Support Vector Machines *

The Interface to `libsvm` in package `e1071`

by David Meyer
FH Technikum Wien, Austria
David.Meyer@R-Project.org

October 23, 2022

"Hype or Hallelujah?" is the provocative title used by Bennett & Campbell (2000) in an overview of Support Vector Machines (SVM). SVMs are currently a hot topic in the machine learning community, creating a similar enthusiasm at the moment as Artificial Neural Networks used to do before. Far from being a panacea, SVMs yet represent a powerful technique for general (nonlinear) classification, regression and outlier detection with an intuitive model representation.

The package `e1071` offers an interface to the award-winning[1] C++-implementation by Chih-Chung Chang and Chih-Jen Lin, `libsvm` (current version: 2.6), featuring:

- C- and ν-classification
- one-class-classification (novelty detection)
- ϵ- and ν-regression

and includes:

- linear, polynomial, radial basis function, and sigmoidal kernels
- formula interface
- k-fold cross validation

For further implementation details on `libsvm`, see Chang & Lin (2001).

Basic concept

SVMs were developed by Cortes & Vapnik (1995) for binary classification. Their approach may be roughly sketched as follows:

Class separation: basically, we are looking for the optimal separating hyperplane between the two classes by maximizing the *margin* between the classes' closest points (see Figure 1)—the points lying on the boundaries are called *support vectors*, and the middle of the margin is our optimal separating hyperplane;

*A smaller version of this article appeared in R-News, Vol.1/3, 9.2001

[1] The library won the IJCNN 2001 Challenge by solving two of three problems: the Generalization Ability Challenge (GAC) and the Text Decoding Challenge (TDC). For more information, see: http://www.csie.ntu.edu.tw/~cjlin/papers/ijcnn.ps.gz.

Overlapping classes: data points on the "wrong" side of the discriminant margin are weighted down to reduce their influence (*"soft margin"*);

Nonlinearity: when we cannot find a *linear* separator, data points are projected into an (usually) higher-dimensional space where the data points effectively become linearly separable (this projection is realised via *kernel techniques*);

Problem solution: the whole task can be formulated as a quadratic optimization problem which can be solved by known techniques.

A program able to perform all these tasks is called a *Support Vector Machine*.

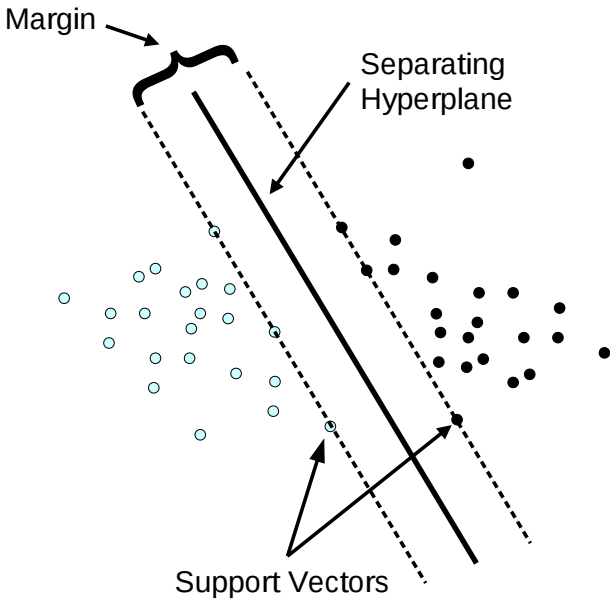

Figure 1: Classification (linear separable case)

Several extensions have been developed; the ones currently included in `libsvm` are:

ν-classification: this model allows for more control over the number of support vectors (see Schölkopf et al., 2000) by specifying an additional parameter ν which approximates the fraction of support vectors;

One-class-classification: this model tries to find the support of a distribution and thus allows for outlier/novelty detection;

Multi-class classification: basically, SVMs can only solve binary classification problems. To allow for multi-class classification, `libsvm` uses the *one-against-one* technique by fitting all binary subclassifiers and finding the correct class by a voting mechanism;

ϵ-regression: here, the data points lie *in between* the two borders of the margin which is maximized under suitable conditions to avoid outlier inclusion;

ν-regression: with analogue modifications of the regression model as in the classification case.

Usage in R

The R interface to `libsvm` in package `e1071`, `svm()`, was designed to be as intuitive as possible. Models are fitted and new data are predicted as usual, and both the vector/matrix and the formula interface are implemented. As expected for R's statistical functions, the engine tries to be smart about the mode to be chosen, using the dependent variable's type (y): if y is a factor, the engine switches to classification mode, otherwise, it behaves as a regression machine; if y is omitted, the engine assumes a novelty detection task.

Examples

In the following two examples, we demonstrate the practical use of `svm()` along with a comparison to classification and regression forests as implemented in `randomForest()`.

Classification

In this example, we use the glass data from the UCI Repository of Machine Learning Databases for classification. The task is to predict the type of a glass on basis of its chemical analysis. We start by splitting the data into a train and test set:

```
> library(e1071)
> library(randomForest)
> data(Glass, package="mlbench")
> ## split data into a train and test set
> index     <- 1:nrow(Glass)
> N         <- trunc(length(index)/3)
> testindex <- sample(index, N)
> testset   <- Glass[testindex,]
> trainset  <- Glass[-testindex,]
```

Both for SVM and randomForest (via `randomForest()`), we fit the model and predict the test set values:

```
> ## svm
> svm.model <- svm(Type ~ ., data = trainset, cost = 100, gamma = 1)
> svm.pred  <- predict(svm.model, testset[,-10])
```

(The dependent variable, `Type`, has column number 10. `cost` is a general penalizing parameter for C-classification and `gamma` is the radial basis function-specific kernel parameter.)

```
> ## randomForest
> rf.model <- randomForest(Type ~ ., data = trainset)
> rf.pred  <- predict(rf.model, testset[,-10])
```

A cross-tabulation of the true versus the predicted values yields:

```
> ## compute svm confusion matrix
> table(pred = svm.pred, true = testset[,10])

    true
pred  1  2  3  5  6  7
   1 19  3  4  0  0  0
   2  9 18  1  2  4  2
   3  0  0  0  0  0  0
   5  0  0  0  3  0  0
   6  0  0  0  0  2  0
   7  0  0  0  0  0  4

> ## compute randomForest confusion matrix
> table(pred = rf.pred, true = testset[,10])

    true
pred  1  2  3  5  6  7
   1 24  0  2  0  0  0
   2  4 18  0  1  2  0
   3  0  1  3  0  0  0
   5  0  2  0  3  0  0
   6  0  0  0  0  3  0
   7  0  0  0  1  1  6
```

	method	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
Accuracy	svm	0.56	0.6	0.61	0.63	0.67	0.7
	randomForest	0.39	0.43	0.45	0.47	0.52	0.57
Kappa	svm	0.68	0.75	0.77	0.77	0.8	0.85
	randomForest	0.56	0.66	0.68	0.69	0.73	0.79

Table 1: Performance of `svm()` and `randomForest()` for classification (10 replications)

Finally, we compare the performance of the two methods by computing the respective accuracy rates and the kappa indices (as computed by `classAgreement()` also contained in package `e1071`). In Table 1, we summarize the results of 10 replications—Support Vector Machines show better results.

Non-linear ϵ-Regression

The regression capabilities of SVMs are demonstrated on the ozone data. Again, we split the data into a train and test set.

```
> library(e1071)
> library(randomForest)
> data(Ozone, package="mlbench")
> ## split data into a train and test set
> index     <- 1:nrow(Ozone)
> N         <- trunc(length(index)/3)
```

```
> testindex <- sample(index, N)
> testset   <- na.omit(Ozone[testindex,-3])
> trainset  <- na.omit(Ozone[-testindex,-3])
> ## svm
> svm.model <- svm(V4 ~ ., data = trainset, cost = 1000, gamma = 0.0001)
> svm.pred  <- predict(svm.model, testset[,-3])
> sqrt(crossprod(svm.pred - testset[,3]) / N)

         [,1]
[1,] 3.653846

> ## random Forest
> rf.model <- randomForest(V4 ~ ., data = trainset)
> rf.pred  <- predict(rf.model, testset[,-3])
> sqrt(crossprod(rf.pred - testset[,3]) / N)

         [,1]
[1,] 3.497016
```

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
svm	3.05	3.23	3.43	3.41	3.58	3.73
randomForest	3.09	3.32	3.49	3.57	3.86	4.08

Table 2: Performance of `svm()` and `randomForest()` for regression (Root Mean Squared Error, 10 replications)

We compare the two methods by the root mean squared error (RMSE)—see Table 2 for a summary of 10 replications. Again, as for classification, `svm()` does a better job than `randomForest()`.

Elements of the `svm` object

The function `svm()` returns an object of class "`svm`", which partly includes the following components:

`SV`: matrix of support vectors found;

`labels`: their labels in classification mode;

`index`: index of the support vectors in the input data (could be used e.g., for their visualization as part of the data set).

If the cross-classification feature is enabled, the `svm` object will contain some additional information described below.

Other main features

Class Weighting: if one wishes to weight the classes differently (e.g., in case of asymmetric class sizes to avoid possibly overproportional influence of bigger classes on the margin), weights may be specified in a vector with named components. In case of two classes A and B, we could use something like: `m <- svm(x, y, class.weights = c(A = 0.3, B = 0.7))`

Cross-classification: to assess the quality of the training result, we can perform a k-fold cross-classification on the training data by setting the parameter `cross` to k (default: 0). The `svm` object will then contain some additional values, depending on whether classification or regression is performed. Values for classification:

`accuracies:` vector of accuracy values for each of the k predictions

`tot.accuracy:` total accuracy

Values for regression:

`MSE:` vector of mean squared errors for each of the k predictions

`tot.MSE:` total mean squared error

`scorrcoef:` Squared correlation coefficient (of the predicted and the true values of the dependent variable)

Tips on practical use

- Note that SVMs may be very sensitive to the proper choice of parameters, so allways check a range of parameter combinations, at least on a reasonable subset of your data.
- For classification tasks, you will most likely use C-classification with the RBF kernel (default), because of its good general performance and the few number of parameters (only two: C and γ). The authors of `libsvm` suggest to try small and large values for C—like 1 to 1000—first, then to decide which are better for the data by cross validation, and finally to try several γ's for the better C's.
- However, better results are obtained by using a grid search over all parameters. For this, we recommend to use the `tune.svm()` function in `e1071`.
- Be careful with large datasets as training times may increase rather fast.
- Scaling of the data usually drastically improves the results. Therefore, `svm()` scales the data by default.

Model Formulations and Kernels

Dual representation of models implemented:

- C-classification:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^\top \mathbf{Q}\alpha - \mathbf{e}^\top \alpha \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C,\ i = 1, \ldots, l, \\ & \mathbf{y}^\top \alpha = 0\ , \end{aligned} \tag{1}$$

where $\mathbf{e}$ is the unity vector, C is the upper bound, $\mathbf{Q}$ is an l by l positive semidefinite matrix, $Q_{ij} \equiv y_i y_j K(x_i, x_j)$, and $K(x_i, x_j) \equiv \phi(x_i)^\top \phi(x_j)$ is the kernel.

- ν-classification:

$$\begin{array}{ll} \min\limits_{\alpha} & \frac{1}{2}\alpha^\top \mathbf{Q}\alpha \\ \text{s.t.} & 0 \le \alpha_i \le 1/l,\ i = 1, \ldots, l, \\ & \mathbf{e}^\top \alpha \ge \nu, \\ & \mathbf{y}^\top \alpha = 0\ . \end{array} \tag{2}$$

  where $\nu \in (0, 1]$.

- one-class classification:

$$\begin{array}{ll} \min\limits_{\alpha} & \frac{1}{2}\alpha^\top \mathbf{Q}\alpha \\ \text{s.t.} & 0 \le \alpha_i \le 1/(\nu l),\ i = 1, \ldots, l, \\ & \mathbf{e}^\top \alpha = 1\ , \end{array} \tag{3}$$

- ϵ-regression:

$$\begin{array}{ll} \min\limits_{\alpha, \alpha^*} & \frac{1}{2}(\alpha - \alpha^*)^\top \mathbf{Q}(\alpha - \alpha^*) + \\ & \epsilon \sum\limits_{i=1}^{l}(\alpha_i + \alpha_i^*) + \sum\limits_{i=1}^{l} y_i(\alpha_i - \alpha_i^*) \\ \text{s.t.} & 0 \le \alpha_i, \alpha_i^* \le C,\ i = 1, \ldots, l, \\ & \sum\limits_{i=1}^{l}(\alpha_i - \alpha_i^*) = 0\ . \end{array} \tag{4}$$

- ν-regression:

$$\begin{array}{ll} \min\limits_{\alpha, \alpha^*} & \frac{1}{2}(\alpha - \alpha^*)^\top \mathbf{Q}(\alpha - \alpha^*) + \mathbf{z}^\top(\alpha_i - \alpha_i^*) \\ \text{s.t.} & 0 \le \alpha_i, \alpha_i^* \le C,\ i = 1, \ldots, l, \\ & \mathbf{e}^\top(\alpha - \alpha^*) = 0 \\ & \mathbf{e}^\top(\alpha + \alpha^*) = C\nu\ . \end{array} \tag{5}$$

Available kernels:

kernel	formula	parameters
linear	$\mathbf{u}^\top \mathbf{v}$	(none)
polynomial	$(\gamma \mathbf{u}^\top \mathbf{v} + c_0)^d$	γ, d, c_0
radial basis fct.	$\exp\{-\gamma \lvert \mathbf{u} - \mathbf{v} \rvert^2\}$	γ
sigmoid	$\tanh\{\gamma \mathbf{u}^\top \mathbf{v} + c_0\}$	γ, c_0

Conclusion

We hope that `svm` provides an easy-to-use interface to the world of SVMs, which nowadays have become a popular technique in flexible modelling. There are some drawbacks, though: SVMs scale rather badly with the data size due to the quadratic optimization algorithm and the kernel transformation. Furthermore, the correct choice of kernel parameters is crucial for obtaining good results, which practically means that an extensive search must be conducted on the parameter space before results can be trusted, and this often complicates the task (the authors of `libsvm` currently conduct some work on methods of efficient automatic parameter selection). Finally, the current implementation is optimized for the radial basis function kernel only, which clearly might be suboptimal for your data.

References

Bennett, K. P. & Campbell, C. (2000). Support vector machines: Hype or hallelujah? *SIGKDD Explorations*, **2**(2). http://www.acm.org/sigs/sigkdd/explorations/issue2-2/bennett.pdf.

Chang, C.-C. & Lin, C.-J. (2001). LIBSVM: a library for support vector machines. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm, detailed documentation (algorithms, formulae, ...) can be found in http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.ps.gz

Cortes, C. & Vapnik, V. (1995). Support-vector network. *Machine Learning*, **20**, 1–25.

Schölkopf, B., Smola, A., Williamson, R. C., & Bartlett, P. (2000). New support vector algorithms. *Neural Computation*, **12**, 1207–1245.

Vapnik, V. (1998). *Statistical learning theory*. New York: Wiley.