100 & _n<=1200
replace x3= (_n-1200)/100 if _n>1200 & _n<=1300
replace x4= (_n-1300)/100 if _n>1300 & _n<=1400
capture drop pred
svmachines y x1 x2 x3 x4 in 1/1000, c(10000) gamma(1) eps(1) type(svr)
predict pred
line pred x1 if _n>1000 & _n<=1100, ylabel(0 5 10 to 25) ///
   ytitle("Predicted value")
graph save "x1_pred", replace
line pred x2 if _n>1100 & _n<=1200, ylabel(0 5 10 to 25) ///
   ytitle("Predicted value")
graph save "x2_pred", replace
line pred x3 if _n>1200 & _n<=1300, ylabel(0 5 10 to 25) ///
   ytitle("Predicted value")
graph save "x3_pred", replace
line pred x4 if _n>1300 & _n<=1400, ylabel(0 5 10 to 25) ///
   ytitle("Predicted value")
graph save "x4_pred", replace
graph combine  x1_pred.gph x2_pred.gph x3_pred.gph x4_pred.gph
```