Chapter 9
Support Vector Machine

Abstract Support Vector Machine is one of the classical machine learning techniques that can still help solve big data classification problems. Especially, it can help the multidomain applications in a big data environment. However, the support vector machine is mathematically complex and computationally expensive. The main objective of this chapter is to simplify this approach using process diagrams and data flow diagrams to help readers understand theory and implement it successfully. To achieve this objective, the chapter is divided into three parts: (1) modeling of a linear support vector machine; (2) modeling of a nonlinear support vector machine; and (3) Lagrangian support vector machine algorithm and its implementations. The Lagrangian support vector machine with simple examples is also implemented using the R programming platform on Hadoop and non-Hadoop systems.

9.1 Linear Support Vector Machine

Support vector machine [1], as mentioned in Chap. 6, provides a classification learning model and an algorithm rather than a regression model and an algorithm. It uses the simple mathematical model $\mathbf{y} = \mathbf{w}\mathbf{x}' + \gamma$, and manipulates it to allow linear domain division. The support vector machine can be divided into linear and nonlinear models [2]. It is called linear support vector machine if the data domain can be divided linearly (e.g., straight line or hyperplane) to separate the classes in the original domain. If the data domain cannot be divided linearly, and if it can be transformed to a space called the feature space where the data domain can be divided linearly to separate the classes, then it is called nonlinear support vector machine.

Therefore, the steps in the linear support vector machine are: the mapping of the data domain into a response set and the dividing of the data domain. The steps in the nonlinear support vector machines are: the mapping of the data domain to a feature space using a kernel function [3], the mapping of the feature space domain into

S. Suthaharan, *Machine Learning Models and Algorithms for Big Data Classification*, Integrated Series in Information Systems 36,
DOI 10.1007/978-1-4899-7641-3_9

the response set, and then the dividing of the data domain. Hence, mathematically, we can say that the modeling of a linear support vector machine adopts the linear equation $\mathbf{y} = \mathbf{w}\mathbf{x}' + \gamma$, and the modeling of a nonlinear support vector machine adopts the nonlinear equation $\mathbf{y} = \mathbf{w}\phi(\mathbf{x}') + \gamma$. The kernel function makes it nonlinear. The classification technique using a support vector machine includes the parametrization and the optimization objectives. These objectives mainly depend on the topological class structure on the data domain. That is, the classes may be linearly separable or linearly nonseparable. However, linearly separable classes may be nonlinearly separable. Therefore, the parametrization and optimization objectives that focus on the data domain must take these class properties into consideration.

9.1.1 Linear Classifier: Separable Linearly

This section mainly focuses on the two-class classification problem [4] using the support vector machine. However, a multiclass support vector machine can easily be derived from a combination of two-class support vector machines by integrating an ensemble approach [5]. This chapter focuses only on the two-class classification using the support vector machine learning models. Let us first consider the linear case. In Chap. 7, some preliminaries for the support vector machine were discussed, and a straight line equation was derived as:

$$\mathbf{w}\mathbf{x}' + \gamma = 0 \tag{9.1}$$

Considering a data domain, this parameterized straight line divides the data domain into two subdomains, and we may call them left subdomain and right subdomain (as we state in decision tree-based models), denote them by D_1 and D_2, and define them as follows:

$$\begin{aligned} D_1 &= \{\mathbf{x} : \mathbf{w}\mathbf{x}' + \gamma \leq 0\} \\ D_2 &= \{\mathbf{x} : \mathbf{w}\mathbf{x}' + \gamma > 0\} \end{aligned} \tag{9.2}$$

The points falling in these subdomains may be distinguished with labels 1 for the subdomain D_1 and -1 for the subdomain D_2. Therefore, the parametrization objective of the support vector machine can be defined as follows:

$$\begin{aligned} \mathbf{w}\mathbf{x}' + \gamma &= 1, \mathbf{x} \in D_1 \\ \mathbf{w}\mathbf{x}' + \gamma &= -1, \mathbf{x} \in D_2 \end{aligned} \tag{9.3}$$

In the parametrization objectives, we have modeled two straight lines (or hyperplanes) that can help to define boundaries between the classes. The optimization objective is to define an objective function (in this case, the distance between the straight lines) and search for the parameter values that maximize the distance. These lines are parallel to each other; therefore, we can simply use the standard distance formula between two parallel lines $y = mx + b_1$ and $y = mx + b_2$ as follows [6]:

$$d = \frac{(b_2 - b_1)}{\sqrt{m^2 + 1}} \tag{9.4}$$

where the slopes of the straight lines are $m = \mathbf{w}$, and their intercepts are $b_1 = \gamma + 1$ and $b_2 = \gamma - 1$. By substituting these variables, we can establish the following:

$$d = \frac{\pm 2}{\sqrt{\mathbf{w}\mathbf{w}' + 1}} \tag{9.5}$$

Ultimately, this distance formula will be the measure for the optimization problem that we build; therefore, without loss of generality, we can rewrite it as follows:

$$d = \frac{\pm 2}{\sqrt{\mathbf{w}\mathbf{w}'}} \tag{9.6}$$

In practice, the support vector machine optimization problem is written using the mathematical *norm* notation, therefore we rewrite the above equation as follows:

$$d = \frac{\pm 2}{||\mathbf{w}||^2} \tag{9.7}$$

By squaring both sides of the equation, and then dividing both sides of the equation by the value of 2, we can obtain the following simple mathematical relationship:

$$\frac{d^2}{2} = \frac{1}{\frac{||\mathbf{w}||^2}{2}} \tag{9.8}$$

It states that instead of maximizing the distance function $d^2/2$, we can minimize $||\mathbf{w}||^2/2$. In other words, we can minimize the prediction error with respect to the above classifier while maximizing the distance between them (this is the optimization objective). Therefore, the following mathematical expression can be defined for the prediction error between $\mathbf{x} \in D$ and its response variables y:

$$e = 1 - y(\mathbf{w}\mathbf{x}' + \gamma) \tag{9.9}$$

This error function plays a major role in the development of an optimization problem for the support vector machine. Let us now understand its role through the following thinking with examples.

Thinking with Example 9.1

Suppose the actual response y is -1, and the predicted response based on the classifier $\mathbf{w}\mathbf{x}' + \gamma = -1$ is -1, then we have $e = 1 - (-1)(-1) = 1 - 1 = 0$. Similarly, suppose the actual response y is 1, and the predicted response based on the classifier is $\mathbf{w}\mathbf{x}' + \gamma = 1$ is 1, then $e = 1 - (1)(1) = 1 - 1 = 0$. Therefore, it is clear the classification error is 0. However, if the actual response y is 1 and the predicted response

based on the classifier $\mathbf{wx}' + \gamma = -1$ is -1, then $e = 1 - (1)(-1) = 1 + 1 = 2$. This indicates the error in the predicted response. Similarly, if the actual response y is -1, and the predicted response based on the classifier $\mathbf{wx}' + \gamma = 1$ is 1, then $e = 1 - (-1)(1) = 1 + 1 = 2$—this also gives the error indicator 2.

Suppose the actual response y is -1.1, and the predicted response of the classifier $\mathbf{wx}' + \gamma = -1$ is -1, then what is the value of e? Well! $e = 1 - (-1.1)(-1) = 1 - 1.1 = -0.1$. This means the variable x that corresponds to the response $y = -1.1$ is on the correct side of the classifier. Now suppose the actual response y is -0.9, and the response of the classifier $\mathbf{wx}' + \gamma = -1$, then what is the value of e? The answer is: $e = 1 - (-0.9)(-1) = 1 - 0.9 = 0.1$. It means the variable x that corresponds to $y = -1.1$ is on the wrong side of the classifier. Therefore, we can conclude that the negative error e is preferred when we optimize the classification.

9.1.1.1 The Learning Model

These examples show that the parameters $\mathbf{w}$ and γ must be selected such that the error $e \leq 0$. This leads to the following inequality:

$$1 - y(\mathbf{wx}' + \gamma) \leq 0. \tag{9.10}$$

$$y(\mathbf{wx}' + \gamma) \geq 1. \tag{9.11}$$

By combining the minimization goals, we can create the following optimization problem, and it is the basis for the two-class support vector machine [7]:

$$\begin{aligned} \underset{\mathbf{w},\gamma}{\text{Minimize:}} \quad & \frac{||\mathbf{w}||^2}{2} \\ \text{subject to:} \quad & y(\mathbf{wx}' + \gamma) \geq 1 \end{aligned} \tag{9.12}$$

We can now extend this optimization problem to a multidimensional data domain with a complete matrix representation as follows [8]:

$$\begin{aligned} \underset{\mathbf{w},\gamma}{\text{Minimize:}} \quad & \frac{||\mathbf{w}||^2}{2} \\ \text{subject to:} \quad & \mathbf{s}(\mathbf{wx}' + \gamma\mathbf{I}) \geq \mathbf{I} \end{aligned} \tag{9.13}$$

We can call the above "Minimize" term the *svm-measure* and the "subject to" term the *label error*. In this equation, $\mathbf{x}$ represents the matrix or the n points in the data domain D, and $\mathbf{s}$ is the set that represents the response variables of $\mathbf{x}$. The matrix I is the identity matrix, and γ is the intercept of the straight line (or the hyperplane). Three coding examples are designed to help you understand the svm-based optimization problem presented in Eq. (9.13). These examples are based on: (1) two points and single line svm-based domain division, (2) two points and three lines svm-based domain division, and (3) five points and three lines svm-based domain division, which will help you extend it to a generalized svm-based domain division.

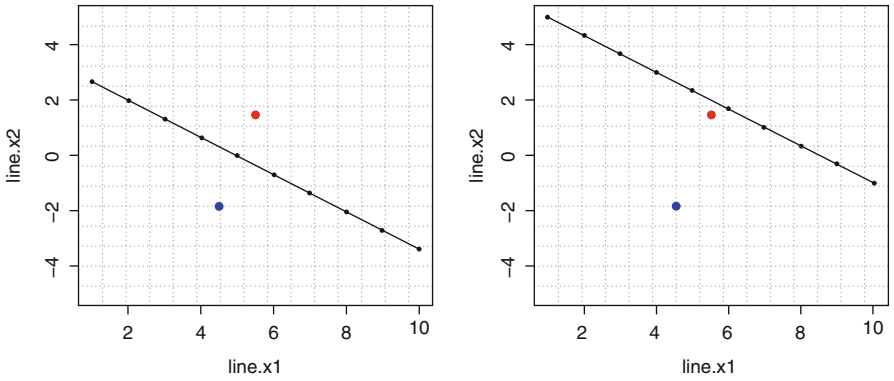

Fig. 9.1 The results of the "two point, straight line" coding example in Listing 9.1

9.1.1.2 A Coding Example: Two Points, Single Line

The main objective of this coding example is to illustrate the first iterative step of the svm-based optimization problem presented in Eq. (9.13). In Listing 9.1, a coding example is given to illustrate the problem of dividing the data domain linearly without applying any optimization mechanism. It is written in the R programming language, and it is expected that this example will help you build the concepts of the support vector machine. In this example, two-class points $\mathbf{x}_1 = (5.5, 1.5)$ and $\mathbf{x}_2 = (4.5, -1.8)$ are considered as illustrated in the first figure of Fig. 9.1 on a two-dimensional data domain. The goal is to find a straight line that separates the points.

Listing 9.1 An R programming example—a svm-based domain division

```
# Date: May 21st, 2015
# svm-prog1-new

# select weights for the straight line
weight.w = matrix(c(2,3),nrow=1,ncol=2,byrow=TRUE)
weight.w

# select intercept for the straight line
gamma.g = matrix(c(-10,-10),nrow=1,ncol=2,byrow=TRUE)
#gamma.g = matrix(c(-17,-17),nrow=1,ncol=2,byrow=TRUE)
gamma.g

# select points
point.x = matrix(c(5.5,1.5,4.5,-1.8),nrow=2,ncol=2,byrow=TRUE)
t(point.x)

# assign class labels
label.s = matrix(c(1,-1),nrow=1,ncol=2,byrow=TRUE)
label.s

# determine label error
```

```
22 hat.y = weight.w %*% t(point.x) + gamma.g
23 hat.y
24
25 # check for minimum error
26 label.s * hat.y
27
28 # calculate SVM measure
29 (weight.w %*% t(weight.w))/2
30
31 # display the straight line and points
32 line.x1 = rep(0,10)
33 line.x2 = rep(0,10)
34
35 slope = weight.w[1]/weight.w[2]
36 intercept = gamma.g[1]/weight.w[2]
37
38 for (i in 1:10) {
39   line.x1[i] = i
40   line.x2[i] = -intercept - slope * line.x1[i]
41 }
42
43 plot(line.x1, line.x2, type="o", pch=20, ylim=c(-5,5))
44 points(point.x[1,1], point.x[1,2], col="red", pch=19)
45 points(point.x[2,1], point.x[2,2], col="blue", pch=19)
46 grid(15, 15, lwd=2)
```

The block of code from line 4 to line 11 sets the parameters for a straight line, which could be the svm-based classifier. Two parameter values -10 and 17 are selected for the intercept parameter of the straight line. The codes in lines 14 and 18 select two points and assign labels, respectively. The code in line 26 helps us select the index of the minimum error and will be used for selecting the weights and intercept values as shown in the codes in lines 35 and 36. In line 22, the label error is determined based on the straight line defined earlier, and the svm-measure is calculated in line 29.

The slope and the intercept are calculated as shown in lines 35 and 36. The block of code in lines 38–46 produced the figures in Fig. 9.1. The first figure is related to the intercept value selected according to line 9, and the second figure is related to line 10 when uncommented. The program statements are written sequentially to help you understand the mathematical processes for the support vector machine presented at the beginning of this chapter. Comparing the two choices of the parameters, we can see the first set of parameters provides a better domain division than the second set for svm-based classification. It also illustrates the effect of the intercept parameter.

9.1.1.3 A Coding Example: Two Points, Three Lines

The main objective of the coding example in Listing 9.2 is to illustrate the iterative steps that lead to an optimization in the svm-based classification problem which is

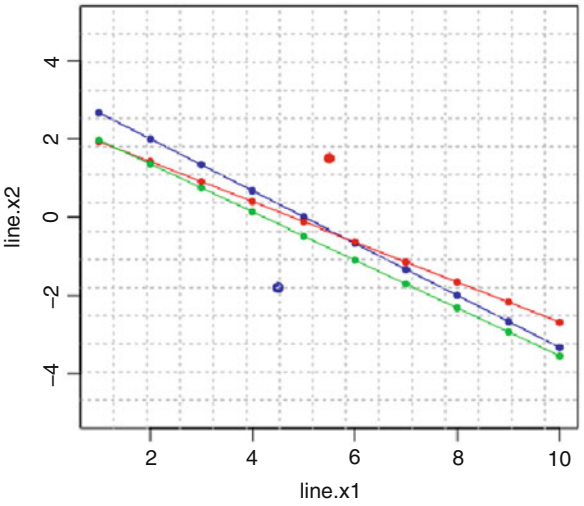

Fig. 9.2 A possible classifier for two points

presented in Eq. (9.13). However, the iterative steps are shown sequentially in the program, so that you can understand the algorithm better. As an exercise, once you understand the algorithm, make the program efficient using loops and functions.

Listing 9.2 An R programming example—the svm-based optimization problem

```
# Date: May 21st, 2015
# svm-prog1-new

# ITERATION 1 ###################

# select weights for parametrization
weight.w = matrix(c(2,3),nrow=1,ncol=2,byrow=TRUE)
weight.w

# select intercept for parametrization
gamma.g = matrix(c(-10,-10),nrow=1,ncol=2,byrow=TRUE)
gamma.g

# select points
point.x = matrix(c(5.5,1.5,4.5,-1.8),nrow=2,ncol=2,byrow=TRUE)
t(point.x)

# assign class labels
label.s = matrix(c(1,-1),nrow=1,ncol=2,byrow=TRUE)
label.s

# determine label error
hat.y = weight.w %*% t(point.x) + gamma.g
hat.y

```

```
# check for minimum error
label.s * hat.y

# calculate SVM measure for optimization
measure.one = (weight.w %*% t(weight.w))/2

# display the straight line and points
line.x1 = rep(0,10)
line.x2 = rep(0,10)

slope = weight.w[,1]/weight.w[,2]
intercept=gamma.g/weight.w[,2]

for (i in 1:10) {
  line.x1[i] = i
  line.x2[i] = -intercept - slope*line.x1[i]
}

plot(line.x1, line.x2, type="o", col="blue", pch=20, ylim=c(-5,5)
    )
points(point.x[1,1], point.x[1,2], col="red", pch=19)
points(point.x[2,1], point.x[2,2], col="blue", pch=19)
grid(15, 15, lwd=2)

# ITERATION 2 ###################

# select weights for parametrization
weight.w = matrix(c(2.1,4.1),nrow=1,ncol=2,byrow=TRUE)
weight.w

# select intercept for parametrization
gamma.g = matrix(c(-10,-10),nrow=1,ncol=2,byrow=TRUE)
#gamma.g = matrix(c(-17,-17),nrow=1,ncol=2,byrow=TRUE)
gamma.g

# select points
point.x = matrix(c(5.5,1.5,4.5,-1.8),nrow=2,ncol=2,byrow=TRUE)
t(point.x)

# assign class labels
label.s = matrix(c(1,-1),nrow=1,ncol=2,byrow=TRUE)
label.s

# determine label error
hat.y = weight.w %*% t(point.x) + gamma.g
hat.y

# check for minimum error
label.s * hat.y

# calculate SVM measure for optimization
measure.two = (weight.w %*% t(weight.w))/2

# display the straight line and points
```

```
line.x1 = rep(0,10)
line.x2 = rep(0,10)

slope = weight.w[,1]/weight.w[,2]
intercept=gamma.g/weight.w[,2]

for (i in 1:10) {
  line.x1[i] = i
  line.x2[i] = -intercept - slope*line.x1[i]
}

lines(line.x1, line.x2, type="o", col="red", pch=20, ylim=c(-5,5)
    )

# ITERATION 3 ##################

# select weights for parametrization
weight.w = matrix(c(1.9,3.1),nrow=1,ncol=2,byrow=TRUE)
weight.w

# select intercept for parametrization
gamma.g = matrix(c(-8,-8),nrow=1,ncol=2,byrow=TRUE)
#gamma.g = matrix(c(-17,-17),nrow=1,ncol=2,byrow=TRUE)
gamma.g

# select points
point.x = matrix(c(5.5,1.5,4.5,-1.8),nrow=2,ncol=2,byrow=TRUE)
t(point.x)

# assign class labels
label.s = matrix(c(1,-1),nrow=1,ncol=2,byrow=TRUE)
label.s

# determine label error
hat.y = weight.w %*% t(point.x) + gamma.g
hat.y

# check for minimum error
label.s * hat.y

# calculate SVM measure for optimization
measure.three = (weight.w %*% t(weight.w))/2

# display the straight line and points
line.x1 = rep(0,10)
line.x2 = rep(0,10)

slope = weight.w[,1]/weight.w[,2]
intercept=gamma.g/weight.w[,2]

for (i in 1:10) {
  line.x1[i] = i
  line.x2[i] = -intercept - slope*line.x1[i]
}
```

```
132
133 lines(line.x1, line.x2, type="o", col="green", pch=20, ylim=c
        (-5,5))
134
135 # optimization
136 measure.one
137 measure.two
138 measure.three
```

The first iteration with the first set of weights is presented in the block of code from line 6 to line 47, and it reflects the code presented in Listing 9.1. Similarly, with the other sets of weights, the iterations 2 and 3 are presented in the blocks of code from line 51 to line 90 and from line 94 to line 133, respectively. In the block of code from line 136 to 138, the results of svm-measures are displayed, and we can then select the one with the smaller value for the best classifier. Thus this program calculates the label errors for the straight line equations $y = 2x_1 + 3x_2 - 10$, $y = 2.1x_1 + 4.1x_2 - 10$, and $y = 1.9x_1 + 3.1x_2 - 8$, and the svm-measures to determine the straight line that minimizes the svm-measure. It also produced the graph in Fig. 9.2, and we can see the three classifiers and the best among them.

9.1.1.4 A Coding Example: Five Points, Three Lines

The main objective of the coding example in Listing 9.3 is to generalize the iterative steps that lead to optimization in the svm-based optimization problem presented in Eq. (9.13) using matrix formulation. This example inputs a file "file3.txt," which contains the data points and class labels. The file contains three columns where the first two columns represent the two features, and the third column represents the class labels. It has two features f_1 and f_2 with values $f_1 = \{5.5, 5.7, 6.1, 4.5, 4.3\}$ and $f_2 = \{1.5, 0.5, 1.1, -1.8, -1.5\}$, and their corresponding label set $L = \{1, 1, 1, -1, -1\}$.

Listing 9.3 An R programming example—linear svm-based classifiers

```
1  # Date: May 21st, 2015
2  # svm-prog2-new
3
4  # read data from a file
5  data <- read.table("file3.txt", sep="")
6
7  # extract feature and separate the labels
8  point.x = matrix(c(data$V1, data$V2),nrow=2,ncol=5,byrow=TRUE)
9  point.x
10
11 # select weights for parametrization
12 weight.w=matrix(c(2,3,2.1,4.1,1.9,3.1),nrow=3,ncol=2,byrow=TRUE)
13 weight.w
14
15 # select intercepts for parametrization
16 gamma.g = matrix(c(-10,-10,-8),nrow=3,ncol=1,byrow=TRUE)
17 gamma.g
18
```

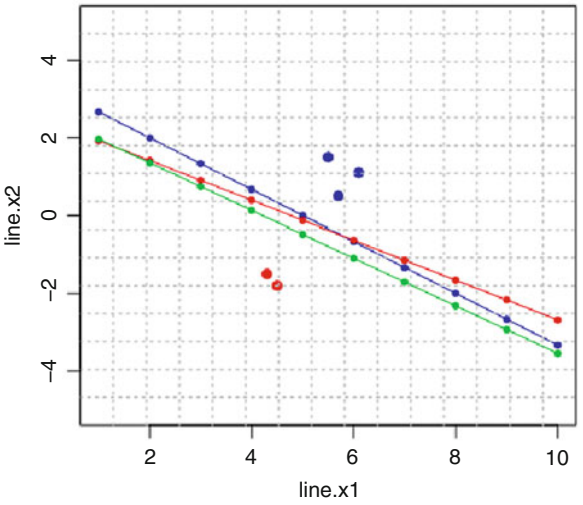

Fig. 9.3 A possible classifier for five points

```
mgamma.g = matrix(rep(gamma.g,5),nrow=3,ncol=5)
mgamma.g

# assign class labels
label.s = matrix(c(data$V3),nrow=5,ncol=1,byrow=TRUE)
label.s

label.ss=matrix(rep(label.s,3),ncol=3)
label.ss

# determine label error
hat.y = weight.w %*% point.x + mgamma.g
hat.y

# check for minimum error
t(label.ss) * hat.y

# calculate SVM measure for optimization
measure.m = (weight.w %*% t(weight.w))/2
measure.m

# display the straight line and points
line.x1 = rep(0,10)
line.x2 = rep(0,10)

slope = weight.w[,1]/weight.w[,2]
intercept=gamma.g/weight.w[,2]

line.x1 = 1:10
line.x2 = -intercept[1]-slope[1]*line.x1

```

```
50 plot(line.x1, line.x2, type="o", col="blue", pch=20, ylim=c(-5,5)
       )
51 points(point.x[1,], point.x[2,], col=ifelse(data$V3==1,"blue","
       red"), pch=19)
52 grid(15, 15, lwd=2)
53
54 line.x1 = 1:10
55 line.x2 = -intercept[2]-slope[2]*line.x1
56 lines(line.x1, line.x2, type="o", col="red", pch=20, ylim=c(-5,5)
       )
57
58 line.x1 = 1:10
59 line.x2 = -intercept[3]-slope[3]*line.x1
60 lines(line.x1, line.x2, type="o", col="green", pch=20, ylim=c
       (-5,5))
61
62 # optimization - select the weights that correspond
63 # to the smallest measure.
64 diag(measure.m)
```

This program has produced the figure presented in Fig. 9.3, and we can see the five points in the data domain with two classes separated by the same three straight lines considered previously. The program has also produced svm-measures that are calculated in line 37 and displayed in line 64. The difference between Listing 9.3, and Listings 9.2 and 9.1 is the calculations using the matrix form rather than the iterative steps; hence, the diagonal values of the matrix variable "measure.m" are the svm-measures for the three lines considered.

9.1.2 Linear Classifier: Nonseparable Linearly

In the above section we studied the classification problem of separable classes. If classes are nonseparable to an acceptable level, then a slack variable that describes the false positives must be introduced to the optimization problem described in Eq. (9.12). This will lead to the following equation [7]:

$$\begin{aligned} \underset{\mathbf{w},\gamma,\zeta\geq 0}{\text{Minimize:}} \quad & \frac{||\mathbf{w}||^2}{2} + \varepsilon(\zeta) \\ \text{subject to:} \quad & \mathbf{s}(\mathbf{w}\mathbf{x}' + \gamma\mathbf{I}) + \zeta \geq \mathbf{I} \end{aligned} \tag{9.14}$$

where the new variable ζ is called the slack variable, and it describes the acceptance of false positive and true negative errors in the classification results. Incorporating this error variable in the optimization goal of the support vector machine, we can obtain a better and acceptable classifier.

9.2 Lagrangian Support Vector Machine

The Lagrangian Support Vector Machine may be conceptualized as matrix expansion and matrix multiplications. The paper [9] by Mangasarian and Musicant provides mathematical modeling of this approach with a detail explanation. It is mathematically intensive; therefore, this approach is simplified in this section with the usage of matrix expansions and multiplications with a conceptualized example.

9.2.1 Modeling of LSVM

The modeling of the Lagrangian support vector machine can be easily understood if you have a clear understanding of the support vector machine theory presented in Chap. 7 and in the earlier sections of this chapter. The Lagrangian support vector machine may be explained based on the details in [7, 9], however, adopting the following optimization problem proposed by Dunbar [10] as a new formulation (called $L_1 + L_2 - \text{SVM}$) can help with better implementation:

$$\begin{aligned} \underset{\mathbf{W},\gamma,\hat{\xi}\geq 0}{\text{Minimize:}} \quad & \frac{\mathbf{W}'\mathbf{L_1}\mathbf{W}}{2} + \frac{\gamma^2}{2} + \frac{\lambda_2}{2}\hat{\xi}'\hat{\xi} \\ \text{subject to:} \quad & \mathbf{S}(\mathbf{X}\mathbf{W} + \hat{I}\gamma) + \hat{\xi} \geq \hat{I} \end{aligned} \tag{9.15}$$

where

$$\mathbf{W} = \begin{bmatrix} \mathbf{w} \\ \mathbf{v} \end{bmatrix}; \mathbf{L_1} = \lambda_1 \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}; \mathbf{S} = \begin{bmatrix} \mathbf{s} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}; \hat{\mathbf{I}} = \begin{bmatrix} \mathbf{I} \\ \mathbf{0} \end{bmatrix}; \tag{9.16}$$

$$\mathbf{X} = \begin{bmatrix} \mathbf{x} & \mathbf{0} \\ \mathbf{I} & \mathbf{0} \\ -\mathbf{I} & \mathbf{0} \end{bmatrix}; \hat{\xi} = \begin{bmatrix} \zeta \\ \mathbf{v} \\ 0 \end{bmatrix}. \tag{9.17}$$

It can be considered as the generalized model of the support vector machine presented in Eq. (9.14). Say, for example, if you substitute value 1 for λ_1 and λ_2 with appropriate matrix dimensions, then we will be able to obtain the same optimization model as the one presented in Eq. (9.14).

9.2.2 Conceptualized Example

The optimization model $L_1 + L_2 - \text{SVM}$ proposed in [10] and presented above may be simplified for its implementation by the process diagram with data flow illustrated in Figs. 9.4 and 9.5. These figures illustrate the approach using a simple example with two classes (labeled 1 and -1), and 5 data points $\{(5.5, 1.5), (5.7, 0.5), (6.1, 1.1), (4.5, -1.8), (4.3, -1.5)\}$, $\lambda_1 = 0.95$, and $\lambda_2 = 1$. Let us begin our explanation

with the "Start" from the top of the diagram. The data table is first divided into the data domain and the response set. The data domain is then processed via the left side of the diagram, and the response set is processed via the right side of the diagram, and the results are combined to generate the input to the Mangasarian and Musicant code, which provides the results in the bottom right-hand corner of the figure. The step to get the final weights for the slope and the intercept of the classifier is illustrated in Fig. 9.5. The process in Figs. 9.4 and 9.5 reflects the code in Listing 9.4. This conceptualized example provides a simple visual tool to understand the code of $L_1 + L_2 -$ SVM in Listing 9.4.

9.2.3 Algorithm and Coding of LSVM

The code in Listing 9.4 is written in R programming language, based on the process diagram with the data flow in Fig. 9.4, which was developed based on the $L_1 + L_2 -$ SVM formulation proposed by Dunbar [10] and the pseudo code presented by Mangasarian and Musicant in [9].

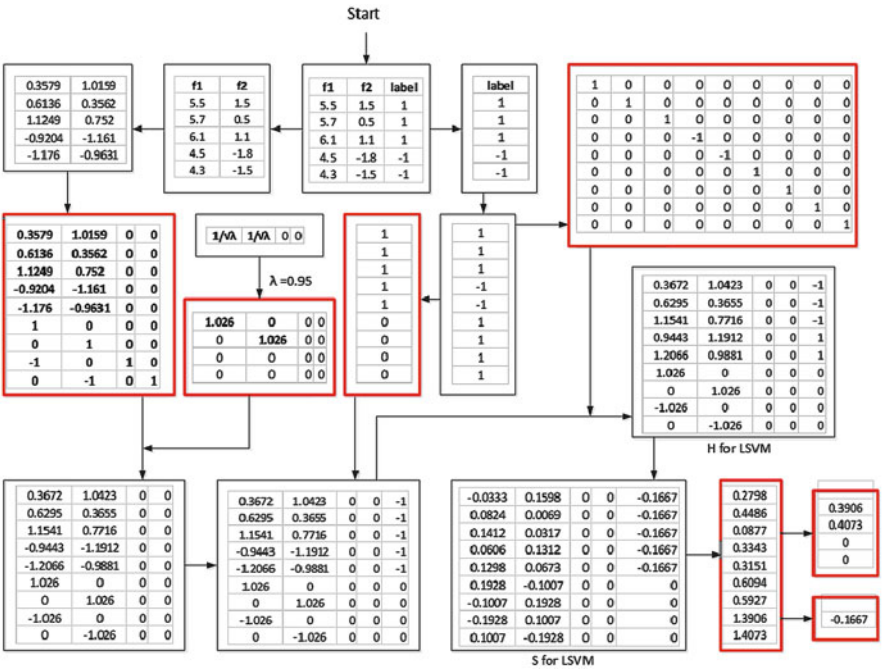

Fig. 9.4 The process diagram with data flow to develop the classifier based on $L_1 + L_2 -$ SVM and the Mangasarian and Musicant pseudo code in [9]

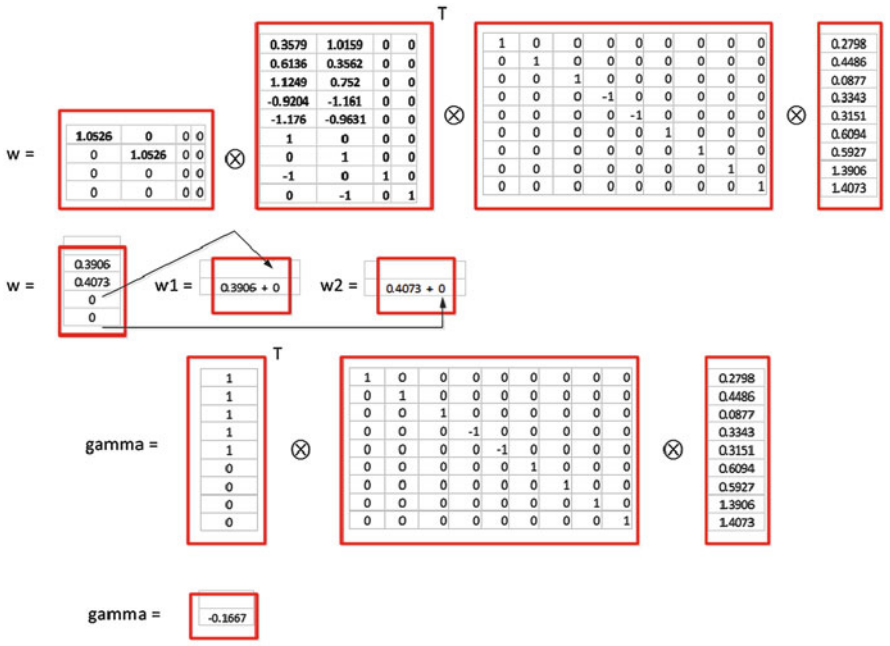

Fig. 9.5 Calculation of the final output of the process diagram presented in the previous figure in Fig. 9.4

Listing 9.4 An R programming example—implementation of LSVM

```
# Date: October 4th, 2014
# my-svm

data <- read.table("file3.txt", sep="")

A = data[1:2]
tmp1.A = scale(A)

scaled.A = tmp1.A[,]
class.label = data[3]
plot(scaled.A[,1], scaled.A[,2], col=ifelse(data$V3==1,"blue","
    red"), pch=19)

dim1.A = dim(A)[1]
dim2.A = dim(A)[2]

tmp1.D = c(t(class.label),rep(1,2*dim2.A))
diag.D = diag(tmp1.D)

tmp2.A = cbind(scaled.A, matrix(0,dim1.A,dim2.A))
tmp3.A = cbind(diag(dim2.A),0.0*diag(dim2.A))
tmp4.A = cbind(-diag(dim2.A),diag(dim2.A))

```

```
23
24 tmp.expanded.A = rbind(tmp2.A, tmp3.A, tmp4.A)
25 dim1.expanded.A = dim(tmp.expanded.A)[1]
26 dim2.expanded.A = dim(tmp.expanded.A)[2]
27 expanded.A = matrix(tmp.expanded.A, dim1.expanded.A, dim2.
       expanded.A)
28
29 lamda = 0.95
30 tmp1.lamda = sqrt(1/lamda)*diag(dim2.A)
31 tmp2.lamda = 0*diag(dim2.A)
32 tmp3.lamda = cbind(tmp1.lamda, tmp2.lamda)
33 tmp4.lamda = cbind(tmp2.lamda, tmp2.lamda)
34
35 expanded.lamda = rbind(tmp3.lamda, tmp4.lamda)
36
37 one.zero = c(rep(1,dim1.A), rep(0,2*dim2.A))
38
39 inter1.A = expanded.A \%*\% expanded.lamda
40 inter2.A = cbind(inter1.A, -one.zero)
41
42 lsvm.H = diag.D \%*\% inter2.A
43
44
45 ##### Mangasarian code starts #####
46 nu =1
47 lsvm.S = lsvm.H \%*\% solve((1/nu)*diag(2*dim2.A + 1) + t(lsvm.H)
       \%*\% lsvm.H)
48
49 lsvm.u = nu*(1-lsvm.S \%*\% (t(lsvm.H) \%*\% one.zero))
50
51 prev.lsvm.u = lsvm.u + 1
52
53 ii=0
54 alpha = 1.9/nu
55 while(ii < 1000 & norm(prev.lsvm.u-lsvm.u) > 0.0001) {
56   z = ((1/nu) + lsvm.H \%*\% t(lsvm.H) - alpha) \%*\% lsvm.u - 1
57   z = 1 + (abs(z) + z)/2
58   prev.lsvm.u = lsvm.u
59   lsvm.u = nu*(z-lsvm.S \%*\% (t(lsvm.H) \%*\% z))
60   ii = ii+1
61 }
62 ##### Mangasarian code ends #####
63
64 lsvm.y = expanded.lamda \%*\% t(expanded.A) \%*\% diag.D \%*\%
       lsvm.u
65 lsvm.y
66
67 lsvm.gamma = -t(one.zero) \%*\% diag.D \%*\% lsvm.u
68 lsvm.gamma
69
70 lsvm.w = lsvm.y[1:dim2.A] - lsvm.y[(dim2.A+1):(2*dim2.A)]
71 lsvm.w
72
73 intercept = lsvm.gamma/lsvm.w[2]
```

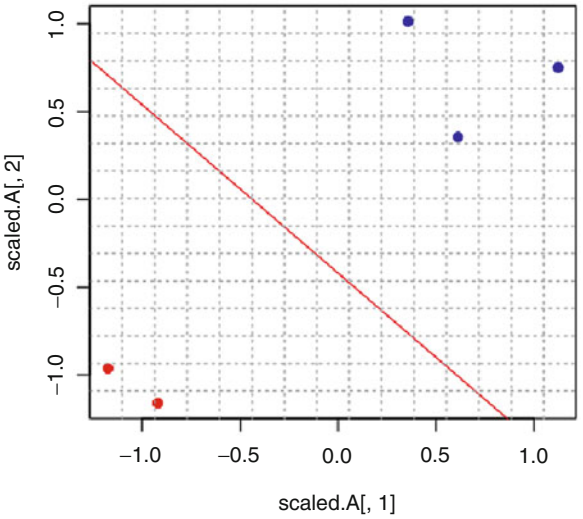

Fig. 9.6 The implementation of SVM and the classifier

```
74 slope = -lsvm.w[1]/lsvm.w[2]
75
76 abline(a=intercept, b=slope, col=2)
77 grid(15, 15, lwd=2)
```

The output of this program is presented in Fig. 9.6. It shows the scatter plot of the input data in file "file3.txt" and the support vector machine classifier calculated by this program. We can easily agree with this linear classification result.

9.3 Nonlinear Support Vector Machine

We have seen that the scatter plots play a major role in classification by facilitating the domain divisions. In the scatter plots, the dimension (i.e., each axis) is defined by a feature, and the space defined by the feature is called the vector space. The scatter plot describes the relationship between the features, and thus the correlated and uncorrelated data points can be identified in the vector space. The classification (in other words, the domain division) may be carried out either in a vector space or in a feature space, where the vector space is defined as the space that contains the scatter plot of the original features, and the feature space is defined as the space that contains the scatter plot of the transformed features using kernel functions [3].

9.3.1 Feature Space

Suppose there are p features $X_1, X_2, \ldots, X_p$ and the ith observation is denoted by $x_{i1}, x_{i2}, \ldots, x_{ip}$, where $i = 1, \ldots, n$. Then we can plot these n data points in a p-dimensional space to form a multidimensional scatter plot. This space is the vector space, and it displays both the magnitude and the directional information of the data. This set of p features may be transformed to a new set of d features using a polynomial kernel [3]. This space is called the feature space and, in general, each data point in the feature space carries information about a single data point in the vector space. The advantage of a feature space is that the nonseparable classes in the vector space may be turned into separable classes using a right choice of a kernel. However, finding a kernel and generating such a transformation are not simple.

$$\phi : R^p \rightarrow R^d \tag{9.18}$$

where R^p is the vector space (original domain) and R^d is the feature space, which is high dimensional (generally $d >> p$). It is possible to find ϕ that helps transform the vector space to a feature space where the classes are linearly separable. Hence, the support vector machine classifier in the feature space can be written as follows:

$$\begin{aligned} \underset{\mathbf{w},\gamma}{\text{Minimize:}} \quad & \frac{||\mathbf{w}||^2}{2} \\ \text{subject to:} \quad & \mathbf{s}(\mathbf{w}\phi(\mathbf{x}') + \gamma\mathbf{I}) \geq \mathbf{I} \end{aligned} \tag{9.19}$$

However, $\phi(\mathbf{x})$ is high dimensional, thus the computation becomes very expensive. This can be tackled using an approach called a *kernel trick*. Very useful lecture notes on kernel trick can be found at http://www.cs.berkeley.edu/~jordan/courses/281B-spring04/lectures/lec3.pdf.

9.3.2 Kernel Trick

The usefulness of the kernel trick technique is explained in this section using the data points shown in Fig. 9.7. This figure shows classes that are not linearly separable. However, if we transform them to a three-dimensional feature space (higher dimensional) using the following transformation, then we can obtain linear separability as shown in Fig. 9.8:

$$\phi(u_1, u_2) = ({u_1}^2, {u_2}^2, u_1 u_2) \tag{9.20}$$

Therefore, the support vector machine technique can be applied to this higher dimensional space, and a hyperplane can be derived as a classifier. In many real applications, such linear separability may be achieved in a very high-dimensional space, which makes it infeasible to apply the support vector machine techniques.

This is where the kernel tricks help. What exactly is the kernel trick? It is explained with the following simple example: Let us take two points (u_1, u_2) and (v_1, v_2) from the two-dimensional space presented in Fig. 9.7. Then we can have their transformed points as follows:

$$\begin{aligned} \phi(u_1, u_2) &= (a{u_1}^2, b{u_2}^2, cu_1u_2) \\ \phi(v_1, v_2) &= (a{v_1}^2, b{v_2}^2, cv_1v_2) \end{aligned} \tag{9.21}$$

Let us now define a new function, which is called the kernel function, as follows:

$$k(u, v) = \phi(u_1, u_2).\phi(v_1, v_2) \tag{9.22}$$

It gives us

$$k(u, v) = a^2{u_1}^2{v_1}^2 + b^2{u_2}^2{v_2}^2 + c^2u_1v_1u_2v_2 \tag{9.23}$$

If we select $c^2 = 2ab$, then we can have

$$k(u, v) = (au_1v_1 + bu_2v_2)^2 \tag{9.24}$$

This can be written in the following matrix form:

$$k(u, v) = \left(\begin{bmatrix} u_1 & u_2 \end{bmatrix} * \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} * \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \right)^2 \tag{9.25}$$

It shows that even if the function ϕ transforms the original data domain to a higher dimensional domain, the product $\phi.\phi$ can be easily defined based on the data in the original domain. Therefore, we can conclude that the kernel functionpresented

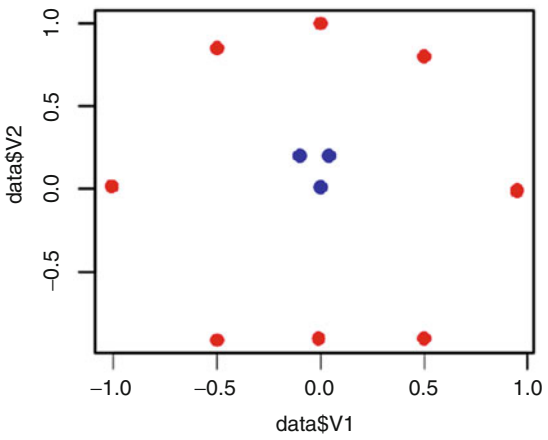

Fig. 9.7 It shows nonlinear classifiers are required to classify these two classes—a circle or an ellipse is needed to separate these two classes

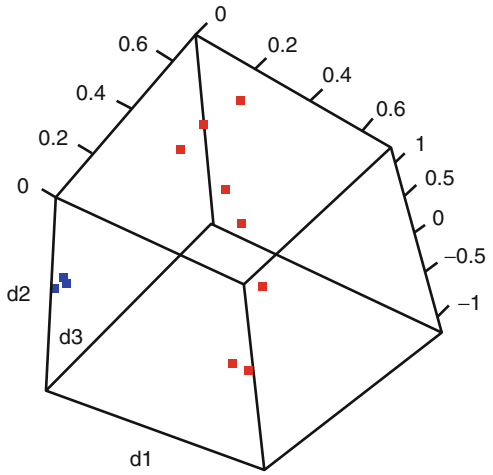

Fig. 9.8 This example shows that the classification of nonlinear separable classes is possible in a higher dimensional space called feature space

in Eq. (9.22) can be obtained by the matrix operations inside the original vector space rather than in the higher dimensional feature space. With the dual form [10] and the kernel function, the support vector machine can be applied in the original space (which is the lower dimension) with the same effect as its application inside the higher dimensional feature space.

Listing 9.5 An R programming example—Kernel trick example

```
1  # Date: May 23rd, 2015
2  # kernel-trick
3
4  library("rgl")
5
6  data <- read.table("file4.txt", sep="")
7
8  png("nonlinear2d.png", width=4, height=4, units="in",res=300)
9  plot(data$V1, data$V2, col=ifelse(data$V3==1,"blue","red"), pch
       =19)
10 grid(15, 15, lwd=2)
11 dev.off()
12
13 a = 1
14 b = 1
15 c = sqrt(2*a*b)
16
17 d1 = a*data$V1*data$V1
18 d2 = b*data$V2*data$V2
19 d3 = c*data$V1*data$V2
20
21 plot3d(d1,d2,d3,col=ifelse(data$V3==1,"blue","red"))
22 rgl.snapshot("somefile.png")
```

This R program reads the contents of "file4.txt" and first generates the scatter plot in Fig. 9.7. We can see the need for a nonlinear classifier. A kernel trick code in the rest of the program transforms the data to a higher dimension (in this case, 3D) and generates the plot as shown in Fig. 9.8. We can clearly see a linear separation in the transformed data.

The kernel trick generally increases the dimensionality and, in turn, it can increase the computational time.

9.3.3 SVM Algorithms on Hadoop

Big data classification requires the support vector machine to be implemented on the system like the RHadoop, which provides a distributed file system and the R programming framework. As we recall, this framework provides mapper(), reducer(), and mapreduce() functions. Therefore, the MapReduce programming on Hadoop distributed files systems allows the implementation of svm-based algorithms either inside the mapper() function or inside the reducer() function. Two examples are considered in this section: the first example adopts the five points, three lines svm-based example discussed earlier and implements it inside the reducer() function, and the second example implements the lsvm algorithm inside the mapper() function and illustrates the conceptual example presented previously.

9.3.3.1 SVM: Reducer Implementation

Once again, the $L_1 + L_2 -$ SVM formulation proposed by Dunbar [10] and the pseudo code presented by Mangasarian and Musicant in [9] have been used in this implementation. The RHadoop system requires a number of environment variables [11]; therefore, they are included in the program from lines 4 to 6 in Listing 9.6. They provide path to the home of MapReduce (to access necessary libraries), to the Hadoop command (for program execution), and streaming jar file in the Linux system. In line 8, the data is uploaded into the R environment. The implementations on RHadoop requires two libraries [12, 13], rmr2, and rhdfs, and they are included in lines 10 and 11.

Listing 9.6 An RHadoop example—LSVM as a reducer() function

```
1 # Date: May 21st, 2015
2 # svm-on-hadoop
3
4 Sys.setenv(HADOOP_HOME='/usr/lib/hadoop-0.20-mapreduce')
5 Sys.setenv(HADOOP_CMD='/usr/bin/hadoop')
```

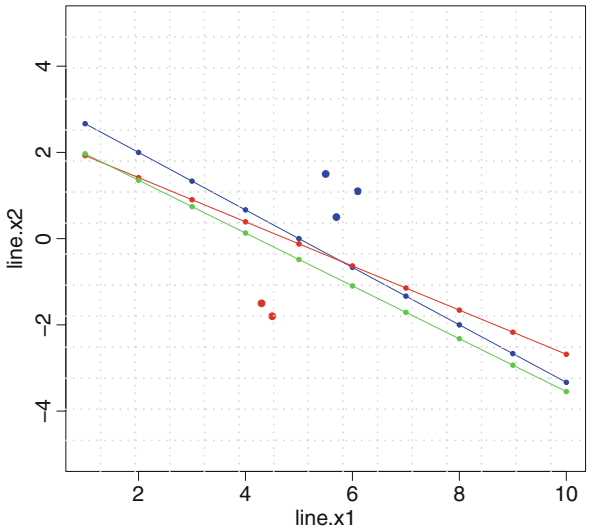

Fig. 9.9 It shows the implementation of SVM and the classifier

```
Sys.setenv(HADOOP_STREAMING='/usr/lib/hadoop-0.20-mapreduce/
    contrib/streaming/hadoop-streaming-2.0.0-mr1-cdh4.7.0.jar')

data <- read.table("file3.txt", sep="")

library(rmr2)
library(rhdfs)

hdfs.init()
data.content <- to.dfs(data)

data.map.fn <- function(k,v) {
  key <- 1
  val <- c(v[,1],v[,2],v[,3])
  #val <- c(scale(v[,1]),scale(v[,2]),v[,3])
  keyval(key,val)
}

data.reduce.fn <- function(k,v) {

  # extract feature and separate the labels
  rmr.str(v)
  point.x = matrix(v[1:10],nrow=2,ncol=5,byrow=TRUE)

  # select weights for parametrization
  weight.w = matrix(c(2,3,2.1,4.1,1.9,3.1),nrow=3,ncol=2,byrow=
      TRUE)
  weight.w

  # select intercepts for parametrization
```

```
gamma.g = matrix(c(-10,-10,-8),nrow=3,ncol=1,byrow=TRUE)
gamma.g

mgamma.g = matrix(rep(gamma.g,5),nrow=3,ncol=5)
mgamma.g

# assign class labels
label.s = matrix(c(v[11:15]),nrow=5,ncol=1,byrow=TRUE)
label.s

label.ss=matrix(rep(label.s,3),ncol=3)
label.ss

# determine label error
hat.y = weight.w %*% point.x + mgamma.g
hat.y

# check for minimum error
t(label.ss) * hat.y

# calculate SVM measure for optimization
measure.m = (weight.w %*% t(weight.w))/2
measure.m

# display the straight line and points
line.x1 = rep(0,10)
line.x2 = rep(0,10)

slope = -weight.w[,1]/weight.w[,2]
intercept = -gamma.g/weight.w[,2]

line.x1 = 1:10
line.x2 = intercept[1]+slope[1]*line.x1

plot(line.x1, line.x2, type="o", col="blue", pch=20, ylim=c
    (-5,5))
points(point.x[1,], point.x[2,], col=ifelse(data$V3==1,"blue","
    red"), pch=19)
grid(15, 15, lwd=2)

line.x1 = 1:10
line.x2 = intercept[2]+slope[2]*line.x1
lines(line.x1, line.x2, type="o", col="red", pch=20, ylim=c
    (-5,5))

line.x1 = 1:10
line.x2 = intercept[3]+slope[3]*line.x1
lines(line.x1, line.x2, type="o", col="green", pch=20, ylim=c
    (-5,5))

# optimization - select the weights that correspond
# to the smallest measure.
kk = diag(measure.m)
ii = which(kk == min(kk))
```

```
84    ss=c(slope[ii],intercept[ii])
85    keyval(k, ss)
86  }
87
88  classify <- mapreduce(input=data.content,
89                        map = data.map.fn,
90                        reduce = data.reduce.fn)
91
92  results = from.dfs(classify)
93  results
```

We should initialize the Hadoop environment and feed the data to it, and these tasks are presented in lines 13 and 14. Once these tasks are performed, we can define mapper() and reducer() functions, and then input them to the MapReduce model. The mapper() function is defined from line 16 to 21, and it creates a (key, value) pair from the input data. The integer value of 1 is used as key (see line 17) because of the single file processing, and the features in the first and the second column ($v[,1], v[,2]$) of the file are used as values (see line 18) in the key value pair presented in line 20. The reducer() function accepts the (key, value) pair, and uses the values to find the classifier (i.e., the slope and the intercept), which gives the minimum measure adopted in the svm's optimization approach [see Eq. (9.13)]. These steps are in the code from lines 23 to 86. Then these optimal parameters are tagged with the key (see line 85). Each block of code in this program is commented such that they are self explanatory. The block of code from line 58 to line 78 produces the scatter plot and the straight lines (possible svm-based classifiers) presented in Fig. 9.9. Because this program is executed inside the RHadoop it saves this result as Rplots.pdf file. The MapReduce model in lines 88–90 then assigns them to the variable called "classify." These data processing tasks occur inside the Hadoop environment, and they must be transferred to outside the Hadoop environment as performed with the command in line 92.

9.3.3.2 LSVM: Mapper Implementation

The mapper implementation of the $L_1 + L_2 -$ SVM formulation proposed by Dunbar [10] is presented, and it uses the pseudo code presented by Mangasarian and Musicant in [9]. In the reducer() implementation, the sorted data was obtained from the mapper(), and then the reducer() implemented the LSVM-based approach to derive the weights for the slope and intercept parameters of the svm classifier. But in the mapper implementation, the LSM-based approach is implemented in the mapper() function, and the slope and intercept parameters are calculated. These parameters are passed to the reducer() function. In both cases, a single key is used. However, multiple keys can be used to take advantage of the parallelization and sorting features of the MapReduce framework.

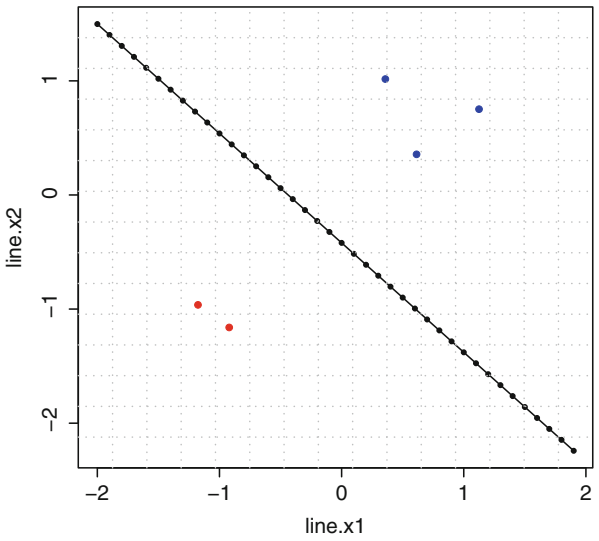

Fig. 9.10 The implementation of SVM on RHadoop and the classifier

Listing 9.7 An RHadoop example—LSVM as a mapper() function

```
# Date: May 21st, 2015
# lsvm-on-hadoop

Sys.setenv(HADOOP_HOME='/usr/lib/hadoop-0.20-mapreduce')
Sys.setenv(HADOOP_CMD='/usr/bin/hadoop')
Sys.setenv(HADOOP_STREAMING='/usr/lib/hadoop-0.20-mapreduce/
    contrib/streaming/hadoop-streaming-2.0.0-mr1-cdh4.7.0.jar')

data <- read.table("file3.txt", sep="")

library(rmr2)
library(rhdfs)

hdfs.init()
data.content <- to.dfs(data)

data.map.fn <- function(k,v) {
  key <- 1
  vw <- cbind(scale(v[,1]),scale(v[,2]))

  scaled.A <- matrix(vw, ncol=2, byrow=FALSE)
  class.label <- v[,3]
  rmr.str(class.label)

  dim1.A = dim(scaled.A)[1]
  dim2.A = dim(scaled.A)[2]

```

```
tmp1.D = c(t(class.label),rep(1,2*dim2.A))
diag.D = diag(tmp1.D)

tmp2.A = cbind(scaled.A, matrix(0,dim1.A,dim2.A))
tmp3.A = cbind(diag(dim2.A),0.0*diag(dim2.A))
tmp4.A = cbind(-diag(dim2.A),diag(dim2.A))

tmp.expanded.A = rbind(tmp2.A, tmp3.A, tmp4.A)
dim1.expanded.A = dim(tmp.expanded.A)[1]
dim2.expanded.A = dim(tmp.expanded.A)[2]
expanded.A = matrix(tmp.expanded.A, dim1.expanded.A, dim2.
    expanded.A)

lamda = 0.95
tmp1.lamda = sqrt(1/lamda)*diag(dim2.A)
tmp2.lamda = 0*diag(dim2.A)
tmp3.lamda = cbind(tmp1.lamda, tmp2.lamda)
tmp4.lamda = cbind(tmp2.lamda, tmp2.lamda)

expanded.lamda = rbind(tmp3.lamda, tmp4.lamda)
one.zero = c(rep(1,dim1.A), rep(0,2*dim2.A))

inter1.A = expanded.A %*% expanded.lamda
inter2.A = cbind(inter1.A, -one.zero)

#rmr.str(scaled.A)
lsvm.H = diag.D %*% inter2.A

##### Mangasarian code starts ################
nu =1
lsvm.S = lsvm.H %*% solve((1/nu)*diag(2*dim2.A + 1) + t(lsvm.H)
     %*% lsvm.H)

lsvm.u = nu*(1-lsvm.S %*% (t(lsvm.H) %*% one.zero))

prev.lsvm.u = lsvm.u + 1

ii=0
alpha = 1.9/nu
while(ii < 1000 & norm(prev.lsvm.u-lsvm.u) > 0.0001) {
  z = ((1/nu) + lsvm.H %*% t(lsvm.H) - alpha) %*% lsvm.u - 1
  z = 1 + (abs(z) + z)/2
  prev.lsvm.u = lsvm.u
  lsvm.u = nu*(z-lsvm.S %*% (t(lsvm.H) %*% z))
  ii = ii+1
}
###### end of Mangasarian

lsvm.y = expanded.lamda %*% t(expanded.A) %*% diag.D %*% lsvm.u

lsvm.gamma = -t(one.zero) %*% diag.D %*% lsvm.u
lsvm.gamma

lsvm.w = lsvm.y[1:dim2.A] - lsvm.y[(dim2.A+1):(2*dim2.A)]
```

```
  lsvm.w

  intercept = lsvm.gamma/lsvm.w[2]
  slope = -lsvm.w[1]/lsvm.w[2]

  line.x1 = rep(0,10)
  line.x2 = rep(0,10)

  xx = -2.0
  for (i in 1:40) {
    line.x1[i] = xx
    line.x2[i] = slope*line.x1[i] + intercept
    xx=xx+0.1
  }

  plot(line.x1, line.x2, type="o", pch=20)
  points(vw[,1], vw[,2], col=ifelse(v[,3]==1,"blue","red"), pch
      =19)
  grid(15, 15, lwd=2)

  val <- c(intercept,slope)
  keyval(key,val)
}

data.reduce.fn <- function(k,v) {
  keyval(k, v)
}

classify <- mapreduce(input=data.content,
                      map = data.map.fn,
                      reduce = data.reduce.fn)

aa = from.dfs(classify)
aa
```

The output of this program is presented in Fig. 9.10. This is similar to the one in Fig. 9.6, except this classifier is obtain using the RHadoop and MapReduce computing tools. However, the results show a significant similarity. The slopes and the intercepts may be numerically compared to determine their similarities.

9.3.4 Real Application

In this real application, the hardwood floor and carpet floor data sets are used. As you recall, these data sets have 1024 observations in each with 64 features that correspond to the intensity values of the pixels. To show the performance of the support vector machine implemented in Listing 9.7 in a two-dimensional data domain, the features 48 and 49 are selected. The scatter plots of the data sets corresponding

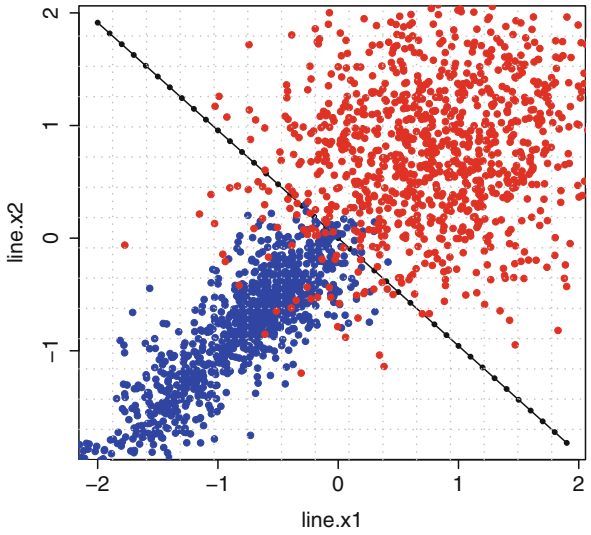

Fig. 9.11 The implementation of SVM on RHadoop, and the classifier with hardwood floor and carpet floor data sets

to these two features and the support vector machine classifier obtained using the algorithm are presented in Fig. 9.11. We can clearly see the linear classification performance of the Lagrangian support vector machine.

Problems

9.1. Code Revision

Revise the MapReduce programs presented in this chapter using the coding principles taught in Chap. 5.

9.2. Coding Efficiency

(a) Study the programs presented in the listings and draw the structure diagrams, data flow diagrams, and process diagrams based on the software engineering principles.
(b) Study the R programs in the Listings and improve their efficiencies using coding principles and modularization. Make this program more efficient using arrays and input files as well.

9.3. Comparison

Discuss the advantages and disadvantages of the mapper() and the reducer() implementations of the svm-based approaches. You may also run these implementations and obtain the system times to support the discussion.

9.4. Split-Merge-Split

(a) Assuming that you have completed the problem presented in "Problem 3.1," perform the same steps using the RHadoop system with the R programming framework.

(b) Compare the results that you obtained in (a) with the results that you obtained in Problem 3.1.

Acknowledgements I would like to thank Professor Vaithilingam (Jeya) Jeyakumar of the University of New South Wales, Australia, for giving me an opportunity to work with him and his research team on support vector machine problems and associated implementations to different applications. I also participated in the research focusing on enhancing the support vector machine technique and published our theory, results, and findings. This research contributed to this chapter.

References

1. M. A. Hearst, S. T. Dumais, E. Osman, J. Platt, and B. Scholkopf. "Support vector machines." Intelligent Systems and their Applications, IEEE, 13(4), pp. 18–28, 1998.
2. T. Hastie, R. Tibshirani, and J. Friedman. The Elements of Statistical Learning. New York: Springer, 2009.
3. B. Scholkopf, S. Mika, C. J. C. Burges, P. Knirsch, K. R. Muller, G. Ratsch and A. J. Smola. "Input space versus feature space in kernel-based methods," IEEE Trans. On Neural Networks, vol. 10, no. 5, pp. 1000–1017, 1999.
4. G. Huang, H. Chen, Z. Zhou, F. Yin and K. Guo. "Two-class support vector data description." Pattern Recognition, 44, pp. 320–329, 2011.
5. V. Franc, and V. Hlavac. "Multi-class support vector machine." In Proceedings of the IEEE 16th International Conference on Pattern Recognition, vol. 2, pp. 236–239, 2002.
6. http://en.wikipedia.org/wiki/Distance_between_two_straight_lines, accessed June 5th, 2015.
7. M. Dunbar, J. M. Murray, L. A. Cysique, B. J. Brew, and V. Jeyakumar. "Simultaneous classification and feature selection via convex quadratic programming with application to HIV-associated neurocognitive disorder assessment." European Journal of Operational Research 206(2): pp. 470–478, 2010.
8. V. Jeyakumar, G. Li, and S. Suthaharan. "Support vector machine classifiers with uncertain knowledge sets via robust optimization." Optimization, pp. 1–18, 2012.
9. O. L. Mangasarian and D. R. Musicant. 2000. "LSVM Software: Active set support vector machine classification software." Available online at http://research.cs.wisc.edu/dmi/lsvm/.
10. M. Dunbar. "Optimization approaches to simultaneous classification and feature selections," Technical Report (supervised by V. Jeyakumar) School of Mathematics and Statistics, The University of New South Wales, Australia, pp. 1–118, 2007.
11. http://www.meetup.com/Learning-Machine-Learning-by-Example/pages/Installing_R_and_RHadoop/
12. http://projects.revolutionanalytics.com/rhadoop/
13. http://bighadoop.wordpress.com/2013/02/25/r-and-hadoop-data-analysis-rhadoop/