Homework 3

Due: end of day Saturday, February 4

Submission instructions: Submit one write-up per group on gradescope.com.

IMPORTANT:

- Write names of everyone that worked on the assignment on the submission.
- Specify every member of the group when submitting on Gradescope (https://help.gradescope.com/article/m5qz2xsnjy-student-add-group-members)

For this homework, we will be using the case *Retention Modeling at Scholastic Travel Company*. Read:

- Case: Retention Modeling at Scholastic Travel Company (A);
- Supplement: Retention Modeling at Scholastic Travel Company (B);

which are available on Canvas.

Your goal is to help David build a model for retention.

The following code will get you started.

1 Load relevant libraries

```
library(dplyr)
library(caret)
library(glmnet)
```

2 Load the data

Here we will load the data from the CSV data file, examine its structure, and fix the data types incorrectly identified by R when importing from CSV.

```
STCdata_A<-read.csv('travelData.csv')
STCdata_A<-STCdata_A[,-1]
```

You can use the function `str` to quickly check the internal structure of an R object. Here we are using it to investigate type of data in each column of the loaded data.

```
str(STCdata_A)
```

Notice that some columns are identified as numerical or integer, but really the should be factors.

For instance, we have that column `From.Grade`

```
n_distinct(STCdata_A$From.Grade, na.rm = FALSE)    ## n_distinct is a function from dplyr package
```

only has 11 levels. It might be a better idea to treat it as a factor instead.

You can fix incorrectly classified data types as follows:

```
STCdata_A <- mutate_at(STCdata_A, vars(From.Grade), as.factor)
```

We can check that indeed the column represents a factor:

```
str( STCdata_A$From.Grade )
```

Fix other columns that are numeric at the moment, but could be converted to factors. The following line first finds numeric columns and then identifies the number of unique elements in each one.

```
( unique.per.column <- sapply( dplyr::select_if(STCdata_A, is.numeric), n_distinct ) )
```

Let us convert every column that has less than 15 unique values into a factor. The following line identify names of such columns.

```
( column.names.to.factor <- names(unique.per.column)[unique.per.column < 15] )
```

From this, we can see that the columns `To.Grade`, `Is.Non.Annual.`, `Days`, `CRM.Segment`, `Parent.Meeting.Flag`, `MDR.High.Grade`, `School.Sponsor`, `NumberOfMeetingswithParents`, `SingleGradeTripFlag` can be converted to factors. We can also convert the output `Retained.in.2012.`

Convert these columns into factors.

```
STCdata_A <- mutate_at(STCdata_A, column.names.to.factor, as.factor)
```

Now let's take care of date columns.

```
date.columns = c('Departure.Date', 'Return.Date', 'Deposit.Date', 'Early.RPL', 'Latest.RPL',
                 'Initial.System.Date', 'FirstMeeting', 'LastMeeting')
STCdata_A <- mutate_at(STCdata_A, date.columns, function(x) as.Date(x, format = "%m/%d/%Y"))
```

And finally we change all the character columns to factors as well.

```
STCdata_A <- mutate_if(STCdata_A, is.character, as.factor)
```

Let's see what we have:

```
str(STCdata_A)
```

Pretty good!!!

3 Data preprocessing

The data contains a number of columns with missing values. Let's investigate. The following tells us the number of missing values in each column.

```
sapply(STCdata_A, function(x) sum(is.na(x)))
```

Dealing with missing values is a challenging problem, which could occupy a quarter of its own. The purpose of this homework is not to investigate in-depth approaches to dealing with missing values, but rather to investigate classification. For that reason, we take the following simple approach.

The function `fixNAs` below fixes missing values. The function defines reactions:

- adds a new category "FIXED_NA" for a missing value of a categorical/factor variable;
- fills zero value for a missing value of a numeric variable;
- fills "1900-01-01" for a missing value of a date variable.

Then it loops through all columns in the dataframe, reads their types, and loops through all the values, applying the defined reaction to any missing data point. In addition, the function creates a surrogate dummy variable for each column containing at least one missing value (for example, `Special.Pay_surrogate`), which takes a value of 1 whenever the original variable (`Special.Pay`) has a missing value, and 0 otherwise.

```r
# Create a custom function to fix missing values ("NAs") and
# preserve the NA info as surrogate variables
fixNAs <- function(data_frame){
  # Define reactions to NAs
  integer_reac <- 0
  factor_reac <- "FIXED_NA"
  character_reac <- "FIXED_NA"
  date_reac <- as.Date("1900-01-01")

  # Loop through columns in the data frame
  # and depending on which class the
  # variable is, apply the defined reaction and
  # create a surrogate

  for (i in 1:ncol(data_frame)) {
    if (class(data_frame[,i]) %in% c("numeric","integer")) {
      if (any(is.na(data_frame[,i]))) {
        data_frame[,paste0(colnames(data_frame)[i],"_surrogate")] <-
          as.factor(ifelse(is.na(data_frame[,i]),"1","0"))
        data_frame[is.na(data_frame[,i]), i] <- integer_reac
      }
    } else
      if (class(data_frame[,i]) %in% c("factor")) {
        if (any(is.na(data_frame[,i]))){
          data_frame[,i]<-as.character(data_frame[,i])
          data_frame[,paste0(colnames(data_frame)[i],"_surrogate")] <-
            as.factor(ifelse(is.na(data_frame[,i]),"1","0"))
          data_frame[is.na(data_frame[,i]),i]<-factor_reac
          data_frame[,i]<-as.factor(data_frame[,i])
        }
      } else {
        if (class(data_frame[,i]) %in% c("character")) {
          if (any(is.na(data_frame[,i]))){
            data_frame[,paste0(colnames(data_frame)[i],"_surrogate")]<-
              as.factor(ifelse(is.na(data_frame[,i]),"1","0"))
            data_frame[is.na(data_frame[,i]),i]<-character_reac
          }
        } else {
          if (class(data_frame[,i]) %in% c("Date")) {
            if (any(is.na(data_frame[,i]))){
              data_frame[,paste0(colnames(data_frame)[i],"_surrogate")]<-
                as.factor(ifelse(is.na(data_frame[,i]),"1","0"))
              data_frame[is.na(data_frame[,i]),i]<-date_reac
            }
          }
        }
      }
  }

  return(data_frame)
}
```

We apply the above defined function to our data frame.

```r
STCdata_A<-fixNAs(STCdata_A)
```

We can see that the columns do not have any missing values any more.

```r
any( sapply(STCdata_A, function(x) sum(is.na(x))) > 0)
```

Next, we combine the rare categories. Levels that do not occur often during training tend not to have reliable effect estimates and contribute to over-fit.

Let us check for rare categories in the variable `Group.State`.

```r
table(STCdata_A$Group.State)
```

Let us create a custom function to combine rare categories. The function again loops through all the columns in the dataframe, reads their types, and creates a table of counts for each level of the factor/categorical variables. All levels with counts less than the `mincount` are combined into "other." The function combines rare categories into "Other."+the name of the original variable (for example, `Other.State`). This function has two arguments:

- the name of the dataframe; and
- the count of observations in a category to define "rare."

```r
combinerarecategories<-function(data_frame,mincount){
  for (i in 1:ncol(data_frame)) {
    a<-data_frame[,i]
    replace <- names(which(table(a) < mincount))
    levels(a)[levels(a) %in% replace] <-
      paste("Other", colnames(data_frame)[i], sep=".")
    data_frame[,i]<-a
  }
  return(data_frame)
}
```

Let us combine categories with < 10 values in `STCdata` into "Other." Ultimately, it is going to depend on the person doing the analysis on what they decide to call "rare'".

```r
STCdata_A<-combinerarecategories(STCdata_A,10)
```

Let us look at `Group.State` again.

```r
table(STCdata_A$Group.State)
```

You can investigate other columns to see if everything looks fine.

4 Split the data into training and testing sets

This is a very important step, both conceptually and technically. Conceptually, because the goal of predictive modeling is not to build a model that fits well the data it trains on, but rather one that would best predict the new data. A test set is in this sense the best representation of what the "new data" may look like. Technically, to facilitate comparison between different models, we need to maintain the same IDs in the corresponding sets at all times. We will accomplishes this through two "tricks":

- a random seed ensures that the random-number generator is initialized identically in each run; and
- the `inTrain` vector is created once and can then be applied anytime the data needs to be split.

By default, the code sets 500 data points in the test set, and the remainder 1,889 into the training set.

```
# set a random number generation seed to
# ensure that the split is the same every time
set.seed(233)

inTrain <- createDataPartition(
  y = STCdata_A$Retained.in.2012.,
  p = 1888/2389,
  list = FALSE)
df.train <- STCdata_A[ inTrain,]
df.test <- STCdata_A[ -inTrain, ]
```

Let us check that both the training and test sets have a similar proportion of positive and negative cases.

```
print('Training set proportion:')
table(df.train$Retained.in.2012.) / nrow(df.train)
print('Test set proportion:')
table(df.test$Retained.in.2012.) / nrow(df.test)
```

5 Fitting a logistic regression model

Let us fit a logistic regression model with all the variables included on the training set.

```
lgfit.all <- glm(Retained.in.2012.~ .,
                 data=df.train,
                 family="binomial")
summary(lgfit.all)
```

The model is overfit. It has too many insignificant variables.

Let us fit a much simpler model. We will use stepwise regressions.

Recall stepwise regression from BUS 41100 Applied regression course. See, for example, Week 9 slides. You can also check Section 6.1.2 of the ISLR book.

There are three approaches to running stepwise regressions: backward, forward and both. We need to specify criterion for inclusion/exclusion of variables. We will use one based on Bayesian information criteria.

Observe the process of variables being added to the model, (labeled by "+" in the output), gradual expansion of the model, and improvement of BIC.

```
# Start from a null model with intercept only, and add one covariate at a time until maximum BIC.
lgfit.null <- glm(Retained.in.2012.~ 1,
                 data=df.train, family="binomial")

lgfit.selected <- step(lgfit.null,                    # the starting model for our search
                       scope=formula(lgfit.all),      # the largest possible model that we will consider.
                       direction="forward",
                       k=log(nrow(df.train)),         # by default step() uses AIC, but by
                                                      # multiplying log(n) on the penalty, we get BIC.
                                                      # See ?step -> Arguments -> k
                       trace=1)
```

The algorithm stops once none of the 1-step expanded models lead to a lower BIC.

This is the selected model.

```
summary(lgfit.selected)
```

You can predict probabilities from this model using the following.

```
phat.lgfit.selected <- predict(lgfit.selected,
                               newdata = df.test,
                               type = "response")
```

You will use these probabilities later.

While we are investigating variable selection in logistic regression models, let us also use a more modern approach to variable selection. We will use the lasso.

If you have not seen this in BUS 41100 Applied regression course, do not worry. We will provide more details in the Week 5. You can also check Section 6.2.2 of the ISLR book.

I provide the code to fit a lasso logistic regression model. We find coefficients β that minimize the deviance loss plus the penalty:

$$-2 \cdot \sum_{i=1}^{n} \log p(y_i, x_i; \beta) + \lambda \sum_{j=1}^{p} |\beta_j|.$$

Here, λ is the user chosen penalty that controls the flexibility of the fit.

First, we need to create a model matrix that will be used as an input to the package.

```
X <- model.matrix(formula(lgfit.all), STCdata_A)
#need to subtract the intercept
X <- X[,-1]

X.train = X[ inTrain, ]
X.test = X[ -inTrain, ]
```

Next, we run 5-fold cross-validation.

```
cv.l1.lgfit <- cv.glmnet(
  x       = X.train,
  y       = df.train$Retained.in.2012.,
  family  = "binomial",
  alpha   = 1,    #alpha=0 gives ridge regression
  nfolds  = 5)
```

We can plot the cross-validation curve, which shows us an estimate of out-of-sample deviance as a function of the tuning parameter λ. The x-axis represents to $-\log(\lambda)$. Therefore, on the left we have large values of λ and on the right we have small values of λ. At the top, you can see the number variables that were selected into the model. The two vertical dashed lines correspond to λ values that minimize the cross-validation error and the largest value of lambda such that error is within 1 standard error of the minimum.

```
plot(cv.l1.lgfit, sign.lambda=-1)
```

Let us know plot the fitted coefficients as a function of λ. Note that `cv.l1.lgfit$glmnet.fit` corresponds to a fitted glmnet object for the full data.

```
glmnet.fit <- cv.l1.lgfit$glmnet.fit
plot(glmnet.fit, xvar = "lambda")
abline(v = log(cv.l1.lgfit$lambda.min), lty=2, col="red")
abline(v = log(cv.l1.lgfit$lambda.1se), lty=2, col="green")
legend("topright", legend=c("min", "1se"), lty=2, col=c("red", "green"))
```

For our predictive model, we will use 1 standard error λ. Below you can see the variables that are selected by the lasso.

```r
betas <- coef(cv.l1.lgfit, s = "lambda.1se")
model.1se <- which(betas[2:length(betas)]!=0)
colnames(X[,model.1se])
```

We now use our model to predict probabilities on the test set.

```r
phat.l1.lgfit <- predict(glmnet.fit,
                         newx = X.test,
                         s = cv.l1.lgfit$lambda.1se,
                         type = "response")
```

6 Questions

6.1 How well does logistic regression do?

1. Create a confusion matrix for two logistic regression models build above. Use probabilities `phat.lgfit.selected` and `phat.l1.lgfit` to do so.

   To solve this question, you need to make a major decision. What should the cutoff or "threshold" for the probability be, above which you will label a customer as being classified as "retained?" In our case, the data is slightly unbalanced—about 60.72% of data points are in Class 1. For very unbalanced data, we would first need to balance it (over- or under-sample). In this case, the benefits of balancing are unclear, hence one can implement the average probability of being retained as a cutoff.

   Predict classification using 0.6072 threshold.

   What can we see from the confusion matrices?

2. Plot ROC curves for the two classifiers and report the area under the curve.

   Note that the AUC of an error-free classifier would be 100%, and an AUC of a random guess would be 50%. For values in-between, we can think of AUC as follows:

   - 90%+ = excellent,
   - 80–90% = very good,
   - 70–80% = good,
   - 60–70% = so-so, and
   - below 60% = not much value.

3. Plot lift curves for the two classifiers.

4. Create the profit curve (the amount of net profit vs the number of groups targeted for promotion) for the two classifiers. Suppose that the benefit of retaining a group is \$100, while the cost of a promotion is \$40.

   How many groups should be targeted to maximize the profit?

   How would this number change as the ratio between the benefit and cost changes?

   You can refer to the following code that plots a profit curve:

   ```r
   # Function to plot a profit curve
   #
   # Inputs:
   #  - benefitTP(FN/FP/TN): the net benefit for a true positive (false negative,...)
   #      which is positive for a gain, and negative for a loss
   #  - y: vector of true labels, which has to be labeled as "0" and "1"
   #  - phat: vector of predicted probabilities
   ```

```
 # Outputs:
 #    the function returns the profit curve

 ProfitCurve <- function(benefitTP, benefitFN, benefitFP, benefitTN, y, phat){

if(length(y) != length(phat)) stop("Length of y and phat not identical")
if(length(levels(y))!=2 | levels(y)[1]!="0" | levels(y)[2]!="1")
  stop("y should be a vector of factors, only with levels 0 and 1")

n <- length(y)
df <- data.frame(y, phat)
# Order phat so that we can pick the k highest groups for promotion
df <- df[order(df[,2], decreasing = T),]
TP <- 0; FP <- 0; FN <- table(y)[2]; TN <- table(y)[1]

# Initializing the x and y coordinates of the plot
ratio.vec <- seq(0,n)/n
profit.vec <- rep(0,n+1)
profit.vec[1] <- FN * benefitFN + TN * benefitTN

for(k in 1:n){ # k is the number of groups classified as "YES"
  # In every round, we are picking one more group for promotion.
  # If this group was ratained (positive), then in this round, it is classified
  # as a "YES" instead of "NO" before. The confusion matrix is updated each round
  # with one more TP, and one less FN. It's similar when the group was not ratained.
  if(df[k,1]=="1"){TP <- TP + 1; FN <- FN - 1}
  else{FP <- FP + 1; TN <- TN - 1}
  #print(paste(TP, FP, TP-FP, benefitTP, benefitFP))
  profit.vec[k+1] <- TP*benefitTP + FP*benefitFP + FN*benefitFN + TN*benefitTN
}

plt <- plot(ratio.vec, profit.vec, type="l", lwd=2, col=4, main="Profit Curve",
            xlab="Percentage of Targetted Groups", ylab="Profit")
abline(b=(profit.vec[n+1]-profit.vec[1]), a=profit.vec[1], lty=2) #Random guess
return(plt)
 }
```

5. Develop a decision tree, random forest, and a boosting model using the training data.

   Report ROC, AUC, lift, and profit curves for these models.

   How do these methods compare to the logistic regression models?

6. Investigate whether David can improve performance of his models using data he received from Emily.

   Note that in order to ensure true apples-to-apples comparison, you should use the same split of data into train and test.

   You can load and merge data as follows.

```
STCdata_A <- read.csv('travelData.csv')
STCdata_B <- read.csv('travelData_supplement.csv')
STCdata_merged = merge(STCdata_A, STCdata_B, by = 'ID')
STCdata_merged <- STCdata_merged[,-1]
```

   Remember to fix missing values and combine rare categories.

Comment on the improvement (or lack thereof) from incorporating the NPS data.