HW 9

SDS348 Fall 2019

11/17/19

Raj Sharan - rs52382

This homework is due on Nov 10, 2019 at 11:59pm. Please submit as a pdf file on Canvas.

For all questions, include the R commands/functions that you used to find your answer. Answers without supporting code will not receive credit.

> **How to submit this assignment**
>
> All homework assignments will be completed using R Markdown. These `.Rmd` files consist of text/syntax (formatted using Markdown) alongside embedded R code. When you have completed the assignment (by adding R code inside codeblocks and supporting text outside of the codeblocks), create your document as follows:
>
> - Click the "Knit" button (above) to create an .html file
> - Open the html file in your internet browser to view
> - Go to `File > Print` and print your .html file to a .pdf
> - (or knit to PDF)
> - Upload the .pdf file to Canvas

Question 1

Back to Pokemon! (Sorry if that's not your thing...) There is a somewhat famous DataCamp module for predicting what makes a pokemon legendary and I wanted to put my own spin on it.

1a. (1 pts) First, run the following code to read in the data and drop the unnecessary variables. With the resulting dataset `poke`, how many are Legendary? How many are not?

```r
library(tidyverse)
poke <- read.csv("https://gist.githubusercontent.com/armgilles/194bcff35001e7eb53a2a8b441e8b2c6/raw/922
    row.names = "Name")
poke <- poke %>% dplyr::select(-X., -Total)

# your code here
```

your answer here, 1-2 sentences

1b. (2 pts) Predict Legendary from all of the remaining variables (recall the shortcut for this: `Legendary~.`) using a logistic regression. You don't need to convert it: R is smart enough to do this for you. Generate predicted probabilities for your original observations and save them as an object called `prob` in your environment (don't save them to the `poke` dataset). Use them to compute classification diagnostics with the `class_diag()` function from class (or the equivalent: it is declared in the preamble above so you should be able to use it in any subsequent code chunk). How well is the model performing per AUC? Provide a confusion matrix too and provide brief interpretation of what you see.

```r
# your code here
```

your answer here, 1-2 sentences

1c. (3 pts) Now perform 10-fold cross validation with this model. Summarize the results by reporting average classification diagnostics (e.g., from `class_diags()`) across the ten folds (you might get a `NaN` for ppv, which is fine). Do you see a substantial decrease in AUC when predicting out of sample? (Do you this model shows signs of overfitting?)

```r
set.seed(1234)
k = 10

# your code here
```

your answer here

1d. (3 pts) OK, now perform a LASSO regression on the same model (i.e., predicting Legendary from all predictor variables). You will need to have your predictors in a matrix and your response variable in a separate matrix. A good idea would be to use model.matrix(fit) from above to get the predictor-variable matrix (do not include the (intercept) term; i.e., drop the first column). To get the response, just do `as.matrix(poke$Legendary)`. Perform cross-validation via `cv.glmnet` to select the regularization parameter lambda and pick `lambda.1se` to regularize. Which coefficient estimates are non-zero? Report classification diagnostics. Also, provide a confusion matrix and talk about what you see. How does this compare with the full model from 1b?

```r
# install.packages('glmnet')
library(glmnet)

# your code here
```

your answer here, 1-2 sentences

1e. (2 pts) Re-run your 10-fold CV from above (1c), but this time use only the predictor variables which had non-zero LASSO coefficient estimates (like you did in lab). Note that you will need to use dummies/indicator variables for the significant types, so you might grab the corresponding vars from the model.matrix(fit) you used above. How does this model compare to the full model in terms of out-of-sample prediction?

```r
set.seed(1234)
k = 10

# your code here
```

your answer here, 1-2 sentences

1f. (1 pts) Let's do something *fancy-sounding*. We will use a random forest classifier to predict Legendary status from our predictor variables. In brief, this technique uses decision trees fitted (or "grown") on multiple bootstrapped samples of the data (i.e., with replacement), using bootstrapped samples of the predictor variables to predict (so not every model has all of the predictors: interesting twist!). The predictions (legendary/not) from each such tree are then averaged ("bagged" or bootstrap-aggregated); the proportion of trees predicting each pokemon as Legendary can be gotten and used as a probability for diagnostic purposes. Just like in CV, averaging performance (of many decision trees) across many (bootstrapped) samples reduces overfitting and makes our predictions more robust to noise. We will use these proportions to compare classification performance with our logistic regression (using the variables selected by lasso regularization, a technique also designed to curtail overfitting). Note that we are using the random forest function right out-of-the-box and better performance could be achieved if we tuned certain argumentes (e.g., ntree, mtry).

Because the technique is based on fitting a model to repeated bootstrapped samples and aggregating across predictions, we don't expect that using k-fold CV will show much difference. But let's see! Run the following code to (1) fit the random forest and produce the classification diagnostics and (2) perform 10-fold CV on the random forest model (to show it doesn't really change).

Compare the classification performance of the logistic regression using the variables lasso selected (i.e., the CV performance) with the classification performance of the random forest. Is there much difference?

```r
# install.packages('randomForest')
library(randomForest)
fit_rf = randomForest(Legendary ~ ., data = poke)

class_diag(fit_rf$votes[, 2], poke$Legendary)
```

```
##          acc      sens     spec       ppv       auc
## True 0.95125 0.5692308 0.985034 0.7708333 0.9696285
```

```r
######### CV
set.seed(1234)
k = 10

data1 <- poke[sample(nrow(poke)), ]
folds <- cut(seq(1:nrow(poke)), breaks = k, labels = F)

diags <- NULL
for (i in 1:k) {
    train <- data1[folds != i, ]
    test <- data1[folds == i, ]
    truth <- test$Legendary

    fit <- randomForest(Legendary ~ ., data = train)
    probs <- predict(fit, newdata = test, type = "prob")[, 2]

    diags <- rbind(diags, class_diag(probs, truth))
}

diags %>% summarize_all(mean)
```

```
##    acc      sens      spec       ppv       auc
## 1 0.95 0.5940476 0.9836056 0.7515476 0.9707662
```

your answer here

2a. (2 pts) Below, you are given 6 malignant patients and 6 benign patients. The vectors contain their predicted probabilities (i.e., the probability of malignancy from some model). If you compare every malignant patient with every benign patient, how many times does a malignant patient have a higher predicted probability than a benign patient? What proportion of all the comparisons is that? You can easily do this by hand, but you might try to find a way to use `expand.grid()`, `outer()`, or even a loop to calculate this in R (use ?functionname to read about these functions).

```r
malig <- c(0.49, 0.36, 0.57, 0.53, 0.61, 0.66)
benign <- c(0.41, 0.22, 0.26, 0.56, 0.31, 0.39)


# example of how to use expand.grid
expand.grid(lets = c("A", "B", "C"), nums = c(1, 2, 3))
```

```
##   lets nums
## 1    A    1
## 2    B    1
## 3    C    1
## 4    A    2
## 5    B    2
## 6    C    2
## 7    A    3
## 8    B    3
## 9    C    3
```

```
# example of how to use outer()
outer(c(4, 5, 6), c(1, 2, 3), "-")
```

```
##      [,1] [,2] [,3]
## [1,]    3    2    1
## [2,]    4    3    2
## [3,]    5    4    3
```

2b (1 pts) Now, treat the predicted probabilities as the response variable and perform an Wilcoxon/Mann-Whitney U test in R using `wilcox.test(group1, group2)` comparing the distribution of predicted probabilities for both groups (malig and benign). What does your W/U statistic equal?

```
## Your code here
```

2c (2 pts) Tidy this data by creating a dataframe and putting all predicted probabilities into one column and malignant/benign labels in another (you should end up with twelve rows, one for each observation). Use this data and ggplot to make a graph of the distribution of probabilities for both groups (histogram): fill by group. Leave default binwidth alone (it will look kind of like a barcode). Eyeballing and counting manually, for each benign (red) compute the number of malignants (blue) it is greater than (blue) and add them all up. This is the number of times a benign has a higher predicted probability than a malignant! In 1a you found the number of times a malignant beats a benign (i.e., has a higher predicted probability than a benign): What do those two numbers add up to?

```
## your code here
```

your answer here, 1-2 sentences

2d (2 pts) Set the cutoff/threshold at .2, .25, .3, .35, .4, .45, .5, .55, .6, .65, .7 and for for each cutoff, compute the true-positive rate (TPR) and the false-postive rate (FPR). You may do this manually, but I encourage you to try to figure out a way to do it in R (e.g., using `expand.grid` and `dplyr` functions). Save the TPR and FPR for each cut-off. Then make a plot of the TPR (y-axis) against the FPR (x-axis) using geom_path.

```
cutoffs <- c(0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6,
    0.65, 0.7)

# your code here
```

your answer here, 1-2 sentences

2e (1 pt) Use the `class_diags()` function to calculate the AUC (and other diagnostics on this data). Where in this assignment have you seen that AUC number before? (If you haven't seen that number before, go back and redo 2a and make sure you are answering both questions.)

```
# your code here
```

your answer here, 1-2 sentences

```
## R version 3.4.4 (2018-03-15)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 18.04.3 LTS
##
## Matrix products: default
## BLAS: /usr/lib/x86_64-linux-gnu/openblas/libblas.so.3
## LAPACK: /usr/lib/x86_64-linux-gnu/libopenblasp-r0.2.20.so
```

```
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] randomForest_4.6-12 glmnet_2.0-16       foreach_1.4.7
##  [4] Matrix_1.2-12       forcats_0.4.0       stringr_1.4.0
##  [7] dplyr_0.8.3         purrr_0.3.2         readr_1.3.1
## [10] tidyr_1.0.0         tibble_2.1.3        ggplot2_3.2.1
## [13] tidyverse_1.2.1     knitr_1.23
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_0.2.5 xfun_0.8         haven_2.1.1      lattice_0.20-35
##  [5] colorspace_1.4-1 vctrs_0.2.0      generics_0.0.2   htmltools_0.3.6
##  [9] yaml_2.2.0       rlang_0.4.0      pillar_1.4.2     glue_1.3.1
## [13] withr_2.1.2      modelr_0.1.4     readxl_1.0.0     lifecycle_0.1.0
## [17] munsell_0.5.0    gtable_0.3.0     cellranger_1.1.0 rvest_0.3.4
## [21] codetools_0.2-15 evaluate_0.14    broom_0.5.2      Rcpp_1.0.2
## [25] scales_1.0.0     backports_1.1.4  formatR_1.7      jsonlite_1.6
## [29] hms_0.5.0        digest_0.6.20    stringi_1.4.3    grid_3.4.4
## [33] cli_1.1.0        tools_3.4.4      magrittr_1.5     lazyeval_0.2.2
## [37] crayon_1.3.4     pkgconfig_2.0.2  zeallot_0.1.0    xml2_1.2.1
## [41] lubridate_1.7.4  assertthat_0.2.1 rmarkdown_1.14   httr_1.4.0
## [45] rstudioapi_0.10  iterators_1.0.12 R6_2.4.0         nlme_3.1-131
## [49] compiler_3.4.4
```

```
## [1] "2019-11-19 17:15:42 CST"
```

```
##                                    sysname
##                                    "Linux"
##                                    release
##                       "4.15.0-62-generic"
##                                    version
## "#69-Ubuntu SMP Wed Sep 4 20:55:53 UTC 2019"
##                                   nodename
##               "educcomp02.ccbb.utexas.edu"
##                                    machine
##                                   "x86_64"
##                                      login
##                                  "unknown"
##                                       user
##                                  "rs52382"
##                             effective_user
##                                  "rs52382"
```