a feature that allows users to specify conversion routines for certain types of objects that are handled via the `.Call()` (see http://cm.bell-labs.com/stat/duncan/SCConverters).

A potentially important use of the registration mechanism relates to security, and specifically prohibiting some users calling certain native routines that have access to sensitive data. We have been developing packages that embed R within spreadsheets such as Gnumeric and Excel; Web browsers such as Netscape; relational databases such as Postgres; and so on. One benefit of this approach is that one can run R code that is dynamically downloaded from the Web. However, as we all know, this is a common way to download viruses and generally make ones machine vulnerable. Using the registration mechanism, developers can mark their routines as being vulnerable and to be used only in "secure" sessions. What this means exactly remains to be defined!

Building the table automatically

This registration mechanism offers all the advantages that we have mentioned above. However, it requires a little more work by the developer. Since the original lookup mechanism still works, many developers may not take the time to create the arrays of routine definitions and register them. It would be convenient to be able to generate the registration code easily and without a lot of manual effort by the developer.

The **Slcc** (http://www.omegahat.org/Slcc/) package from the Omegahat project provides a general mechanism for processing C source code and returning information about the data structures, variables and routines it contains. This information is given as S objects and can be used to generate C code. The package provides a function to read both the S and C code of a library and generate the C code to register (only) the routines that are referenced in the S code.

The **Slcc** package is in the early stages of development. It runs on Linux, but there are some minor installation details to be worked out for other platforms.

Summary

The new registration mechanism is being used in the R packages within the core R distribution itself and seems to be working well. We hope some of the benefits are obvious. We expect that others will appear over time when we no longer have to deal with subtle differences in the behavior of various operating systems and how to handle dynamically loaded code. The only extra work that developers have to do is to explicitly create the table of routines that are to be registered with R. The availability of the **Slcc** package will hopefully help to automate the creation of the registration code and make it a trivial step. We are very interested in peoples' opinions and suggestions.

Duncan Temple Lang
Bell Labs, Murray Hill, New Jersey, U.S.A
duncan@research.bell-labs.com

Support Vector Machines

The Interface to `libsvm` in package e1071

by David Meyer

"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.

Package **e1071** offers an interface to the award-winning[1] C++ SVM implementation by Chih-Chung Chang and Chih-Jen Lin, `libsvm` (current version: 2.31), 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).

[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.

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;

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*.

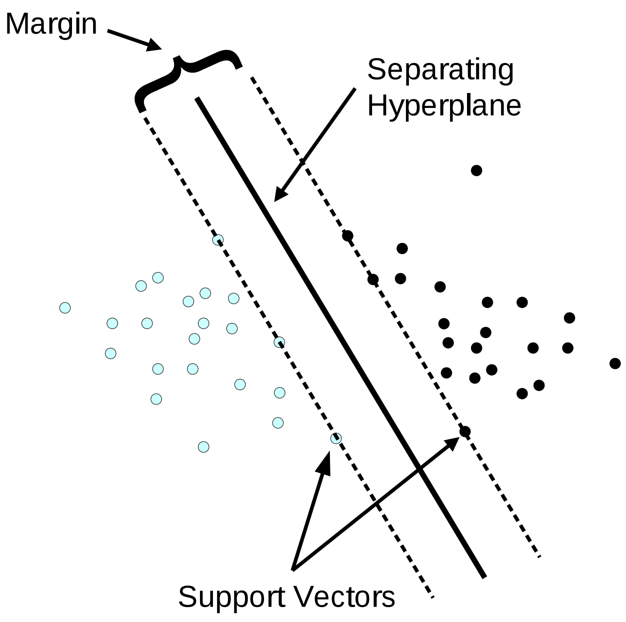

Figure 1: Classification (linear separable case)

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 analogous 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 trees as implemented in `rpart()`.

Classification

In this example, we use the glass data from the UCI Repository of Machine Learning Databases (available in package **mlbench**) 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(rpart)
library(mlbench)
data(Glass)

## split data into a training and test set
index     <- 1:nrow(x)
testindex <- sample(index,
                    trunc(length(index)/3))
testset   <- x[testindex,]
trainset  <- x[-testindex,]
```

Both for the SVM and the partitioning tree (via `rpart()`), we fit the model and try to 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 parameter for *C*-classification and `gamma` is the radial basis function-specific kernel parameter.)

```
## rpart
rpart.model <- rpart(Type ~ ., data = trainset)
rpart.pred  <- predict(rpart.model,
                testset[,-10], type = "class")
```

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 8  7 2 0 0 0
   2 5 19 0 0 1 0
   3 3  3 2 0 0 0
   5 0  4 0 2 2 0
   6 0  0 0 0 3 0
   7 2  0 0 0 0 8

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

    true
pred 1  2 3 5 6 7
   1 8 10 2 2 2 0
   2 9 17 1 0 2 0
   3 0  4 1 0 0 0
   5 0  1 0 0 2 0
   6 0  0 0 0 0 0
   7 1  1 0 0 0 8
```

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 100 replications: `svm()` seems to perform slightly better than `rpart()`.

Non-linear ϵ-regression

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

```
library(e1071)
library(rpart)
library(mlbench)
data(Ozone)

## split data into a training and test set
index     <- 1:nrow(x)
testindex <- sample(index,
                    trunc(length(index)/3))
testset   <- x[testindex,]
trainset  <- x[-testindex,]

## svm
svm.model <- svm(V4 ~ ., data = trainset,
                 cost = 1000, gamma = 0.0001)
svm.pred  <- predict(svm.model, testset[,-4])

## rpart
rpart.model <- rpart(V4 ~ ., data = trainset)
rpart.pred  <- predict(rpart.model, testset[,-4])
```

We compare the two methods by the mean squared error (MSE)—see Table 2. Here, in contrast to classification, `rpart()` does a better job than `svm()`.

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

Index	Method	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
accuracy rate	svm	0.55	0.63	0.68	0.68	0.70	0.79
	rpart	0.49	0.63	0.65	0.66	0.70	0.79
kappa	svm	0.40	0.51	0.56	0.56	0.61	0.72
	rpart	0.33	0.49	0.52	0.53	0.59	0.70

Table 1: Performance of `svm()` and `rpart()` for classification (100 replications)

Method	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
svm	7.8	10.4	11.6	11.9	13.1	17.0
rpart	4.8	7.7	8.8	9.0	10.3	14.2

Table 2: Performance of `svm()` and `rpart()` for regression (mean squared error, 100 replications)

`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 sensible to the proper choice of parameters, so always 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.
- Be careful with large datasets as training times may increase rather fast.

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.

Bibliography

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`. 23

Chang, C.-C. & Lin, C.-J. (2001). Libsvm: a library for support vector machines (version 2.31). `http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf`. 23

Cortes, C. & Vapnik, V. (1995). Support-vector network. *Machine Learning*, **20**, 1–25. 24

Schölkopf, B., Smola, A., Williamson, R. C., & Bartlett, P. (2000). New support vector algorithms. *Neural Computation*, **12**, 1207–1245. 24

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

David Meyer
Technische Universität Wien, Austria
`David.Meyer@ci.tuwien.ac.at`