Project3

Jung-Han Wang

Monday, November 17, 2014

Project 3

Problem 1

```
mydat <- read.table('/home/robert/cloud/Classes/STA6106 Stat Computing/Project2/Project3/training datas
my_matrix <- as.matrix(mydat)
```

This problem is to get some codes to perform the support vector data description (SVDD)

a. Write an *R* function to perform the SVDD.

First, we want to compute the kernel matrix

$$\begin{bmatrix} k(x_1,x_1) & k(x_1,x_2) & ... & k(x_1,x_n) \\ k(x_2,x_1) & k(x_2,x_2) & ... & k(x_2,x_n) \\ ... & ... & ... & ... \\ k(x_n,x_1) & k(x_n,x_2) & ... & k(x_n,x_n) \end{bmatrix}$$

To do this, we must define a kernel function. This function essentially calculates the distance between each pair of data vectors. For simplicity, we begin by using the simplest distance, the euclidean distance. The euclidean distance between two data vectors is just their dot product. The `kernlab` package includes a function `vanilladot()` that when called, creates another function that will compute these dot products.

```
my_kernel <- vanilladot()
```

We have now created a function `my_kernel()` that will calculate the linear distance between two data vectors for us. We check that this is equivalent to the dot product.

```
my_kernel(my_matrix[1, ], my_matrix[2, ]) # dot prod using kernel function
```

```
      [,1]
[1,] 37.49
```

```
crossprod(my_matrix[1, ], my_matrix[2, ]) # dot prod using base R function
```

```
      [,1]
[1,] 37.49
```

```
my_matrix[1, ] %*% my_matrix[2, ] # old school matrix multiplication operator
```

```
      [,1]
[1,] 37.49
```

Now that we have defined a function for applying our kernel function to a pair of data vectors, we can easily create a kernel matrix from our data matrix. There is a handy function in the `kernlab` package called `kernelMatrix()` that does exactly this. It requires as arguments `kernel`, the kernel function to be used, and `x`, the data matrix from which to compute the kernel matrix. We pass the function our kernel function `my_kernel()` and our data matrix `my_matrix`. The function returns a nxn (66x66) matrix of class `kernelMatrix`.

```
H <- kernelMatrix(kernel=my_kernel, x=my_matrix)
dim(H)
```

```
[1] 66 66
```

```
class(H)
```

```
[1] "kernelMatrix"
attr(,"package")
[1] "kernlab"
```

The SVDD problem can be stated mathematically as

$$\underset{\alpha}{max} \sum_i \alpha_i \; k(x_i, x_i) - \sum_{i,j} \alpha_i \alpha_j \; k(x_i, x_j)$$

subject to $\alpha_i \geq 0$ and $\sum \alpha_i = 1$.

The quadratic solver in the `kernlab` package solves quadratic programming problems in the form

$$min(c'x + \frac{1}{2}x'Hx)$$

subject to $b \leq Ax \leq b + r$ and $l \leq x \leq u$.

To re-state the SVDD problem in the form required by the quadratic solver we set

$$x' = [\alpha_1, \alpha_2, ..., \alpha_n]$$

$$H = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & ... & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & ... & k(x_2, x_n) \\ ... & ... & ... & ... \\ k(x_n, x_1) & k(x_n, x_2) & ... & k(x_n, x_n) \end{bmatrix}$$

$$c' = [k(x_1, x_1), k(x_2, x_2), ..., k(x_n, x_n)] = diag(H)$$

then

$$c'x + \frac{1}{2}x'Hx = [k(x_1, x_1), ..., k(x_n, x_n)] \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ ... \\ \alpha_n \end{bmatrix} + \frac{1}{2}(2)[\alpha_1, ..., \alpha_n] \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & ... & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & ... & k(x_2, x_n) \\ ... & ... & ... & ... \\ k(x_n, x_1) & k(x_n, x_2) & ... & k(x_n, x_n) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ ... \\ \alpha_n \end{bmatrix}$$

To re-state the constraints of the SVDD problem in the form required by the quadratic solver, we set

$$b = 1,\ A = [1, 1, ..., 1],\ x = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ ... \\ \alpha_n \end{bmatrix},\ r = 0$$

$$l = \begin{bmatrix} 0 \\ 0 \\ ... \\ 0 \end{bmatrix}, x = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ ... \\ \alpha_n \end{bmatrix}, u = \begin{bmatrix} \infty \\ \infty \\ ... \\ \infty \end{bmatrix}$$

then $b \leq Ax \leq b + r$ is equivalent to

$$1 \leq [1, 1, ..., 1] \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ ... \\ \alpha_n \end{bmatrix} \leq 1 + 0$$

or

$$1 \leq \sum \alpha_i \leq 1$$

and $l \leq x \leq u$ is equivalent to $0 \leq \alpha_i \leq \infty$.

b. Write an *R* function to perform the prediction of a new observation using SVDD.

c. Write an *R* function for detecting potential outliers for a new set of observations, along with the upper threshold.

Problem 2

The goal of problem 2 is to perform the support vector data description (SVDD) using the Mahalanobis kernel function. We will simplify the problem by using the identity function for g.

(a) Write an *R* function to compute the Mahalanobis kernel distance $\boldsymbol{d_g(x)}$

(b) Write an *R* function to perform the Mahalanobis kernel SVDD.

(c) Write an *R* function to perform the prediction of a new observation using the Mahalanobis kernel SVDD.

(d) Write an *R* function for detecting potential outliers for a new set of observations, along with the upper threshold.