Intro to ML PS 3

Takahiro Minami

2/17/2020

Part 1

question 1

```
library(tree)
library(ISLR)
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(MASS)
library(gbm)
```

```
## Loaded gbm 2.1.5
```

```
library(tidyverse)
```

```
## Registered S3 method overwritten by 'cli':
##   method     from
##   print.tree tree
```

```
## -- Attaching packages --------------------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.4
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------------------------ tidyverse_conflicts() --
## x dplyr::combine()  masks randomForest::combine()
## x dplyr::filter()   masks stats::filter()
## x dplyr::lag()      masks stats::lag()
## x ggplot2::margin() masks randomForest::margin()
## x dplyr::select()   masks MASS::select()
```

```
library(ipred)
set.seed(1000)
```

```
# load data
df1 <- read.csv("nes2008.csv",header=TRUE)

# number of features
p <- select(df1, -one_of("biden")) %>% ncol()
```

```r
# set lamda
lamda <- seq(from=0.0001,to=0.04, by=0.001)
```

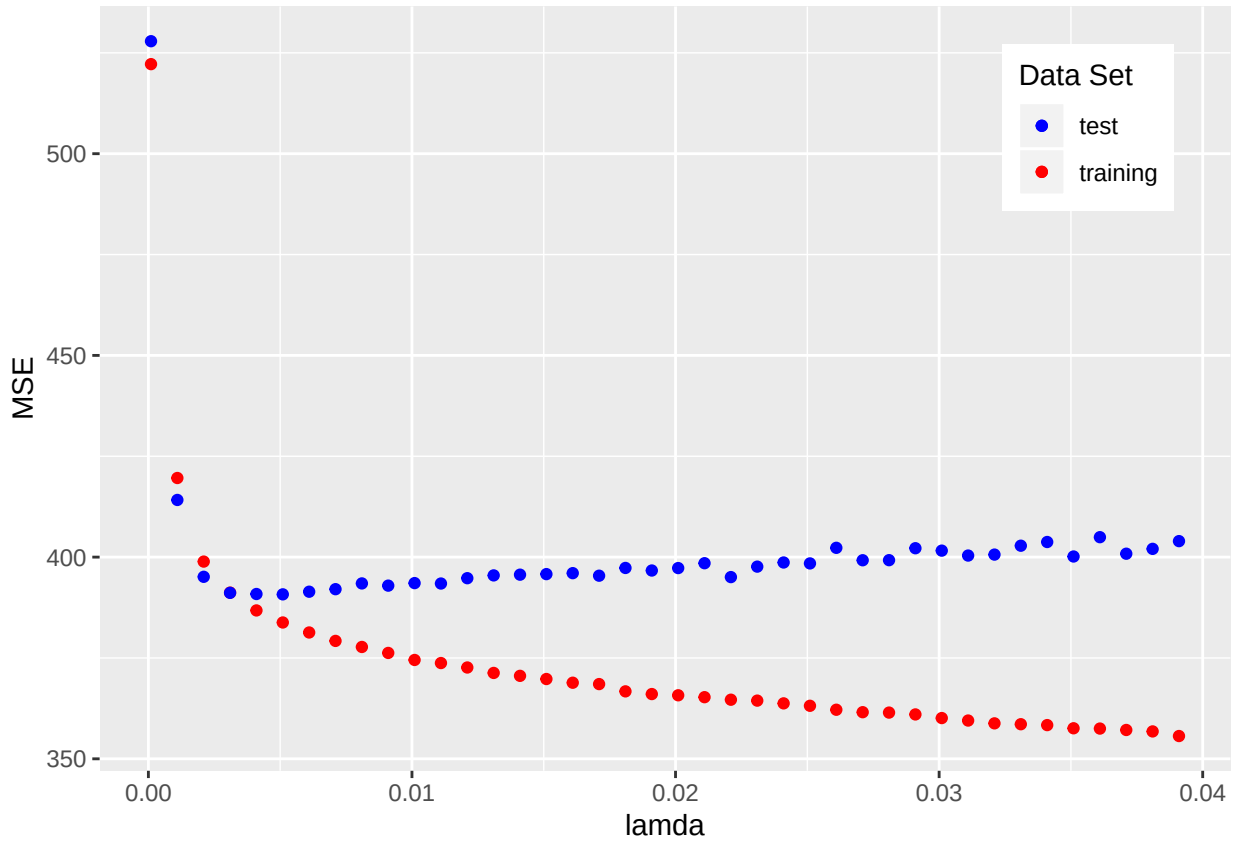
question 2

```r
set.seed(1000)
# 75% of data is training set
train=sample(1:nrow(df1),round(nrow(df1)*0.75))
# The remaining data is test set
test=c(1:nrow(df1))[!(c(1:nrow(df1)) %in% train)]
```

question 3

```r
# vectors for storing mes
mse_training <- c()
mse_test <- c()
set.seed(1000)

# boosting
for (i in 1:length(lamda)) {

  # Run model
  boost.biden <- gbm(biden ~ .,
                     data=df1[train,],
                     distribution="gaussian",
                     n.trees=1000,
                     shrinkage=lamda[i],
                     interaction.depth = 2)
  # calculate mes
  ## training
  preds_training = predict(boost.biden, newdata=df1[train,],
                           n.trees=1000)

  mse_training[i] = mean((preds_training - df1$biden[train])^2)

  preds_test = predict(boost.biden, newdata=df1[test,],
                       n.trees=1000)

  mse_test[i] = mean((preds_test - df1$biden[test])^2)
}

# plot
colors <- c("training"="red", "test"="blue")

data.frame(lamda=lamda, training=mse_training,test=mse_test) %>%
  ggplot()+
  geom_point(aes(x=lamda,y=training,color="training"))+
  geom_point(aes(x=lamda,y=test,color="test"))+
  labs(x = "lamda",y = "MSE",color = "Data Set") +
  scale_color_manual(values = colors)+
  theme(legend.position = c(.95, .95),
        legend.justification = c("right", "top"),
```

```
        legend.box.just = "right",
        legend.margin = margin(6, 6, 6, 6))
```



question 4

```
set.seed(1000)
# boosting with lamda=0.01
boost.biden <-
  gbm(biden ~ .,
    data=df1[train,],
    distribution="gaussian",
    n.trees=1000,
    shrinkage=0.01,
    interaction.depth = 2)

# fit to the test data
preds_boost = predict(boost.biden, newdata=df1[test,],n.trees=1000)
mes_boost = mean((preds_boost - df1$biden[test])^2)

paste0("MSE of boosting is ",round(mes_boost,2),".")
```

```
## [1] "MSE of boosting is 392.99."
```

The MSE is 392.99. This means that the average of squared deviation between the true feeling thermometers and the predicted values is 393. Given that the interval of feeling thermometer is 0 to 100, this MSE seems to be relatively large. Comparing this result with the one in question3, I found that the test MSE doesn't

change a lot even though λ increases more. The test MSE decreases sharply with small λ at first, but after λ becomes bigger than around 0.005, MSE is almost stable at around 400.

question 5

```
set.seed(1000)
# bagging
bag.biden <-
  bagging(
   formula = biden ~ .,
   data = df1[train,],
  nbagg = 100,
  coob = TRUE,
)

# fit to the test data
preds_bag = predict(bag.biden, newdata=df1[test,])
mes_bag = mean((preds_bag - df1$biden[test])^2)

paste0("MSE of bagging is ",round(mes_bag,2),".")
```

```
## [1] "MSE of bagging is 390.4."
```

question 6

```
set.seed(1000)
# random forest
forest.biden <-
  randomForest(biden ~ .,
               data = df1,
               subset = train)

# fit to the test data
preds_forest = predict(forest.biden, newdata=df1[test,])
mes_forest = mean((preds_forest - df1$biden[test])^2)

paste0("MSE of random forest is ",round(mes_forest,2),".")
```

```
## [1] "MSE of random forest is 400.63."
```

question 7

```
set.seed(1000)
# linear regression
lm.biden <- lm(biden ~ .,
               df1[train,])

# fit to the test data
preds_lm = predict(lm.biden, newdata=df1[test,])
mes_lm = mean((preds_lm - df1$biden[test])^2)

paste0("MSE of linear regression is ",round(mes_lm,2),".")
```

```
## [1] "MSE of linear regression is 386.18."
```

question 8

```
data.frame(
  method=c("boosting","bagging","random forest","linear regression"),
  MSE=c(round(mes_boost,2),round(mes_bag,2),round(mes_forest,2),round(mes_lm,2))
)
```

```
##               method    MSE
## 1           boosting 392.99
## 2            bagging 390.40
## 3     random forest 400.63
## 4 linear regression 386.18
```

I can conclude that the model with minimum test MSE fits the best to the data because MSE captures the deviation from the true values and predicted values from the model. In this case, the linear regression has the smallest test MSE. Then, linear regression is the best fit.

Part 2

```
library(e1071)
library(ISLR)
set.seed(2000)
df2 <- OJ
```

question 1

```
set.seed(2000)
# 800 data is training set
train2=sample(1:nrow(df2),800)
df2_train <- df2[train2,]
# The remaining data is test set
test2=c(1:nrow(df2))[!(c(1:nrow(df2)) %in% train2)]
df2_test <- df2[test2,]
```

question 2

```
# fit SVM
svmfit_1 <- svm(Purchase ~ .,
             data = df2_train,
             kernel = "linear",
             cost = 0.01,
             scale = FALSE); summary(svmfit_1)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = df2_train, kernel = "linear",
##     cost = 0.01, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
```

```
##
## Number of Support Vectors:  615
##
##  ( 309 306 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
# which obs were the support vectors
svmfit_1$index
```

```
##   [1]   2   3   4   5  13  15  16  17  19  20  21  22  25  28  29  36  39  40
##  [19]  41  42  45  50  51  54  57  59  62  65  68  69  71  72  74  77  81  82
##  [37]  86  91  98  99 112 113 114 117 118 120 121 123 124 127 128 130 131 133
##  [55] 134 135 138 140 142 148 152 155 157 161 164 168 170 174 177 182 187 188
##  [73] 191 197 198 200 201 202 204 206 208 209 211 214 215 216 217 219 222 223
##  [91] 224 225 227 229 230 232 234 240 242 244 245 246 248 250 257 262 263 264
## [109] 267 270 271 272 278 279 280 281 283 285 288 289 290 292 295 297 298 300
## [127] 302 304 306 307 310 311 312 316 321 322 324 326 327 332 333 334 338 341
## [145] 343 352 356 358 359 361 362 367 369 371 372 373 374 375 376 384 385 392
## [163] 393 399 401 413 419 420 422 423 426 431 432 433 435 436 438 444 447 448
## [181] 455 457 461 463 464 467 469 470 472 474 475 476 478 479 484 487 489 492
## [199] 501 502 505 509 511 513 527 529 530 532 534 537 539 540 543 550 555 557
## [217] 558 562 563 568 570 571 573 577 579 583 590 593 596 598 601 603 604 607
## [235] 610 613 619 621 623 624 626 628 634 641 644 647 648 651 654 660 661 662
## [253] 665 666 667 669 670 671 674 676 681 683 688 692 693 694 695 697 701 703
## [271] 704 708 710 713 714 717 725 728 733 734 735 737 738 743 745 746 748 750
## [289] 751 758 762 764 765 766 768 772 775 776 778 779 781 787 788 790 793 795
## [307] 796 798 799   6   8  12  18  23  24  26  27  30  33  35  38  43  44  46
## [325]  47  48  49  53  55  56  58  60  61  63  64  67  70  73  75  76  78  84
## [343]  85  87  88  93  97 100 103 104 106 107 109 111 116 122 126 129 132 136
## [361] 137 141 144 149 150 151 153 156 158 160 163 165 167 169 171 172 173 175
## [379] 176 178 179 180 181 183 184 185 186 189 190 193 194 195 199 203 207 210
## [397] 212 213 218 220 228 231 233 235 238 243 247 251 253 254 255 256 260 261
## [415] 269 273 274 277 286 287 294 299 301 305 319 320 329 330 336 339 342 344
## [433] 345 346 348 350 351 354 355 357 360 363 364 365 366 379 380 381 383 386
## [451] 387 388 389 391 394 395 396 397 398 400 402 404 406 408 409 411 412 417
## [469] 418 424 425 427 428 430 437 439 440 441 442 449 450 451 452 453 454 456
## [487] 459 460 462 466 471 477 480 482 486 488 490 491 495 496 497 504 506 510
## [505] 514 516 517 518 520 524 525 526 528 531 533 535 536 541 545 546 548 552
## [523] 553 559 561 569 574 575 576 578 580 584 585 586 587 588 595 599 606 609
## [541] 611 612 616 618 620 622 625 629 633 637 640 642 643 645 646 649 650 652
## [559] 653 656 657 658 659 664 672 673 677 679 680 682 685 686 689 690 691 696
## [577] 698 702 707 711 712 715 716 719 720 721 723 724 727 729 731 732 736 739
## [595] 741 742 744 749 752 753 756 757 759 760 763 767 770 771 774 777 785 786
## [613] 789 794 800
```

The result shows that there are 615 support vectors, 309 in one class (CH) and 306 in the other (MM). We set small penalty (small cost value),so the margin around the boundary is very wide ,and many observations in the range are to be considered as support vectors.

question 3

```
# prediction
predict1_test <- predict(svmfit_1, df2_test)
predict1_train <- predict(svmfit_1, df2_train)

# confusion matrix (test)
table(predicted = predict1_test,
      true = df2_test$Purchase)
```

```
##          true
## predicted  CH  MM
##        CH 157  78
##        MM   2  33
```

```
# error rate (training)
error1 <- mean(df2_train$Purchase!=predict1_train)*100
paste0("The training error rate is ",round(error1,2),"%.")
```

```
## [1] "The training error rate is 29.38%."
```

```
# error rate (test)
error2 <- mean(df2_test$Purchase!=predict1_test)*100
paste0("The test error rate is ",round(error2,2),"%.")
```

```
## [1] "The test error rate is 29.63%."
```

question 4

```
# set raing of cost
set.seed(2000)
list_cost <- c(0.01,0.1,1,10,100,1000)

# tuning
tune_cost <- tune(svm,
                  Purchase ~ .,
                  data = df2_train,
                  kernel = "linear",
                  ranges = list(cost=list_cost))

# CV errors
summary(tune_cost)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.17
##
## - Detailed performance results:
##    cost   error dispersion
```

```
## 1 1e-02 0.17250 0.03525699
## 2 1e-01 0.17000 0.02958040
## 3 1e+00 0.17125 0.03729108
## 4 1e+01 0.18000 0.04297932
## 5 1e+02 0.17500 0.03952847
## 6 1e+03 0.17625 0.04143687
```

```
# best
tuned_model <- tune_cost$best.model
summary(tuned_model)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = df2_train,
##     ranges = list(cost = list_cost), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
##
## Number of Support Vectors:  354
##
##  ( 178 176 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

I tried 6 cost values (0.01,0.1,1,10,100,1000), and the minimum error was achieved when the cost value is 0.1. Then, I can say 0.1 is the optimal value.

question 5

```
set.seed(2000)

# fit SVM
svmfit_opt <- svm(Purchase ~ .,
             data = df2_train,
             kernel = "linear",
             cost = 0.1,
             scale = FALSE); summary(svmfit_opt)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = df2_train, kernel = "linear",
##     cost = 0.1, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
```

```
##        cost:  0.1
##
## Number of Support Vectors:  449
##
##  ( 225 224 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
# prediction
predict_opt_test <- predict(svmfit_opt, df2_test)
predict_opt_train <- predict(svmfit_opt, df2_train)


# confusion matrix (test)
table(predicted = predict_opt_test,
      true = df2_test$Purchase)
```

```
##          true
## predicted  CH  MM
##        CH 142  29
##        MM  17  82
```

```
# error rate
## error rate (training)
error3 <- mean(df2_train$Purchase!=predict_opt_train)*100
paste0("The training error rate is ",round(error3,2),"%.")
```

```
## [1] "The training error rate is 17.12%."
```

```
## error rate (test)
error4 <- mean(df2_test$Purchase!=predict_opt_test)*100
paste0("The test error rate is ",round(error4,2),"%.")
```

```
## [1] "The test error rate is 17.04%."
```

The test error rate means that the model can predict the products customers buy (CH or MM) correctly in around 83% (the prediction is wrong in around 17%). Since the error rate doesn't change a lot between training and test error rates, I can say it is less likely that the over-fitting problem is not happening here. \

Compared with the result with smaller cost value (0.01), the confusion matrix says that the model with optimal cost value can predict MM much more correctly (predict 82 observations correctly out of 111 compared with 33 out of 111) while it predicts CH slightly less precisely (predict 142 observations correctly out of 159 compared with 157 out of 159). Overall, the test error rate is much less in the model with optimized cost value (around 17%) than the model with smaller cost value (around 30%). This shows that the model with optimized cost value fits better to the data.