CS498AML_homework2

Huamin Zhang, Rongzi Wang

Feb 5, 2017

Name: Huamin Zhang

NetID: huaminz2

Name: Rongzi Wang

NetID: rwang67

3.6(a) First, we read the data file and clean the data. We remove examples with missing value and there are examples left. Then we extract 5408 attributes and labels from data file and store them in `data` and `label` respectively. Also, about the `label` we change "active/inactive" to 1/-1.

```
library(klaR)

library(caret)

library(e1071)

library(rpart)

library(ggplot2)
#read data file
setwd("C:/Users/huaminz2/Documents")
data <- read.csv('K9.csv', header=FALSE)
#data cleaning on attributes
data <- data[, -5410]
data[data == '?'] <- NA
data <- na.omit(data) #remove these examples with missing value
label <- data[,5409]
data <- data[, -5409]  #extract the 5408 attributes
#data cleaning on labels
label <- as.matrix(label)
label[label %in% c("inactive")] <- -1 #change "inactive" to -1
label[label %in% c("active")] <- 1 #change "active" to 1
label <- as.numeric(label)
dim(data)

## [1] 31159  5408
```

Since the original data is to large(31129 examples), we select 2000 examples randomly from them and named `newdata`. Question(1) and Question(2) will use the same data since we set seed as 0 so that we can compare these two classfier. We selected 90% of the data as the trainging data and stored 5480 features in `train1_x` and labels in `train1_y`, and select the other data as the test data. About the training data `train1`, we still selected 80% as `train2` and the rest as `test2`. We use them to optimize lambda.

```
set.seed(1)
random_sample <- sample(1:dim(data)[1], 2000, replace = FALSE)
newdata<-data[random_sample, ]
newlabel<-label[random_sample]
#newdata <- data
#ewlabel <- label
#===================================================
# 9/1 partition where wtd is the index in the 90%
cv_partition <- createDataPartition(y=newlabel, p=.9, list=FALSE)
train1_x <- newdata[cv_partition, ]
test1_x <- newdata[-cv_partition, ]
train1_y <- newlabel[cv_partition]
test1_y <- newlabel[-cv_partition]


# 8/2 partition where wtd is the index in the 80%
cv_partition <- createDataPartition(y=train1_y, p=.8, list=FALSE)
train2_x <- train1_x[cv_partition, ]
test2_x <- train1_x[-cv_partition, ]
train2_y <- train1_y[cv_partition]
test2_y <- train1_y[-cv_partition]
sum(test1_y == 1)

## [1] 1

sum(test2_y == 1)

## [1] 3
```

Then we split `train2` into new training part and test part, and we use them to optimize a and b with svm using stochastic gradient descent. We write function `svm_sgd` to do this. In the training regime, there were 100 seasons. In each season, I applied 200 steps. For each step, I selected one data item uniformly at random (sampling with replacement), then stepped down the gradient. This means the method sees a total of 20000 data items. This means that there is a high probability it has touched 50% of total data. We trained with a steplength of 1/(0.01r+50), where r is the season. At the end of each season, We computed aT*a and the accuracy of the current classifier on the test examples.

```
#We use the function `svm_sgd` to optimize a and b.
svm_sgd<-function(data,label,lambda){

  #lambda = 0.01

  #define the initial a as a column vector with n ones
  #where n is the number of features
```

```r
  a <- as.matrix(rep(1, dim(data)[2])) #define the initial a as
  #define the initial b as 1
  b <- rep(1, 1)

  # run 100 seasons
  season <- 100
  weight_a <- rep(0,season)
  acc <- rep(0,season)
  for(i in 1:season){
    # 8/2 partition where wtd is the index in the 80%
    cv_partition<-createDataPartition(y=label, p=.8, list=FALSE)
    train_x<-data[cv_partition, ]
    test_x<-data[-cv_partition, ]
    train_y<-label[cv_partition]
    test_y<-label[-cv_partition]
    train_x <- matrix(as.numeric(as.matrix(train_x)),nrow = nrow(as.matrix(tr
ain_x)))
    #In the r_th season, define steplength as 1/(r + 50)
    eta <- 1/(i + 50)
    #run 200 steps in each seasons
    for(j in 1:200){
      #select one data item uniformly at random
      random_one <- sample(1:dim(train_x)[1], 1, replace = TRUE)
      #calculate a*x+b
      predict_y <- t(train_x[random_one, ]) %*% a + b
      #calculate the gradient of a and b
      pa = 0
      pb = 0
      if(predict_y * train_y[random_one]  >= 1){
        pa = lambda * a
        pb = 0
      }else{
        pa = lambda * a - train_y[random_one] * train_x[random_one, ]
        pb = -train_y[random_one]
      }
      #update a and b
      a <- a - eta * pa
      b <- b - eta * pb
    }
    #calculte the accuarcy
    test_x <- matrix(as.numeric(as.matrix(test_x)),nrow = nrow(as.matrix(test
_x)))
    predict_value <-  test_x %*% a + b
    predict_value[predict_value >= 1] = 1
    predict_value[predict_value< 1] = -1
    test_result <- (as.vector(predict_value) == test_y)
    test_accuracy <- sum(test_result)/length(test_y)
    acc[i] = test_accuracy
    weight_a[i] = t(a) %*% a
  }
  return(list(lambda=lambda, a=a, b=b,acc=acc,weight_a=weight_a))
}
```

Here we test our SVM implementation on some small balanced fake data. We create a fake data set with 3000 examples.

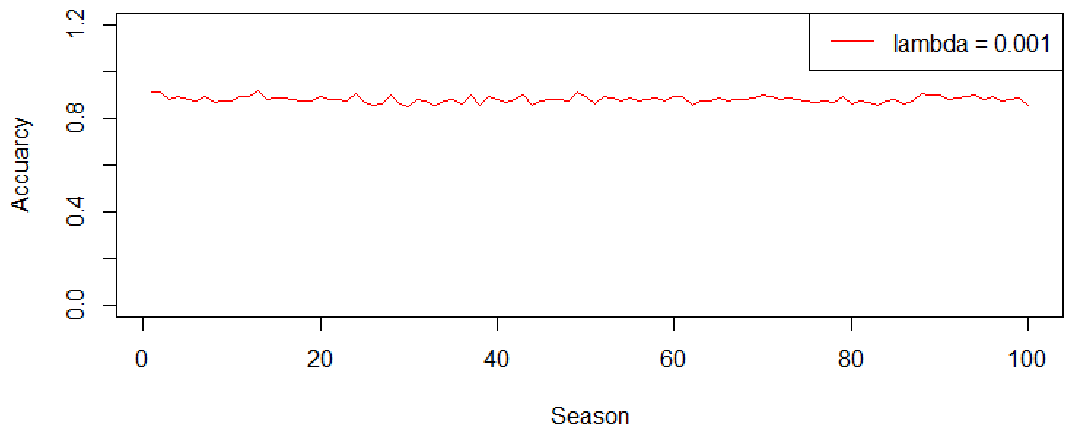
```
y = c(rep(-1,1500),rep(1,1500))
x1 = y + rnorm(3000,0,1)
x2 = y + rnorm(3000,0,1)
fake.data = data.frame(x1,x2,y)
m = fake.data[,c(1,2)]
n = fake.data[3]
n = as.numeric(as.matrix(n))
result_fake = svm_sgd(data=m,label=n,lambda = 0.001)
result_fake
```

```
## $lambda
## [1] 0.001
##
## $a
##          [,1]
## [1,] 1.277437
## [2,] 1.326696
##
## $b
## [1] -0.04469662
##
## $acc
##   [1] 0.9116667 0.9100000 0.8783333 0.8933333 0.8816667 0.8716667 0.895000
0
##   [8] 0.8700000 0.8716667 0.8750000 0.8916667 0.8950000 0.9166667 0.881666
7
##  [15] 0.8866667 0.8866667 0.8783333 0.8716667 0.8716667 0.8933333 0.880000
0
##  [22] 0.8833333 0.8733333 0.9033333 0.8683333 0.8533333 0.8633333 0.900000
0
##  [29] 0.8616667 0.8483333 0.8816667 0.8733333 0.8566667 0.8766667 0.883333
3
##  [36] 0.8600000 0.8966667 0.8583333 0.8900000 0.8783333 0.8700000 0.880000
0
##  [43] 0.9016667 0.8533333 0.8766667 0.8783333 0.8800000 0.8716667 0.910000
0
##  [50] 0.8900000 0.8633333 0.8916667 0.8883333 0.8766667 0.8850000 0.873333
3
##  [57] 0.8816667 0.8883333 0.8766667 0.8950000 0.8900000 0.8566667 0.875000
0
##  [64] 0.8733333 0.8850000 0.8750000 0.8833333 0.8800000 0.8850000 0.900000
0
##  [71] 0.8900000 0.8833333 0.8883333 0.8816667 0.8750000 0.8666667 0.876666
7
```

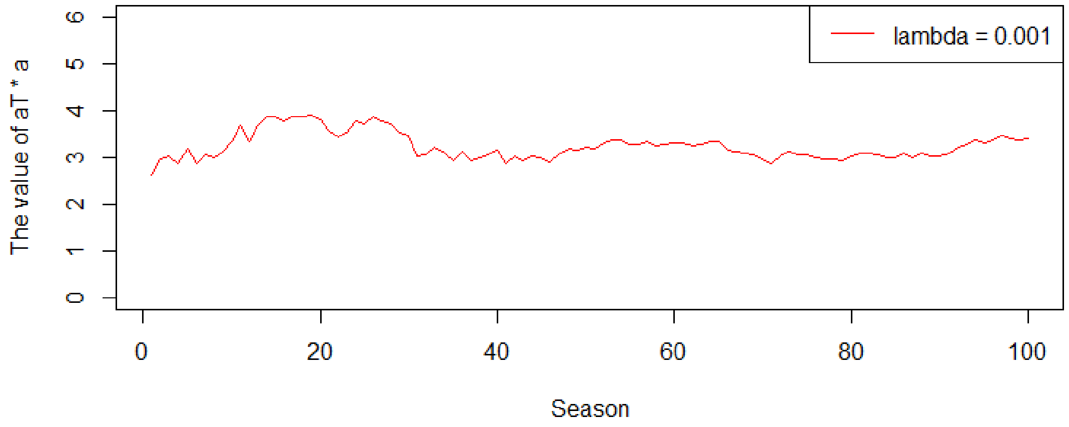
```
##  [78] 0.8666667 0.8916667 0.8616667 0.8766667 0.8666667 0.8583333 0.876666
7
##  [85] 0.8783333 0.8616667 0.8716667 0.9033333 0.8983333 0.8983333 0.881666
7
##  [92] 0.8850000 0.8900000 0.8983333 0.8800000 0.8900000 0.8750000 0.878333
3
##  [99] 0.8883333 0.8550000
##
## $weight_a
##   [1] 2.634727 2.974880 3.043494 2.882314 3.187999 2.871488 3.053006
##   [8] 3.002974 3.126654 3.347313 3.678403 3.328711 3.670725 3.874756
##  [15] 3.872856 3.767774 3.863214 3.876544 3.900749 3.817378 3.544222
##  [22] 3.418816 3.541625 3.766363 3.721782 3.879564 3.786651 3.716476
##  [29] 3.521270 3.477712 3.030838 3.052273 3.229474 3.084914 2.944272
##  [36] 3.117448 2.920543 2.982776 3.061797 3.166861 2.869095 3.042029
##  [43] 2.948156 3.042382 2.991410 2.898824 3.053473 3.172541 3.167935
##  [50] 3.201254 3.179495 3.301723 3.371298 3.363561 3.278751 3.285894
##  [57] 3.349651 3.252356 3.266184 3.310603 3.302530 3.247077 3.288197
##  [64] 3.326874 3.338668 3.160540 3.122252 3.077200 3.068294 2.956368
##  [71] 2.886531 3.015207 3.120371 3.062202 3.068335 3.009619 2.971630
##  [78] 2.977571 2.949145 3.022187 3.099258 3.081379 3.046825 3.000286
##  [85] 2.988602 3.077628 2.993920 3.090811 3.043012 3.022244 3.101956
##  [92] 3.209883 3.274582 3.378034 3.313136 3.384722 3.467861 3.414902
##  [99] 3.383269 3.391966
```

So the accuarcy of the SVM implementation on the fake data is 0.855. Here are two figures of how the value of aT*a and the accuracy change in each season

```r
x_axis = 1:100
plot(x_axis,result_fake$acc, type =  "l",ylim=c(0,1.2),main="",col="red",
     xlab = "Season",  ylab = "Accuarcy")
legend("topright",legend=paste("lambda = 0.001"), col="red", lty=1,
       pt.cex=0.7)
```



```r
plot(x_axis,result_fake$weight_a, type =  "l",ylim=c(0,6),main="",col="red",
     xlab = "Season",  ylab = "The value of aT * a")
legend("topright",legend=paste("lambda = 0.001"), col="red", lty=1,
       pt.cex=0.7)
```



Also, we write a verification function with `svmlight` and find the accuracy(0.92) is just a little higher than our svm implementation. So we think our svm implementation is correct.

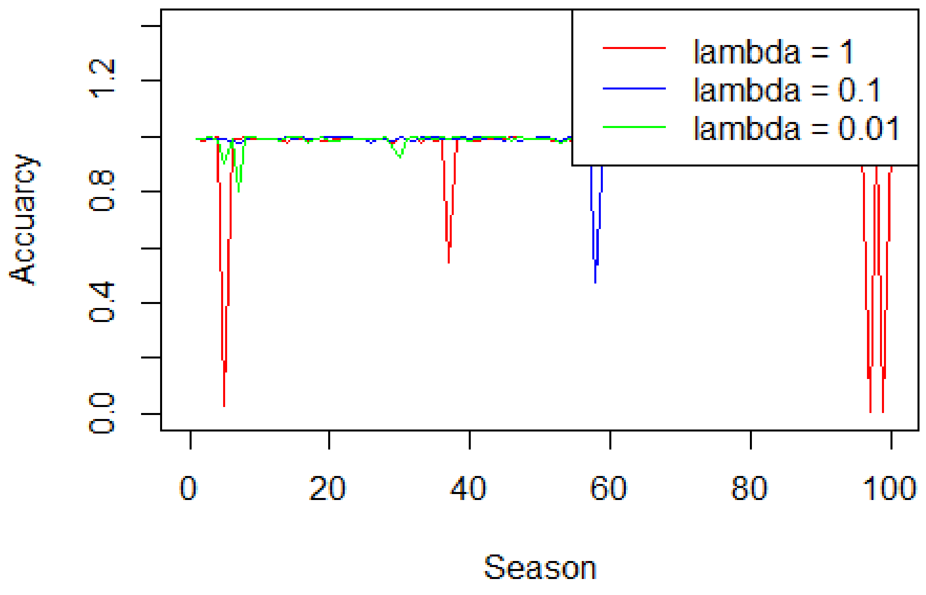
```r
#verification
verification<- function(data,label){
  cv_partition<-createDataPartition(y=label, p=.8, list=FALSE)
  train_x<-data[cv_partition, ]
  test_x<-data[-cv_partition, ]
  train_y<-label[cv_partition]
  test_y<-label[-cv_partition]
  svmlight.model<-svmlight(m, n,pathsvm='C:/Users/huaminz2/Documents/svm_ligh
t_windows64')
  svmlight_predict<-predict(svmlight.model, test_x)
  test_accuracy<-(svmlight_predict$class == test_y)
  test_score<-mean(test_accuracy)
  return(test_score)
}
test_score = verification(m,n)
test_score

## [1] 0.92
```

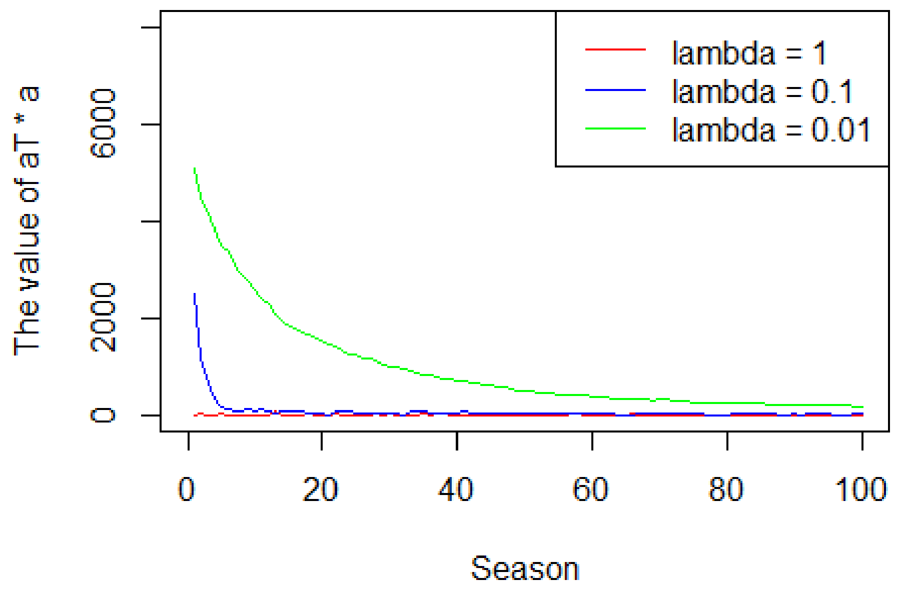
After verifying the correctness of our svm implementation, we test 3 lambda value of 1,0.1,0.01 with `train_2` from p53 mutants data set.

```r
lambda = c(1,0.1,0.01)
result_1 = svm_sgd(data=train2_x,label=train2_y,lambda = lambda[1])
result_2 = svm_sgd(data=train2_x,label=train2_y,lambda = lambda[2])
result_3 = svm_sgd(data=train2_x,label=train2_y,lambda = lambda[3])
```

```
x_axis = 1:100
plot(x_axis,result_1$acc, type =  "l",xlim=c(0,100),ylim=c(0,1.4),
    main="",col="red",xlab = "Season",  ylab = "Accuarcy")
lines (x_axis, result_2$acc , col =  "blue" )
lines (x_axis, result_3$acc , col =  "green" )
text.legend=c("lambda = 1","lambda = 0.1","lambda = 0.01")
legend("topright",legend=text.legend,col=c("red","blue","green"),
       lty=c(1,1,1),pt.cex=c(0.7,0.7,0.7),horiz=F)
```



```
plot(x_axis,result_1$weight_a, type =  "l",xlim=c(0,100),ylim=c(0,8000),
    main="",col="red",xlab = "Season",  ylab = "The value of aT * a")
lines (x_axis, result_2$weight_a , col =  "blue" )
lines (x_axis, result_3$weight_a , col =  "green" )
text.legend=c("lambda = 1","lambda = 0.1","lambda = 0.01")
legend("topright",legend=text.legend,col=c("red","blue","green"),
       lty=c(1,1,1),pt.cex=c(0.7,0.7,0.7),horiz=F)
```



We use each model of lambda = 1,0.1,0.01 to predict `train2` and then find a better lambda value.

```
test2_x <- matrix(as.numeric(as.matrix(test2_x)),nrow = nrow(as.matrix(test2_
x)))
accuracy= array(dim=3)
sensitivity = array(dim=3)
specificity = array(dim=3)
list(result_1,result_2,result_3)->result
for(i in 1:3){
  a = result[[i]]$a
  b = result[[i]]$b
  predict_value <-  test2_x %*% a + b
  predict_value[predict_value >= 1] = 1
  predict_value[predict_value< 1] = -1
  test2_factor <- as.factor(test2_y)
  confusion = confusionMatrix(as.factor(as.vector(predict_value)), test2_fact
or)
  accuracy[i] = confusion$overall["Accuracy"]
  sensitivity[i] = confusion$byClass["Sensitivity"]
  specificity[i] = confusion$byClass["Specificity"]
}
```

When lambda is 1, the accuracy is 0.9916667, the sensitivity is 1,the specificity is 0.

When lambda is 0.1, the accuracy is 0.9916667, the sensitivity is 1,the specificity is 0.

When lambda is 0.01, the accuracy is 0.9944444,the sensitivity is 1,the specificity is 0.3333333.

So we choose the model with lambda is 0.01. We use this model on `test1`

```
  test1_x <- matrix(as.numeric(as.matrix(test1_x)),nrow = nrow(as.matrix(test
1_x)))
  a = result[[3]]$a
  b = result[[3]]$b
  predict_value <-  test1_x %*% a + b
  predict_value[predict_value >= 1] = 1
  predict_value[predict_value< 1] = -1
  test1_factor <- as.factor(test1_y)
  confusion_result = confusionMatrix(as.factor(as.vector(predict_value)), tes
t1_factor)
```

```
## Warning in confusionMatrix.default(as.factor(as.vector(predict_value)), :
## Levels are not in the same order for reference and data. Refactoring data
## to match.
```

```
  confusion_result
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction  -1   1
##         -1 199   1
##         1    0   0
## 
##                Accuracy : 0.995
##                  95% CI : (0.9725, 0.9999)
##     No Information Rate : 0.995
##     P-Value [Acc > NIR] : 0.7358
## 
##                   Kappa : 0
##  Mcnemar's Test P-Value : 1.0000
## 
##             Sensitivity : 1.000
##             Specificity : 0.000
##          Pos Pred Value : 0.995
##          Neg Pred Value :   NaN
##              Prevalence : 0.995
##          Detection Rate : 0.995
##    Detection Prevalence : 1.000
##       Balanced Accuracy : 0.500
## 
##        'Positive' Class : -1
## 
```

So the accuracy of our svm implementation is 0.995,the sensitivity is 1.000, the specificity is NA.

3.6(b) We use the caret and klaR packages to build a naive bayes classifier for this data, The caret package does 10-fold cross-validation and can be used to hold out data. The klaR package can estimate class-conditional densities using a density estimation procedure.

```
ptm <- proc.time()
train1_y= as.factor(train1_y)
test1_y= as.factor(test1_y)
mycontrol<-trainControl(method='cv', number=10)
model<-train(train1_x, train1_y, method = "nb", trControl = mycontrol)
test_result<-predict(model, newdata = test1_x)
levels(test1_y) <- c(-1, 1)
confusion_NB = confusionMatrix(data=test_result,test1_y)
confusion_NB

## Confusion Matrix and Statistics
##
##            Reference
## Prediction  -1   1
##         -1 198   2
##         1    0   0
##
##                Accuracy : 0.99
##                  95% CI : (0.9643, 0.9988)
##     No Information Rate : 0.99
##     P-Value [Acc > NIR] : 0.6767
##
##                   Kappa : 0
##  Mcnemar's Test P-Value : 0.4795
##
##             Sensitivity : 1.00
##             Specificity : 0.00
##          Pos Pred Value : 0.99
##          Neg Pred Value :  NaN
##              Prevalence : 0.99
##          Detection Rate : 0.99
##    Detection Prevalence : 1.00
##       Balanced Accuracy : 0.50
##
##        'Positive' Class : -1
##

cat("Naive Bayes Running Time is:", proc.time()-ptm)

## Naive Bayes Running Time is: 468.91 0.29 469.95 NA NA
```

So the accuracy of the NB classifier is 0.99, the sensitivity is 1, the specificity is 0.

3.6(c) By comparing the result we obtained from SVM and Naïve Bayes classifier, we think SVM classifier is better on the data set. In our result, we observed that SVM has a higher accuracy than Naïve Bayes. Since Naïve Bayes assumes that the value of particular features is independent of the values of any other features given the classifiers. We guess that there are high correlation among some of the attributes in the dataset, and such dependency affects the accuracy of Naïve Bayes classifier. SVM prevents overfitting and is more adaptable to classify data with higher dimension.