Homework 3

Luis Noguera

5/31/2020

Initial Set Up

```
# Importing libraries and set up the work environment
library(knitr)
library(readxl)
library(tidyverse)
library(janitor)
library(rpart)
library(tidymodels)
library(nnet)
library(scales)

knitr::opts_chunk$set(cache = TRUE,
                      warning = FALSE,
                      echo = TRUE,
                      message = FALSE,
                      dpi = 180,
                      fig.width = 6,
                      fig.height = 4)


theme_set(theme_classic())
```

Question 1

Math heavy!

Question 2

Math heavy!

Question 3

Math heavy!

Question 4

Math heavy!

Question 5

Suppose there are several outcome variables {y1,y2, · · ·,yM} associated with a common set of predictor variables x={x1,x2, · · ·,xn}. One could train separate single output neural networks for each outcome ymor train a single network with multiple outputs, one for each ym.What are the relative advantages/disadvantages of these two respective approaches. In what situations would one expect each to be better than the other.

Training Multiple Networks:

Advantages

- It is possible to fine-tune multiple parameters in each network to estimate a specific response accurately.
- If the response variables are unrelated to one an another than training multiple networks is more beneficial than using only one model that tries to learn and estimate multiple outcomes.

Disadvantages

- A lot more computational time and power would be required when traning multiple newtorks and parameters.
- More parameters included on the training set, may overfit the model when trying to estimate on the testing or unseen data.

In summary, is appropiate to train multiple models for each variable when outcome variables are unrelated and computational power and time are not of concern. however, if the response variables are related training one network model to predict multiple variables could be beneficial; if the risk of overfitting exist on the training set due to small amount of data an also benefical approach is to used multi-task learning.

Question 6

Describe K—fold cross-validation. What is it used for. What are the advantages/disadvantages of using more folds (increasing K). When does cross—validation estimate the performance of the actual predicting function being used.

K-Folds is a method for splitting the training data into multiple groups or subsets, each group has a training, testing set and K represents the total number of groups/subsets. The statistical model is trained K folds, leaving out a different random set for evaluation. Once the split and fitting into the multiple sets is completed, the model's prediction error is averaged over each K fold.

Cross-calidation is used for evaluation purposes, specially when multiple models are evaluated. This technique is particularly advantageous in small there is not enough data to have only one set for validation and training purposes. In most cases, the statistical model chosen is the one that holds the lowest average error from the K-folds. Once the model has been chosen is usually trained on the whole dataset.

Advantages

- More of the data available can be used for traning purposes as it gets divided into multile groups. This provides an advantage of lower bias in the model.
- Second, the cross-validation estimation provides a more accurate and less bias representation of the expected prediction error for the final use case.

Disadvantages

- As the number of K-folds increases the computational power required for the fitting the model on each fold also increases.
- A higher number of cross-validation folds includes a higher number of observations in the dataset, making the training datasets very similar with one another, therefore prompt to overfitting and not accurate on new unseen data.

As opposed to estimating the performance on the actual predction function, K-fold cross validation measures the error on the modelling and fitting procedure; testing how good or bad are different hyperparameter tunning for example. If each of the parameters learned in all the evaluation folds are identical then K-fold cross validation would estimate the performance of the actual prediction function, not possible in practice.

Question 7

Math heavy!

Question 8

Consider near neighbor regression.(a) What are the advantages and disadvantages of using a large neighborhood size M ?(b) Describe the advantages and disadvantages of standardizing each of the variables to have the same variance before training.(c) Through a small simulated example, show that standardization in (b) may not always be a good idea.

Advantages

- Increasing the number of K in nearest neighbor regression could effectively make the model more precise, identifying patterns and relationships across many different dimensions.

Disadvantage

- Using a large neighbor size M, could lead to overfitting the evaluation model as the same relationships in data may not be repeated across the evaluation or new data.

Advantages of Standardizing

- With different measurement scales across multiple variables, one variable could have a much higher influence in the model then others if it's not standardized. Generally, variables with high variance can have major influence over the model so it is important to standardize in this case.

Disadvantages of Standardizing

- Categorical variables would need to be transformed into dummy variables in order to standardize the data with all the features, adding one more step to the pre-processing step.
- Interpreting coefficients in linear models is misleading as the original data has been manipulated.
- When converting a continous variable to categorical to capture different groups, the distance and variance of the variables it is of importance.

Standardization in linear model is not a good idea as it does not change or affect the model performance in any form. Here an example and evaluation on the training set:

```
set.seed(415)
train <- data.frame(
  X1 = sample(40:2000, 100, replace = T),
  X2 = sample(2:10, 100, replace = T)
)

train$y <- with(train,2*X1 + 3.4*X2 + rnorm(100,sd=60))

# Linear Model Fit not Standardized
fit_ns <- lm(y ~ X1 + X2, data = train)
summary(fit_ns)
```

```
##
## Call:
## lm(formula = y ~ X1 + X2, data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -146.673  -44.138   -2.423   45.910  167.099
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -10.76467   24.36059  -0.442   0.6596
## X1            1.98892    0.01231 161.546   <2e-16 ***
## X2            6.05044    2.68411   2.254   0.0264 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 64.68 on 97 degrees of freedom
## Multiple R-squared:  0.9964, Adjusted R-squared:  0.9963
## F-statistic: 1.347e+04 on 2 and 97 DF,  p-value: < 2.2e-16
```

```
train_scaled <- as.data.frame(scale(train, center = T, scale = T))

# Linear Model Fit Standardized
fit_standard <- lm(y ~ X1 + X2, data = train_scaled)
summary(fit_standard)
```

```
##
## Call:
## lm(formula = y ~ X1 + X2, data = train_scaled)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.137213 -0.041291 -0.002267  0.042948  0.156322
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 9.976e-17  6.051e-03   0.000   1.0000
## X1          1.001e+00  6.195e-03 161.546   <2e-16 ***
## X2          1.396e-02  6.195e-03   2.254   0.0264 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06051 on 97 degrees of freedom
## Multiple R-squared:  0.9964, Adjusted R-squared:  0.9963
## F-statistic: 1.347e+04 on 2 and 97 DF,  p-value: < 2.2e-16
```

As shown in the previous summaries, both results show the same Adjusted R-squared and different coefficients, standardizing the data is not always a good idea, it would depend on the statistical model to be built.

Question 9

Spam Email. The data sets spam_stats315B_train.csv, spam_stats315B_test.csv and documentation for this problem are the same as in Homework 2 and can be found in theclass web page. You need first to standardize predictors and choose all the weights starting values at random in the interval [-0.5, 0.5].

9.0 Data Import

Loading the data files, assigning the header names and counting to check fro class imabalance.

Data Import

```
spam_train <- read_csv("spam_stats315B_train.csv",
    col_names = FALSE)

spam_test <- read_csv("spam_stats315B_test.csv",
    col_names = FALSE)

header_spam <- c("make", "address", "all", "3d", "our", "over", "remove",
  "internet","order", "mail", "receive", "will",
  "people", "report", "addresses","free", "business",
  "email", "you", "credit", "your", "font","000","money",
  "hp", "hpl", "george", "650", "lab", "labs",
  "telnet", "857", "data", "415", "85", "technology", "1999",
```

```
  "parts","pm", "direct", "cs", "meeting", "original", "project",
  "re","edu", "table", "conference", ";", "(", "[", "!", "$", "#",
  "CAPAVE", "CAPMAX", "CAPTOT","type")

colnames(spam_train) <- header_spam
colnames(spam_test) <- header_spam

# Checking for class imbalance
spam_train %>%
  count(type)
```

```
## # A tibble: 2 x 2
##    type     n
##   <dbl> <int>
## 1     0  1849
## 2     1  1218
```

Accuracy Functions

```
# Important Functions

pct_accuracy <- function(y_hat, y_test, threshold){
   y_hat[y_hat >  threshold] <- 1
  y_hat[y_hat <= threshold] <- 0
  correct <- y_hat == y_test
  pct_accurate <- sum(correct)/length(correct)
  return(pct_accurate)
}

# Missclasification RAtes - Missed in previous HWs
get_misclassification_rates <- function(model, threshold){

  y_hat <- predict(model, spam_test_proc)
  y_hat[y_hat >  threshold] <- 1
  y_hat[y_hat <= threshold] <- 0


  correct <- y_hat == y_test
  correct_spam    <- correct[y_test == 1]
  correct_nonspam <- correct[y_test == 0]

  misclassification_rate <- 1 - sum(correct)/length(correct)
  spam_misclassification_rate <- 1 - sum(correct_spam)/length(correct_spam)
  nonspam_misclassification_rate <- 1 - sum(correct_nonspam)/length(correct_nonspam)

  return(c(
    misclassification_rate,
    spam_misclassification_rate,
    nonspam_misclassification_rate)
  )
}
```

Pre-Processing

Scaling the data using tidymodels and splitting into training and testing set. Included an example of data pre-processed.

```
set.seed(415)

# Scaling Data using Tidymodels
spam_train_rec <- recipe(type ~ ., data = spam_train) %>%
  step_scale(all_predictors()) %>% # Normaliing the data
  step_zv(all_predictors()) %>% # Eliminating Varibales with zero variance
  prep(retain = TRUE)
```

Data Split

```
# Response label
y_test <- spam_test %>%
  select(type)
y_train <- spam_train %>%
  select(type)

# Splitting  data
spam_train_proc <- bake(spam_train_rec, new_data = spam_train) %>%
  select(-type)
spam_test_proc <- bake(spam_train_rec, new_data = spam_test) %>%
  select(-type)

head(spam_train_proc, 10)
```

```
## # A tibble: 10 x 57
##     make address   all   `3d`   our  over remove internet order  mail receive
##    <dbl>   <dbl> <dbl>  <dbl> <dbl> <dbl>  <dbl>    <dbl> <dbl> <dbl>   <dbl>
##  1 0           0 0     0      0     0          0    0     0     0       0
##  2 0           0 1.17  0.0693 0     0          0    0     0.396 0.356   0
##  3 0.189       0 0.796 0      0.201 0.415      0    0.348 0     0       0
##  4 0           0 0     0      0     0          0    0     0     0       0
##  5 0           0 0     0      0     1.40       0    0     0     0       0
##  6 0           0 1.05  0      0     1.69       0    0     0     0.819   0
##  7 0           0 0     0      0     0          0    2.22  0     0       0
##  8 0           0 1.31  0      0     0          0    0     0     0       0
##  9 0           0 0     0      0     0          0    0     0     0       0
## 10 0.316       0 1.39  0      0.310 0          0    0     0.720 0.464   0.538
## # ... with 46 more variables: will <dbl>, people <dbl>, report <dbl>,
## #   addresses <dbl>, free <dbl>, business <dbl>, email <dbl>, you <dbl>,
## #   credit <dbl>, your <dbl>, font <dbl>, `000` <dbl>, money <dbl>, hp <dbl>,
## #   hpl <dbl>, george <dbl>, `650` <dbl>, lab <dbl>, labs <dbl>, telnet <dbl>,
## #   `857` <dbl>, data <dbl>, `415` <dbl>, `85` <dbl>, technology <dbl>,
## #   `1999` <dbl>, parts <dbl>, pm <dbl>, direct <dbl>, cs <dbl>, meeting <dbl>,
## #   original <dbl>, project <dbl>, re <dbl>, edu <dbl>, table <dbl>,
## #   conference <dbl>, `;` <dbl>, `(` <dbl>, `[` <dbl>, `!` <dbl>, `$` <dbl>,
## #   `#` <dbl>, CAPAVE <dbl>, CAPMAX <dbl>, CAPTOT <dbl>
```

(a) Fit on the training set one hidden layer neural networks with 1, 2,..., 10 hidden units and different sets of starting values for the weights (obtain in this way one model for each number of units). Which structural model performs best at classifying on the test set?

First Model - Hidden Neural Layers

```
set.seed(415)

# Parameters
num_neurons <- seq(1:10)
num_reps <- 10
wt_rang = 0.3
threshold <- 0.5
accuracies <- c()


for(size in num_neurons){
  sum_accuracy <- 0


  for(i in c(1:num_reps)){

    set.seed(415)

    model <- nnet(
        spam_train_proc, y_train, size=num_neurons[size],
        linout = FALSE,  softmax = FALSE,
        censored = FALSE, skip = FALSE, rang = wt_rang, decay = 0,
        maxit = 100, Hess = FALSE, trace = FALSE
    )
    y_hat <- predict(model, spam_test_proc)
    sum_accuracy <- sum_accuracy + pct_accuracy(y_hat, y_test, threshold)
  }

  accuracies <- c(accuracies, sum_accuracy/num_reps)
}



#get the num_neurons corresponding with model that produced highest average accuracy
best_performing <- which.max(accuracies)
best_num_neurons <- num_neurons[best_performing]
```

The best number of hidden layer neurons is 4 which provides an accuracy of 94% in the testing set.

(b) Choose the optimal regularization (weight decay for parameters 0,0.1,...,1) for the structural model found above by averaging your estimators of the misclassification error on the test set. The average should be over 10 runs with different starting values. Describe your final best model obtained from the tuning process: number of hidden units and the corresponding value of the regularization parameter. What is an estimation of the misclassification error of your model?

Second Model - Weight Decays

```
set.seed(415)

# Parameters
decays <- seq(0, 1, .1) # Adding weight decay (0.1, 1)
num_neurons <- seq(1:10) # Same layer as before
num_reps <- 10
wt_rang = 0.5
threshold <- 0.5
accuracies <- c()


for(decay in decays){
  sum_accuracy <- 0


  for(i in c(1:num_reps)){

    set.seed(415)

    model <- nnet(
        spam_train_proc, y_train, size=best_num_neurons,
        linout = FALSE,  softmax = FALSE,
        censored = FALSE, skip = FALSE, rang = wt_rang, decay = decay ,
        maxit = 100, Hess = FALSE, trace = FALSE
    )

    y_hat <- predict(model, spam_test_proc)
    sum_accuracy <- sum_accuracy + pct_accuracy(y_hat, y_test, threshold)
  }

  accuracies <- c(accuracies, sum_accuracy/num_reps)
}



# Highest average accuracy by neuron and decay.
best_performing <- which.max(accuracies)

best_decay <- decays[best_performing]
```

The best performing model is achieved using 0.1 as the best weight decay.

Missclasification rate using both parameters Number of Neurons and Weight Decay.

```
#get misclassification rates for our best chosen parameters

model <- nnet(
    spam_train_proc, y_train, size= best_num_neurons,
    linout = FALSE, entropy = FALSE, softmax = FALSE,
    censored = FALSE, skip = FALSE, rang = wt_rang, decay = best_decay,
    maxit = 100, Hess = FALSE, trace = FALSE
```

```
)
misclassification_rates <- get_misclassification_rates(model, threshold)
```

Best Weight Decay: 0.1

Best Number of Layer Neurons: 4

Missclasification rate of the model: 4%

Spam misscalsification rate: 6%

Ham Missclasification Rate: 3%

(c) As in the previous homework the goal now is to obtain a spamfilter. Repeat the previous-point requiring this time the proportion of misclassified good emails to be less than 1%.

```
set.seed(415)
#for a given model
find_threshold <- function(model){

  thresholds <- seq(0,1,0.01)
  y_hat <- predict(model, spam_test_proc)

  for(thresh in thresholds){
    y_hat_nonspam <- y_hat[y_test == 0]
    y_hat_nonspam[y_hat_nonspam >  thresh] <- 1
    y_hat_nonspam[y_hat_nonspam <= thresh] <- 0
    nonspam_misclassification_rate <- sum(y_hat_nonspam)/length(y_hat_nonspam)

    if(nonspam_misclassification_rate <= 0.01) {break}
  }
  return(thresh)
}


# parameters
num_neurons <- seq(1:10)
weight_decays <- seq(0,1,.1)
wt_rang = 0.5

#fitting the modee to specified parameters
models <- list()

for(i in c(1:length(num_neurons))){
  for(j in c(1:length(decays))){

      model <- nnet(
        spam_train_proc, y_train, size=num_neurons[i],
        linout = FALSE, entropy = FALSE, softmax = FALSE,
        censored = FALSE, skip = FALSE, rang = wt_rang, decay = decays[j],
        maxit = 100, Hess = FALSE, trace = FALSE
    )
    models[[paste(i,j,sep="_")]] <- model
  }
}
```

```
# < 1% nonspam misclassification rate for each model
model_thresholds <- lapply(models, function(x) {find_threshold(x)})

#get the overall, spam, and non-spam misclassification rates at each threshold
misclassification_rates <- mapply(get_misclassification_rates, models, model_thresholds)

#find the model with the lowest overall misclassification rate

#(using forced < 1% nonspam threshold)
best_model_idx <- which.min(misclassification_rates[1,])
best_model <- models[[best_model_idx]]
best_misclassification_rates <- misclassification_rates[,best_model_idx]
```

Reporting on the best neural net model:

Best model accuracy for Ham emails: ($< 1\%$ missclasification)

Hidden layer neurons: 8 Weight Decay: 0.3

Overall Missclasification Rate: 5%

Spam Email Missclasification Rate: 11%

Ham Email Missclasification Rate: 1%