hw3

uni:xw2598

2019/11/3

(a) Construct a linear support vector classifier.
(b) Construct a support vector classifier with Radial kernel.
(c) Construct a classifier using AdaBoost algorithm (with 50 boosting iterations) with decision stumps as weak learners.

Select the tuning parameter involved in SVM models appropriately. For each method, compute the test error and its standard error on the test set (synth.te). Provide a simple graphical visualization of the produced classification models (i.e. something similar to Figure 2.2 in the textbook [ESL]) and discuss your results.

```
train = read_table2("synth.tr.txt")
```

```
## Parsed with column specification:
## cols(
##   xs = col_double(),
##   ys = col_double(),
##   yc = col_integer()
## )
```

```
train$yc = as.factor(train$yc)
test = read_table2("synth.te.txt")
```

```
## Parsed with column specification:
## cols(
##   xs = col_double(),
##   ys = col_double(),
##   yc = col_integer()
## )
```

```
test$yc = as.factor(test$yc)
y.tr = train$yc
y = test$yc
x.tr = dplyr::select(train,-yc)
```

(a) Construct a linear support vector classifier.

```
linear.tune = tune.svm(yc~ys+xs, data=train, kernel="linear", cost=c(0.001, 0.01, 0.1, 1,10,100,1000))
summary(linear.tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   100
##
## - best performance: 0.144
```

```
##
## - Detailed performance results:
##    cost error dispersion
## 1 1e-03 0.568 0.03155243
## 2 1e-02 0.156 0.05481281
## 3 1e-01 0.148 0.04638007
## 4 1e+00 0.152 0.04131182
## 5 1e+01 0.148 0.04237400
## 6 1e+02 0.144 0.04299871
## 7 1e+03 0.144 0.04299871
```

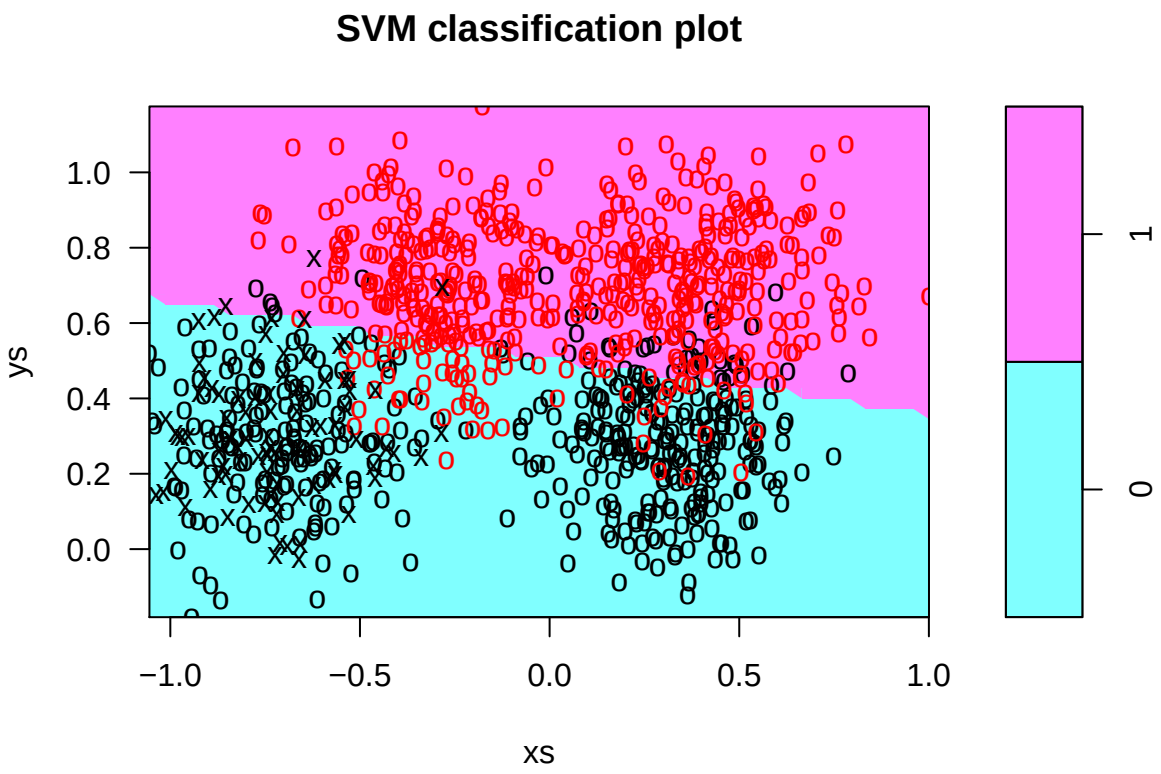
```
#Best tuning parameter C=0.1
best.linear = linear.tune$best.model
tune.test = predict(best.linear, newdata=test[-3])
#prediction table
table(tune.test, as.factor(test$yc))
```

```
##
## tune.test   0   1
##         0 452  57
##         1  48 443
```

```
#test error
linear.mse = sum((as.numeric(y)-as.numeric(tune.test))^2)/dim(test)[1]
#standard error
linear.se = sd((as.numeric(y)-as.numeric(tune.test))^2)/sqrt(dim(test)[1])
tibble(te = linear.mse,se = linear.se)
```

```
## # A tibble: 1 x 2
##      te      se
##   <dbl>   <dbl>
## 1 0.105 0.00970
```

```
plot(best.linear,test)
```



(b) Construct a support vector classifier with Radial kernel.

```
radial.tune = tune.svm(yc~ys+xs, data=train, kernel="radial",
                       gamma = c(0.01,0.1,0.5,1,10,100),
                       cost=c(0.001, 0.01, 0.1, 1,10,100))
summary(radial.tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  gamma cost
##    0.5    1
##
## - best performance: 0.132
##
## - Detailed performance results:
##    gamma  cost error dispersion
## 1  1e-02 1e-03 0.544 0.04299871
## 2  1e-01 1e-03 0.544 0.04299871
## 3  5e-01 1e-03 0.544 0.04299871
## 4  1e+00 1e-03 0.544 0.04299871
## 5  1e+01 1e-03 0.544 0.04299871
## 6  1e+02 1e-03 0.544 0.04299871
## 7  1e-02 1e-02 0.544 0.04299871
```

```
## 8  1e-01 1e-02 0.544 0.04299871
## 9  5e-01 1e-02 0.544 0.04299871
## 10 1e+00 1e-02 0.544 0.04299871
## 11 1e+01 1e-02 0.544 0.04299871
## 12 1e+02 1e-02 0.544 0.04299871
## 13 1e-02 1e-01 0.304 0.12536170
## 14 1e-01 1e-01 0.148 0.05672546
## 15 5e-01 1e-01 0.156 0.06095536
## 16 1e+00 1e-01 0.144 0.05719363
## 17 1e+01 1e-01 0.212 0.10507140
## 18 1e+02 1e-01 0.544 0.04299871
## 19 1e-02 1e+00 0.148 0.05006662
## 20 1e-01 1e+00 0.152 0.05902918
## 21 5e-01 1e+00 0.132 0.04638007
## 22 1e+00 1e+00 0.132 0.04638007
## 23 1e+01 1e+00 0.160 0.06253888
## 24 1e+02 1e+00 0.196 0.07647803
## 25 1e-02 1e+01 0.140 0.05416026
## 26 1e-01 1e+01 0.152 0.05593647
## 27 5e-01 1e+01 0.136 0.07351493
## 28 1e+00 1e+01 0.152 0.06746192
## 29 1e+01 1e+01 0.152 0.08390471
## 30 1e+02 1e+01 0.200 0.06531973
## 31 1e-02 1e+02 0.156 0.05146736
## 32 1e-01 1e+02 0.144 0.05719363
## 33 5e-01 1e+02 0.148 0.07554248
## 34 1e+00 1e+02 0.176 0.06850791
## 35 1e+01 1e+02 0.184 0.09082339
## 36 1e+02 1e+02 0.192 0.06477311
```

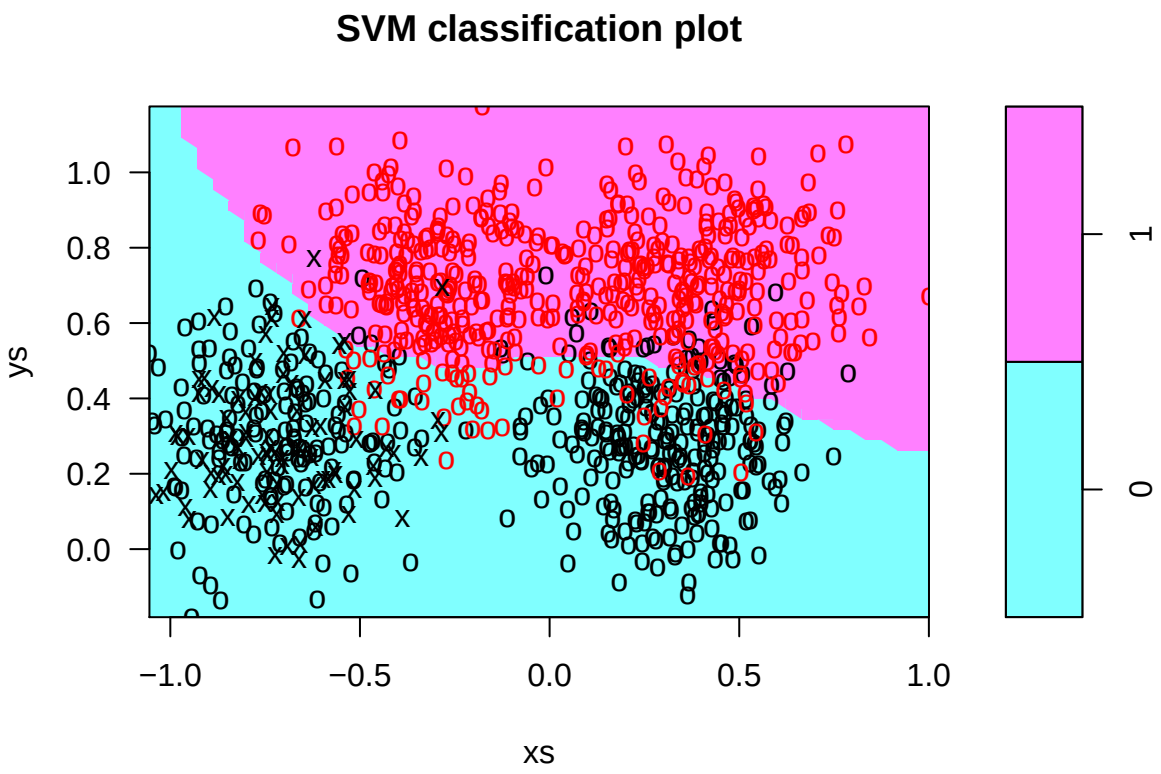
```
#Best tuning parameter C=10,gamma = 1
best.radial = radial.tune$best.model
pred.radial = predict(best.radial, newdata=test[-3])
#prediction table
table(pred.radial, as.factor(test$yc))
```

```
##
## pred.radial   0   1
##           0 458  53
##           1  42 447
```

```
#test error
radial.mse = sum((as.numeric(y)-as.numeric(pred.radial))^2)/dim(test)[1]
#standard error
radial.se = sd((as.numeric(y)-as.numeric(pred.radial))^2)/sqrt(dim(test)[1])
tibble(te = radial.mse,se = radial.se)
```

```
## # A tibble: 1 x 2
##      te      se
##   <dbl>   <dbl>
## 1 0.095 0.00928
```

```
plot(best.radial,test)
```

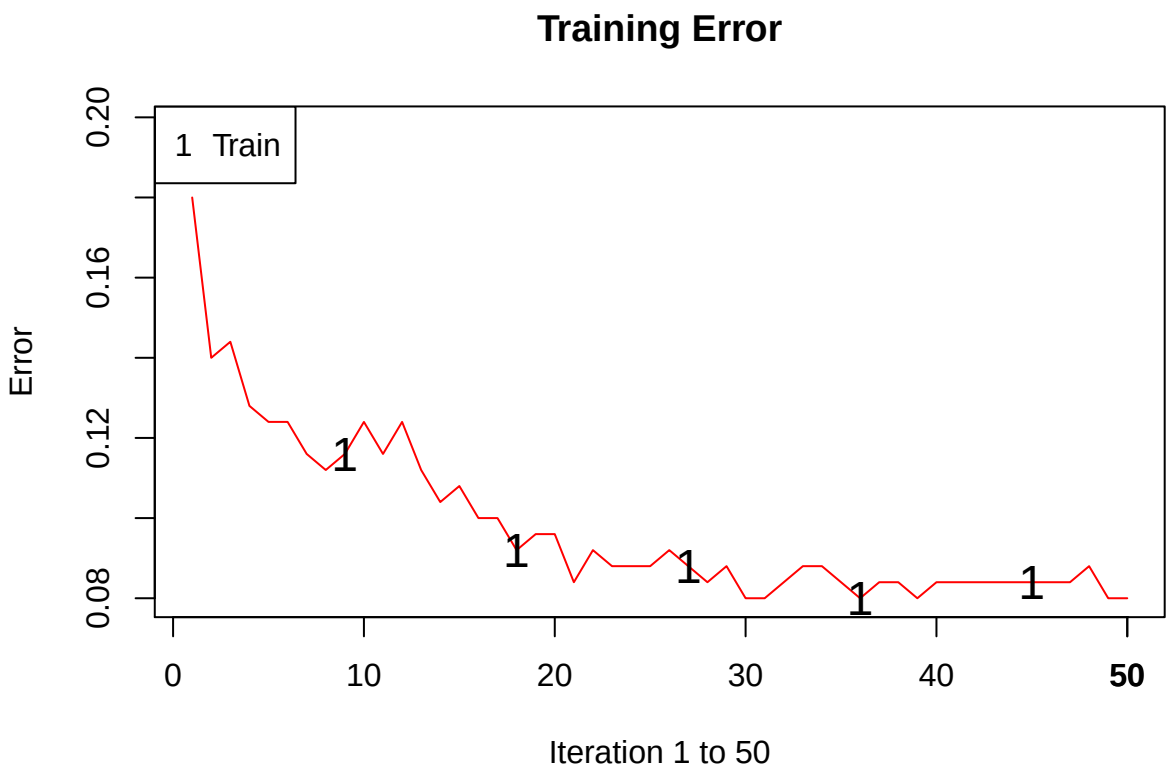


(c) Construct a classifier using AdaBoost algorithm (with 50 boosting iterations) with decision stumps as weak learners.

```
set.seed(12345678)
adaboost.model = ada(yc~ys+xs, data=train,type = "discrete",iter = 50)
pred.ada = predict(adaboost.model,test[-3])
#test error
ada.mse = sum((as.numeric(y)-as.numeric(pred.ada))^2)/dim(test)[1]
#standard error
ada.se = sd((as.numeric(y)-as.numeric(pred.ada))^2)/sqrt(dim(test)[1])
tibble(te = ada.mse,se = ada.se)
```

```
## # A tibble: 1 x 2
##      te      se
##   <dbl>   <dbl>
## 1 0.105 0.00970
```

```
plot(adaboost.model)
```




```
res = tibble(te = c(linear.mse,radial.mse,ada.mse),se = c(linear.se,radial.se,ada.se))
res = t(res)
colnames(res) = c("linear","radial","adaboost")
res
```

```
##         linear     radial    adaboost
## te 0.105000000 0.09500000 0.105000000
## se 0.009698921 0.00927691 0.009698921
```

Disscusion

The result shows that support vector classifier with Radial kernel has better performance. The decreasing rank of prediction performance measured by test error and its standard error is : support vector classifier with Radial kernel > AdaBoost algorithm > linear support vector classifier.

Also, the performance of AdaBoost algorithm will change due to the randomly boosting.

So in this case, we may choose support vector classifier with Radial kernel for training binary outcome data, which helps build a stably precise model.