An Introduction to Support Vector Machine and Implementation in R

Yihui Xie [*]

May 8, 2007

Abstract

Support vector machine (SVM) is a popular technique for classification. This paper mainly focuses on the implementation of SVM in R, besides, we'll make some comparisons among SVM and other methods for classification such as KNN and logistic regression.

1 Introduction

SVM (Support Vector Machine) is a useful technique for data classification. Here we propose an easy approach which usually gives reasonable results, and this preliminary guide is *not* for SVM researchers nor do we guarantee the best accuracy. We also do not intend to solve challenging or difficult problems. Our purpose is to give SVM novices a recipe to obtain acceptable results fast and easily.

Here we briefly introduce SVM basics which are necessary for explaining our procedure. A classification task usually involves with training and testing data which consist of some data instances. Each instance in the training set contains one "target value" (class labels) and several "attributes" (features). The goal of SVM is to produce a model which predicts target value of data instances in the testing set which are given only the attributes. The common approach is to find a real value function[1] $g(x)$ in order that we can predict the classification of y according to the sign of $g(x)$. The "sign", also known as the "discriminant function", is defined as follows:

$$\text{sign}(a) = \begin{cases} 1 & \text{if } a \geq 0 \\ -1 & \text{if } a < 0 \end{cases} \tag{1}$$

[*]School of Statistics, Renmin University of China, Beijing, 100872, China; Email: xieyihui(at)gmail.com; Web: `http://www.yihui.name`

[1]This just means a "rule", which might be either linear (e.g. $g(x) = wx + b$) or nonlinear – it depends on the data

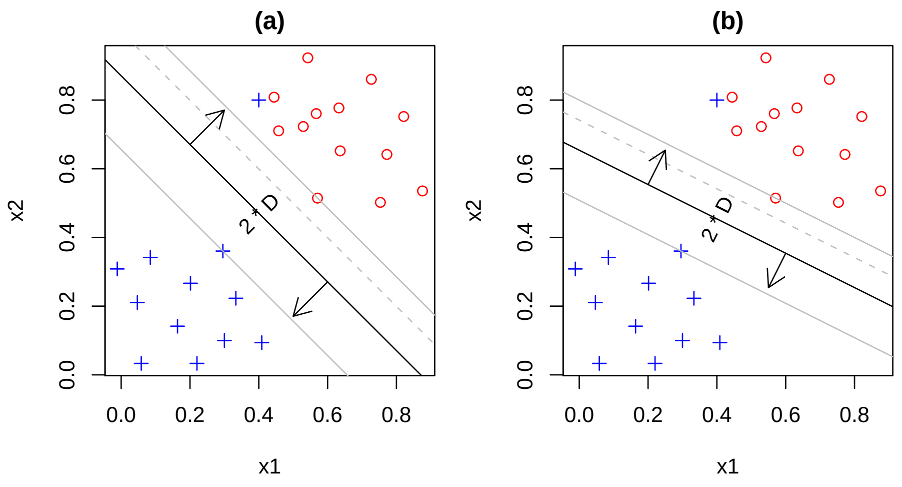

Figure 1: Selection of $g(x)$ through maximizing D

First of all, let's take a look at Figure 1, which is very intuitional for our understanding in the process of finding the best $g(x)$. The points annotated by "+" and "o" are from two separate groups, and now the question is: how to find a line to "divide" these points as far as possible? Before we introduce any mathematical formulae, the reader should be able to judge by intuition that the black line in Figure 1(a) is better than that in Figure 1(b). The reason is simply that the former line separate these points farther than the latter one – this is quite important in applications because it's undesirable for a case (with unknown class label) that can't be classified into a group very *definitely*. According to this consideration, we can also know that the line should be placed just in the *middle* of the two groups' margins. (Why?)

Now our task is clear: find a function $g(x)$ (usually the actual task is to obtain the coefficients) to maximize the distance[2] D between two sample groups. However, points in two groups cannot *always* be separated by just a single straight line. In such overlapping cases, we'll use slack variables to compute the distance between the two margins, i.e. extract the distances denoted by slack variables[3] ξ_i from D first.

Next we'll introduce the formal theory of SVM, taking the linear case of SVM as an example. Here "linear" means $g(x) = \mathbf{x}^T\boldsymbol{\beta} + \beta_0$.

Mathematically speaking, the overall problem can be described by a linear program as: given a training set of instance-label pairs $(\mathbf{x}_i, y_i)$, $i = 1, \cdots, N$

[2]Sure, the actual distance is $2D$ (as annotated in Figure 1), but it is just the same to maximize D

[3]For a point on the correct side of the straight line, $\xi_i = 0$

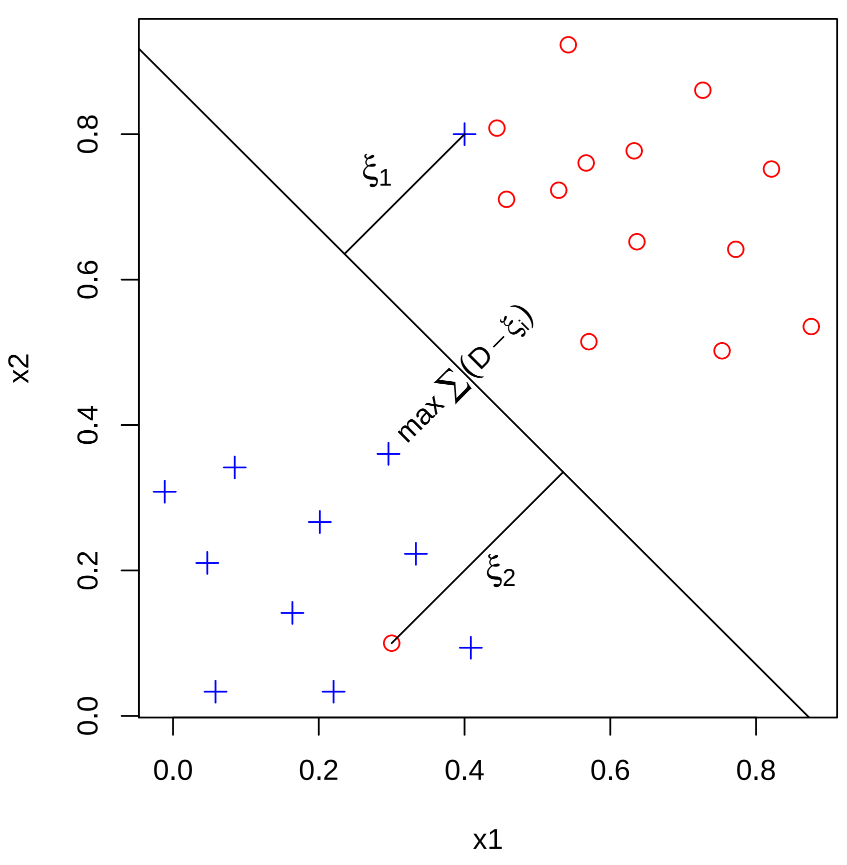

Figure 2: Points that are linearly inseparable by a straight line

where $\mathbf{x}_i \in \mathbb{R}^p$ and $\mathbf{y} \in \{1, -1\}^N$, the support vector machines (SVM) usually require the solution of the following optimization problem:

$$\begin{array}{ll} \max\limits_{\boldsymbol{\beta}, \beta_0, \|\beta\|=1} & D \\ \text{subject to} & y_i(\boldsymbol{\beta}^T \mathbf{x}_i + \beta_0) \geq D, \\ & i = 1, \cdots, N \end{array} \tag{2}$$

With considerations of slack variables and by means of taking the dual program, we can ultimately get an equivalent nonlinear program as:

$$\begin{array}{ll} \min\limits_{\boldsymbol{\beta}, \beta_0, \boldsymbol{\xi}} & \frac{1}{2} \boldsymbol{\beta}^T \boldsymbol{\beta} + C \sum_{i=1}^{N} \xi_i \\ \text{subject to} & y_i(\boldsymbol{\beta}^T \mathbf{x}_i + \beta_0) \geq 1 - \xi_i, \\ & \xi_i \geq 0 \end{array} \tag{3}$$

It's not difficult to generalize this linear program to the nonlinear case replacing $\mathbf{x}_i$ with a nonlinear function $\phi(\mathbf{x}_i)$:

$$\begin{array}{ll} \min\limits_{\boldsymbol{\beta},\beta_0,\boldsymbol{\xi}} & \frac{1}{2}\boldsymbol{\beta}^T\boldsymbol{\beta} + C\sum\limits_{i=1}^{N}\xi_i \\ \text{subject to} & y_i(\boldsymbol{\beta}^T\phi(\mathbf{x}_i) + \beta_0) \geq 1 - \xi_i, \\ & \xi_i \geq 0 \end{array} \tag{4}$$

Here training vectors $\mathbf{x}_i$ are mapped into a higher (maybe infinite) dimensional space by the function $\phi(\cdot)$. Then SVM finds a linear separating hyperplane with the maximal margin in this higher dimensional space. $C > 0$ is the penalty parameter of the error term. Furthermore, $K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ is called the *kernel function*. Though new kernels are being proposed by researchers, beginners may find in SVM books the following four basic kernels, and here we just one of them, i.e. radial basis function (RBF):

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2),\ \gamma > 0 \tag{5}$$

SVM novices may be confused by the concept "kernel function", so what's the use of this function? In fact, this is just a tool for measuring similarity of two sample cases. It's quite natural that if an observation is more "similar" to group 1 than group 2, we'll classify it into group 1. Another question might be "how does it come out", and the answer can be found in Hastie et al (2001). Here we just tell the reader that this kernel function rises from the solution process of the above programs (Wolfe dual).

Actually the theory of SVM is not the critical part of our paper, therefore next we directly go on with issues concerning data analysis after a short introduction of the package e1071 in R.

2 Relative Packages and Functions in R

The package e1071 offers an interface to the award-winning 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).

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.

Below is the usage of *svm()*:

```
## S3 method for class 'formula':
svm(formula, data = NULL, ..., subset, na.action =
na.omit, scale = TRUE)
## Default S3 method:
svm(x, y = NULL, scale = TRUE, type = NULL, kernel =
"radial", degree = 3, gamma = if (is.vector(x)) 1 else 1 / ncol(x),
coef0 = 0, cost = 1, nu = 0.5,
class.weights = NULL, cachesize = 40, tolerance = 0.001, epsilon = 0.1,
shrinking = TRUE, cross = 0, probability = FALSE, fitted = TRUE,
..., subset, na.action = na.omit)
```

Later we'll explain the most frequently used parameters in detail.

3 Dealing with "spam" Data

According to the suggestions by Chang & Lin (2001), below is a suitable procedure for beginners:

- Transform data to the format of an SVM software
- Conduct simple scaling on the data
- Consider the RBF kernel $K(\mathbf{x}, \mathbf{y}) = e^{-\gamma\|\mathbf{x}-\mathbf{y}\|^2}$
- Use cross-validation to find the best parameter C and γ
- Use the best parameter C and to train the whole training set
- Test

Considerations on data-scaling and kernel selection are simply omitted in this paper (though they are very important) and we put our main efforts in the implementation in R.

The "spam" data to be analyzed in this section comes from an R package ElemStatLearn, which is already quite familiar to us. For background on spam please refer to [4].

The last column of the data denotes whether the e-mail was considered spam (1) or not (0), i.e. unsolicited commercial e-mail. Most of the attributes indicate whether a particular word or character was frequently occurring in the e-mail. The run-length attributes (55-57) measure the length of sequences of consecutive capital letters. For the statistical measures of each attribute, see the end of this file. Here are the definitions of the attributes[4]:

- 48 continuous real [0,100] attributes of type `word_freq_WORD` = percentage of words in the e-mail that match WORD, i.e. 100 * (number of times the WORD appears in the e-mail) / total number of words in e-mail. A "word" in this case is any string of alphanumeric characters bounded by non-alphanumeric characters or end-of-string.
- 6 continuous real [0,100] attributes of type `char_freq_CHAR` = percentage of characters in the e-mail that match CHAR, i.e. 100 * (number of CHAR occurences) / total characters in e-mail
- 1 continuous real [1, ...] attribute of type `capital_run_length_average` = average length of uninterrupted sequences of capital letters
- 1 continuous integer [1, ...] attribute of type `capital_run_length_longest` = length of longest uninterrupted sequence of capital letters
- 1 continuous integer [1, ...] attribute of type `capital_run_length_total` = sum of length of uninterrupted sequences of capital letters = total number of capital letters in the e-mail
- 1 nominal $\{0, 1\}$ class attribute of type spam = denotes whether the e-mail was considered spam (1) or not (0), i.e. unsolicited commercial e-mail.

First of all, we may examine the structure of the data using the function *str()* as follows:

```
> data(spam, package = "ElemStatLearn")
> str(spam)
'data.frame':   4601 obs. of  58 variables:
 $ A.1 : num  0 0.21 0.06 0 0 0 0 0 0.15 0.06 ...
 $ A.2 : num  0.64 0.28 0 0 0 0 0 0 0 0.12 ...
 $ A.3 : num  0.64 0.5 0.71 0 0 0 0 0 0.46 0.77 ...
```

[4]For detailed information about these words and characters, see Appendix A

```
$ A.4 : num  0 0 0 0 0 0 0 0 0 0 ...
$ A.5 : num  0.32 0.14 1.23 0.63 0.63 1.85 1.92 1.88 0.61 0.19 ...
$ A.6 : num  0 0.28 0.19 0 0 0 0 0 0 0.32 ...
$ A.7 : num  0 0.21 0.19 0.31 0.31 0 0 0 0.3 0.38 ...
$ A.8 : num  0 0.07 0.12 0.63 0.63 1.85 0 1.88 0 0 ...
$ A.9 : num  0 0 0.64 0.31 0.31 0 0 0 0.92 0.06 ...
$ A.10: num  0 0.94 0.25 0.63 0.63 0 0.64 0 0.76 0 ...
$ A.11: num  0 0.21 0.38 0.31 0.31 0 0.96 0 0.76 0 ...
$ A.12: num  0.64 0.79 0.45 0.31 0.31 0 1.28 0 0.92 0.64 ...
$ A.13: num  0 0.65 0.12 0.31 0.31 0 0 0 0 0.25 ...
$ A.14: num  0 0.21 0 0 0 0 0 0 0 0 ...
$ A.15: num  0 0.14 1.75 0 0 0 0 0 0 0.12 ...
$ A.16: num  0.32 0.14 0.06 0.31 0.31 0 0.96 0 0 0 ...
$ A.17: num  0 0.07 0.06 0 0 0 0 0 0 0 ...
$ A.18: num  1.29 0.28 1.03 0 0 0 0.32 0 0.15 0.12 ...
$ A.19: num  1.93 3.47 1.36 3.18 3.18 0 3.85 0 1.23 1.67 ...
$ A.20: num  0 0 0.32 0 0 0 0 0 3.53 0.06 ...
$ A.21: num  0.96 1.59 0.51 0.31 0.31 0 0.64 0 2 0.71 ...
$ A.22: num  0 0 0 0 0 0 0 0 0 0 ...
$ A.23: num  0 0.43 1.16 0 0 0 0 0 0 0.19 ...
$ A.24: num  0 0.43 0.06 0 0 0 0 0 0.15 0 ...
$ A.25: num  0 0 0 0 0 0 0 0 0 0 ...
$ A.26: num  0 0 0 0 0 0 0 0 0 0 ...
$ A.27: num  0 0 0 0 0 0 0 0 0 0 ...
$ A.28: num  0 0 0 0 0 0 0 0 0 0 ...
$ A.29: num  0 0 0 0 0 0 0 0 0 0 ...
$ A.30: num  0 0 0 0 0 0 0 0 0 0 ...
$ A.31: num  0 0 0 0 0 0 0 0 0 0 ...
$ A.32: num  0 0 0 0 0 0 0 0 0 0 ...
$ A.33: num  0 0 0 0 0 0 0 0 0.15 0 ...
$ A.34: num  0 0 0 0 0 0 0 0 0 0 ...
$ A.35: num  0 0 0 0 0 0 0 0 0 0 ...
$ A.36: num  0 0 0 0 0 0 0 0 0 0 ...
$ A.37: num  0 0.07 0 0 0 0 0 0 0 0 ...
$ A.38: num  0 0 0 0 0 0 0 0 0 0 ...
$ A.39: num  0 0 0 0 0 0 0 0 0 0 ...
$ A.40: num  0 0 0.06 0 0 0 0 0 0 0 ...
$ A.41: num  0 0 0 0 0 0 0 0 0 0 ...
$ A.42: num  0 0 0 0 0 0 0 0 0 0 ...
$ A.43: num  0 0 0.12 0 0 0 0 0 0.3 0 ...
$ A.44: num  0 0 0 0 0 0 0 0 0 0.06 ...
$ A.45: num  0 0 0.06 0 0 0 0 0 0 0 ...
$ A.46: num  0 0 0.06 0 0 0 0 0 0 0 ...
$ A.47: num  0 0 0 0 0 0 0 0 0 0 ...
$ A.48: num  0 0 0 0 0 0 0 0 0 0 ...
$ A.49: num  0 0 0.01 0 0 0 0 0 0 0.04 ...
```

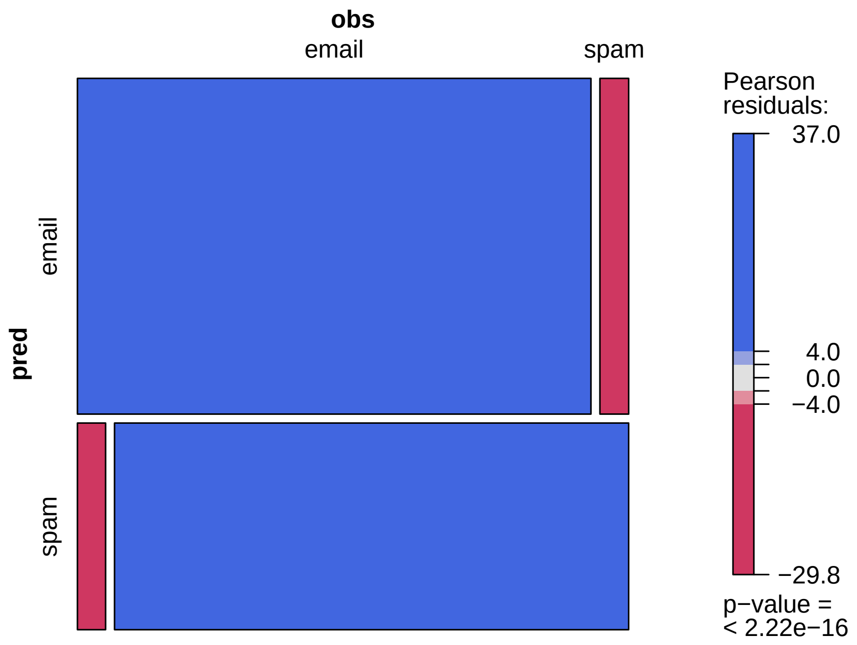

Figure 3: Predictions v.s. Observations: A Mosaicplot

```
$ A.50: num  0 0.132 0.143 0.137 0.135 0.223 0.054 0.206 ...
$ A.51: num  0 0 0 0 0 0 0 0 0 0 ...
$ A.52: num  0.778 0.372 0.276 0.137 0.135 0 0.164 0 0.181 ...
$ A.53: num  0 0.18 0.184 0 0 0 0.054 0 0.203 0.081 ...
$ A.54: num  0 0.048 0.01 0 0 0 0 0 0.022 0 ...
$ A.55: num  3.76 5.11 9.82 3.54 3.54 ...
$ A.56: int  61 101 485 40 40 15 4 11 445 43 ...
$ A.57: int  278 1028 2259 191 191 54 112 49 1257 749 ...
$ spam: Factor w/ 2 levels "email","spam": 2 2 2 2 2 2 2 2 2 2 ...
```

As for the function *svm()*, we only need to pay attention to two parameters, i.e. `cost` and `gamma`, which correspond to C and γ in equation (3), (4) and (5) respectively. If we want to perform a k-fold cross-validation to assess the quality of the model, we may as well specify the parameter `cross` as k.

After a SVM model is constructed, we should check the quality first, and the conventional way may be a contingency table with original classes and predicted ones – this is the case when cross-validation is not performed, otherwise we can assess the quality by the output of a k-fold cross-validation.

Here are some very ordinary codes for SVM modeling:

```
> if (require(ElemStatLearn) & require(e1071)) {
>     data(spam)
>     model = svm(spam ~ ., data = spam)
>     summary(model)

Call:
svm(formula = spam ~ ., data = spam)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.01754386

Number of Support Vectors:  1275

 ( 595 680 )


Number of Classes:  2

Levels:
 email spam

>     pred = fitted(model)
>     obs = spam$spam
>     table(pred, obs)

       obs
pred    email spam
  email  2697  151
  spam     91 1662

>     model = svm(spam ~ ., data = spam, cross = 10)
>     summary(model)

Call:
svm(formula = spam ~ ., data = spam, cross = 10)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.01754386
Number of Support Vectors:  1275

 ( 595 680 )


Number of Classes:  2

Levels:
 email spam

10-fold cross-validation on training data:

Total Accuracy: 93.08846
Single Accuracies:
 94.13043 95       91.95652 93.47826 92.6087 93.26087
 90       92.82609 96.30435 91.32321

>     library(vcd)
>     mosaic(table(pred, obs), shade = TRUE)
}
```

From the results above, we can know that the approximate accuracy[5] is $(2697+1662)/4601 \approx 94.74\%$ when no cross-validation is performed, and 93.09% with a 10-fold cross-validation. Here we only used the RBF kernel, and in section 4 we'll check the results by other kernels.

4 Comparisons

This section mainly discusses different performances of SVM, logistic regression and KNN in classification problems. Restricted by the relatively poor computing ability of our computers, we finally decide to select only a subset of variables as demonstrations to finish the comparisons.

As for the rule of variable selection, we may use AIC or C_p-statistic, etc, but here we just choose two most significant variables *by hand*. This seems to be not quite reasonable from the statistical view, but is fairly convenient for presentation.

4.1 Logistic Regression

As we know, logistic regression is just a special case of generalized linear model: we just have to specify a *binomial* family and a *logit* link function. So it's quite

[5]Figure 3 is a visual display for the accuracy of predictions

easy to perform this regression in R. Let's check the result of logistic regression first:

```
> m = glm(spam ~., data = spam, family=binomial)
> summary(m)

Call:
glm(formula = spam ~ ., family = binomial, data = spam)

Deviance Residuals:
       Min          1Q      Median          3Q         Max
-4.127e+00  -2.030e-01  -1.967e-06   1.140e-01   5.364e+00

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.569e+00  1.420e-01 -11.044  < 2e-16 ***
A.1         -3.895e-01  2.315e-01  -1.683 0.092388 .
A.2         -1.458e-01  6.928e-02  -2.104 0.035362 *
A.3          1.141e-01  1.103e-01   1.035 0.300759
A.4          2.252e+00  1.507e+00   1.494 0.135168
A.5          5.624e-01  1.018e-01   5.524 3.31e-08 ***
A.6          8.830e-01  2.498e-01   3.534 0.000409 ***
A.7          2.279e+00  3.328e-01   6.846 7.57e-12 ***
A.8          5.696e-01  1.682e-01   3.387 0.000707 ***
A.9          7.343e-01  2.849e-01   2.577 0.009958 **
A.10         1.275e-01  7.262e-02   1.755 0.079230 .
A.11        -2.557e-01  2.979e-01  -0.858 0.390655
A.12        -1.383e-01  7.405e-02  -1.868 0.061773 .
A.13        -7.961e-02  2.303e-01  -0.346 0.729557
A.14         1.447e-01  1.364e-01   1.061 0.288855
A.15         1.236e+00  7.254e-01   1.704 0.088370 .
A.16         1.039e+00  1.457e-01   7.128 1.01e-12 ***
A.17         9.599e-01  2.251e-01   4.264 2.01e-05 ***
A.18         1.203e-01  1.172e-01   1.027 0.304533
A.19         8.131e-02  3.505e-02   2.320 0.020334 *
A.20         1.047e+00  5.383e-01   1.946 0.051675 .
A.21         2.419e-01  5.243e-02   4.615 3.94e-06 ***
A.22         2.013e-01  1.627e-01   1.238 0.215838
A.23         2.245e+00  4.714e-01   4.762 1.91e-06 ***
A.24         4.264e-01  1.621e-01   2.630 0.008535 **
A.25        -1.920e+00  3.128e-01  -6.139 8.31e-10 ***
A.26        -1.040e+00  4.396e-01  -2.366 0.017966 *
A.27        -1.177e+01  2.113e+00  -5.569 2.57e-08 ***
A.28         4.454e-01  1.991e-01   2.237 0.025255 *
A.29        -2.486e+00  1.502e+00  -1.656 0.097744 .
A.30        -3.299e-01  3.137e-01  -1.052 0.292972
```

```
A.31        -1.702e-01  4.815e-01  -0.353 0.723742
A.32         2.549e+00  3.283e+00   0.776 0.437566
A.33        -7.383e-01  3.117e-01  -2.369 0.017842 *
A.34         6.679e-01  1.601e+00   0.417 0.676490
A.35        -2.055e+00  7.883e-01  -2.607 0.009124 **
A.36         9.237e-01  3.091e-01   2.989 0.002803 **
A.37         4.651e-02  1.754e-01   0.265 0.790819
A.38        -5.968e-01  4.232e-01  -1.410 0.158473
A.39        -8.650e-01  3.828e-01  -2.260 0.023844 *
A.40        -3.046e-01  3.636e-01  -0.838 0.402215
A.41        -4.505e+01  2.660e+01  -1.694 0.090333 .
A.42        -2.689e+00  8.384e-01  -3.207 0.001342 **
A.43        -1.247e+00  8.064e-01  -1.547 0.121978
A.44        -1.573e+00  5.292e-01  -2.973 0.002953 **
A.45        -7.923e-01  1.556e-01  -5.091 3.56e-07 ***
A.46        -1.459e+00  2.686e-01  -5.434 5.52e-08 ***
A.47        -2.326e+00  1.659e+00  -1.402 0.160958
A.48        -4.016e+00  1.611e+00  -2.493 0.012672 *
A.49        -1.291e+00  4.422e-01  -2.920 0.003503 **
A.50        -1.881e-01  2.494e-01  -0.754 0.450663
A.51        -6.574e-01  8.383e-01  -0.784 0.432914
A.52         3.472e-01  8.926e-02   3.890 0.000100 ***
A.53         5.336e+00  7.064e-01   7.553 4.24e-14 ***
A.54         2.403e+00  1.113e+00   2.159 0.030883 *
A.55         1.199e-02  1.884e-02   0.636 0.524509
A.56         9.118e-03  2.521e-03   3.618 0.000297 ***
A.57         8.437e-04  2.251e-04   3.747 0.000179 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 6170.2  on 4600  degrees of freedom
Residual deviance: 1815.8  on 4543  degrees of freedom
AIC: 1931.8

Number of Fisher Scoring iterations: 13
```

Obviously, there're many variables that are not significant, and now we just select Next we select `A.53` (the frequency of the character "$") and `A.16` (the frequency of the word "free") for further research. Certainly we may use the function *step()* to perform a stepwise logistic regression, but it's too time-consuming for our computers.

Figure 4 shows how the email types depend on two variables, namely the frequency of the word "free" and the character "$". It's not difficult to find out that normal emails almost contain neither "free" nor "$".

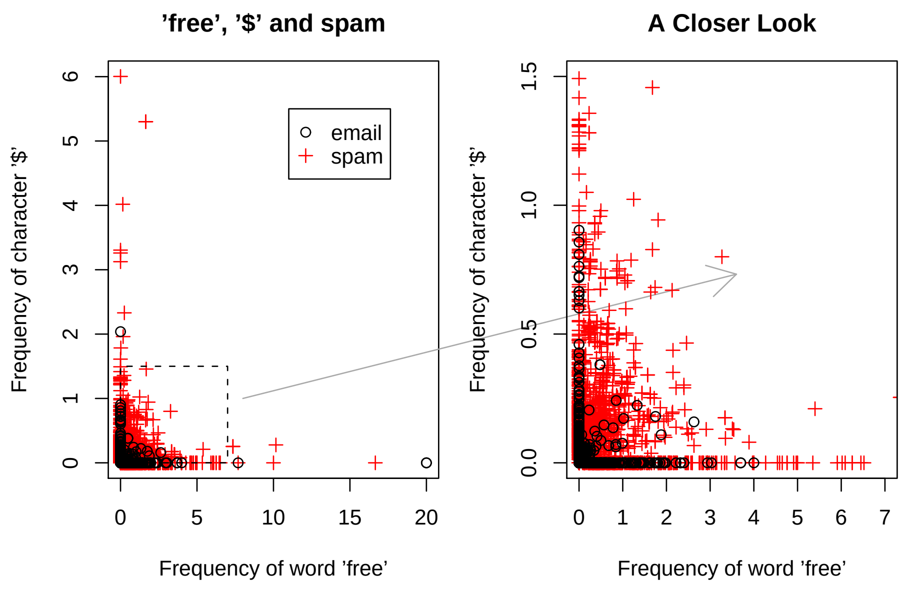

Figure 4: Relationships between the frequencies of “free” & “$” and spam emails

Again we fit a logistic regression model but only use `A.53` and `A.16`.

```
> m2 = glm(spam ~ A.16 + A.53, data = spam, family = binomial,
+           control = glm.control(maxit = 50))
> summary(m2)

Call:
glm(formula = spam ~ A.16 + A.53, family = binomial, data = spam,
    control = glm.control(maxit = 50))

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-7.5403  -0.6732  -0.6732   0.6202   1.7865

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.36921    0.04368  -31.35   <2e-16 ***
A.16         1.48985    0.09699   15.36   <2e-16 ***
A.53        13.88779    0.62472   22.23   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)
```

```
    Null deviance: 6170.2  on 4600  degrees of freedom
Residual deviance: 4425.5  on 4598  degrees of freedom
AIC: 4431.5

Number of Fisher Scoring iterations: 33
```

Please note that why we use the parameter control here.

If we choose 0.5 as a cut point, the accuracy of classification can be computed as follows:

```
table(predict(m2, type = "response") > 0.5, spam$spam)

        email spam
  FALSE  2640  703
  TRUE    148 1110
```

The proportion of emails being correctly classified is $(2640 + 1110)/4601 \approx 81.50\%$, and now we may as well check the performance of SVM again:

```
> model1 = svm(spam ~ A.16 + A.53, data = spam)
> pred = fitted(model1)
> obs = spam$spam
> table(pred, obs)
       obs
pred    email spam
  email  2507  467
  spam    281 1346
```

The accuracy is $(2507 + 1346)/4601 \approx 83.74\%$, which seems to be better than logistic regression. When we perform cross-validations, the accuracy rate of SVM is still higher.

```
> tr = sample(4601, 2300)
> m = glm(spam ~ A.16 + A.53, data = spam[tr, ], family = binomial,
    control = glm.control(maxit = 50))
> (r = table(predict(m, newdata = spam[-tr, ], type = "response") >
    0.5, (spam[-tr, ])$spam))

        email spam
  FALSE  1306  359
  TRUE     74  562

> model = svm(spam ~ A.16 + A.53, data = spam[tr, ])
> pred = predict(model, spam[-tr, ])
> obs = spam[-tr, ]$spam
> (res = table(pred, obs))

       obs
pred    email spam
  email  1252  255
  spam    128  666

> print((r[1] + r[4])/sum(r))
[1] 0.811821
> print((res[1] + res[4])/sum(res))
[1] 0.8335506
```

4.2 *k*-Nearest Neighbour (KNN) Classification

The function for *k*-Nearest Neighbour Classification is *knn()* in the package class. The basic usage is:

```
knn(train, test, cl, k = 1, l = 0, prob = FALSE, use.all = TRUE)
```

Here we just use the same training set cases with the former comparison and set the number of neighbours considered as 1, which is the default value of the parameter k.

```
> library(class)
> k = knn(spam[tr, -58], spam[-tr, -58], spam$spam[tr])
> res = table(k, spam$spam[-tr])

        email spam
  email  1160  224
  spam    229  688

> print((res[1] + res[4])/sum(res))
[1] 0.803129
```

The accuracy rate is 80.3% – obviously it's worse than the performance in either logistic regression or SVM. However, please note that we didn't consider other values of the parameter k, so the conclusion is not that definite.

Judging from the above two cases of comparison, it seems that SVMs outgo the other two methods in the classification of spam emails.

5 Parameter Selection

In former texts we didn't mention the problem of parameter selection for SVMs, and in fact it is also an important aspect for the applications. If we choose the

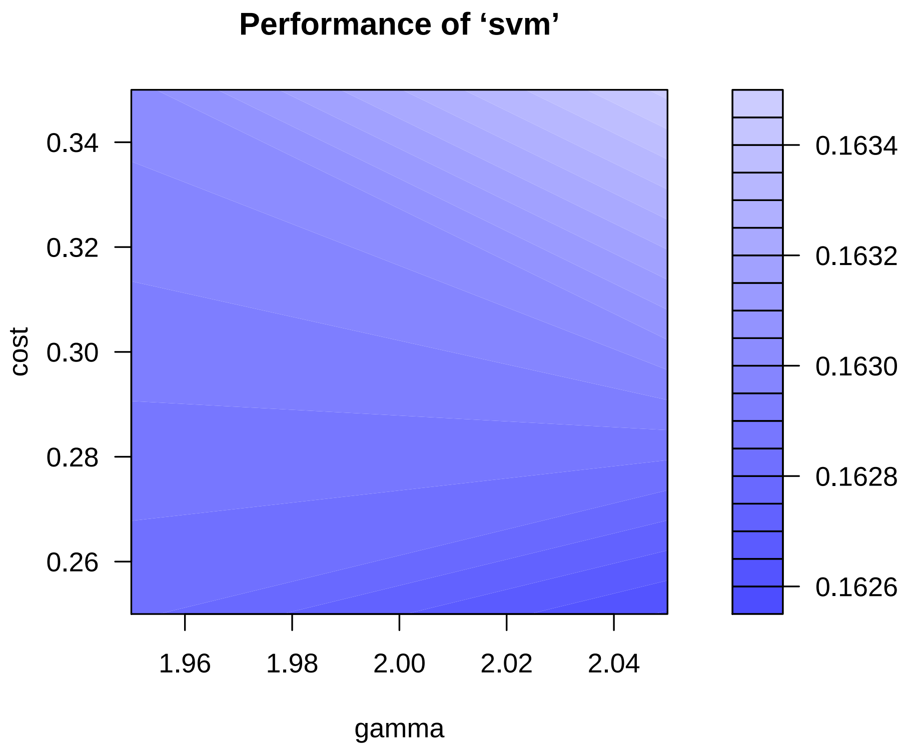

Figure 5: Parameter selection using grid search

RBF kernel, we'll only have two parameters which can be changed by hand: gamma and cost. Actually there's a good method for selecting proper values of them, which is called "grid search ", i.e. to search for the values of certain parameters over *supplied parameter ranges*. Such work can be done through the function *tune()* in the package e1071. Here is the usage:

```
tune(method, train.x, train.y = NULL, data = list(), validation.x =
     NULL, validation.y = NULL, ranges = NULL, predict.func = predict,
     tunecontrol = tune.control(), ...)
```

Since we have just two uncertain parameters, this search can be visualized in a 2-D graph just like Figure 5. The depth of the blue color in Figure 5 stands for the prediction error with corresponding parameter values. It's not difficult to find out that the (probably) most optimal values for gamma and cost lies in the right-bottom corner, and the numbers are listed in the following codes:

```
> obj = tune(svm, spam ~ A.16 + A.53, data = spam,
      ranges = list(gamma = c(1.95, 2.05), cost = c(0.25, 0.35)))
> summary(obj)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 gamma cost
  2.05 0.25

- best performance: 0.1625942

- Detailed performance results:
  gamma cost     error
1  1.95 0.25 0.1628111
2  2.05 0.25 0.1625942
3  1.95 0.35 0.1630299
4  2.05 0.35 0.1634657
> plot(obj)
```

A Variable names in spam data

All the variable names are listed as follows (in the same order as variable labels A.1, A.2, ...):

```
word_freq_make; word_freq_address; word_freq_all;
word_freq_3d; word_freq_our; word_freq_over;
word_freq_remove; word_freq_internet; word_freq_order;
word_freq_mail; word_freq_receive; word_freq_will;
word_freq_people; word_freq_report; word_freq_addresses;
word_freq_free; word_freq_business; word_freq_email;
word_freq_you; word_freq_credit; word_freq_your;
word_freq_font; word_freq_000; word_freq_money;
word_freq_hp; word_freq_hpl; word_freq_george;
word_freq_650; word_freq_lab; word_freq_labs;
word_freq_telnet;word_freq_857; word_freq_data;
word_freq_415; word_freq_85; word_freq_technology;
word_freq_1999; word_freq_parts;word_freq_pm;
word_freq_direct; word_freq_cs; word_freq_meeting;
word_freq_original; word_freq_project;
word_freq_re; word_freq_edu; word_freq_table;
word_freq_conference;

char_freq_; char_freq_(; char_freq_[;
char_freq_!; char_freq_$; char_freq_#;

capital_run_length_average;
capital_run_length_longest;
capital_run_length_total;
```

References

[1] Hastie, T., R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction.* Springer, 2001.

[2] Chih-Chung Chang and Chih-Jen Lin, *LIBSVM : a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`

[3] Newman, D.J. & Hettich, S. & Blake, C.L. & Merz, C.J. (1998). UCI Repository of machine learning databases `http://www.ics.uci.edu/~mlearn/MLRepository.html`. Irvine, CA: University of California, Department of Information and Computer Science.

[4] Cranor, Lorrie F., LaMacchia, Brian A. *Spam!* Communications of the ACM, 41(8):74-83, 1998