Minnesconsin Insurance Company Modeling Problem

Thao Nguyen

December 21, 2017

```r
library(ROCR)
```

```
## Loading required package: gplots

##
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':
##
##     lowess
```

```r
library(stringi)
```

Data Description

The modeling data set was given by Travelers Analytics, based on 4 years of property insurance policies at Minnesconsin Insurance Company from 2013 to 2016. There are approximately 7500 observations and 17 features. (Due to legal matters, the original dataset is not allowed to be shared.)

Setup

```r
col.classes = c("integer", "factor", "integer", "integer", "factor",
                "factor", "numeric", "factor", "factor", "factor", "numeric",
                "factor", "factor", "numeric", "integer", "integer", "integer",
                "integer")
train <- read.csv("train.csv", colClasses = col.classes)
test <- read.csv("test.csv", colClasses = col.classes)
```

Data Pre-processing

Looking at the response variable cancel in the training data, which was expected to be binary, we found several observations to have the value -1, which we decided to delete from the training set with the consensus that we could not impute those values as any pattern found by these observations would not be reliable.

```r
# This functions removes observations with response -1.
remove.weird.response <- function(x) {
  return(subset(x, cancel != -1))
}
```

```r
# This functions reorders columns so numerical, binary, and
# factor data are clustered together.
reorder <- function(x) {
  return(x[, c(18, 17, 15, 1, 3, 4, 7, 11, 14, 2, 6, 5, 8,
               9, 10, 12, 13, 16)])
}
```

For zip code data, we tried clustering them by the first n digits, with n ranging from 1 to 5. However, we found no difference between the different clustering methods. Therefore, in the final model we just chose to cluster zip codes by their first digits.

```r
# This function replaces zip code by its first digit,
# providing a region clustering of zip codes.
cluster.zip <- function(x) {
  x$zip.code <- factor(floor(x$zip.code/10000))
  return(x)
}
```

```r
# This function contains the full data preprocessing pipeline.
my.preprocess = function(train, test) {
  # Remove weird -1 responses.
  train <- remove.weird.response(train)
  # Reorder columns so numeric and factor are next to each other.
  train <- reorder(train)
  test <- reorder(test)
  # Cluster by zip code.
  train <- cluster.zip(train)
  test <- cluster.zip(test)
  return(list(train = train, test = test))
}
```

```r
# Performs preprocessing script on raw data
pp.result = my.preprocess(train, test)
# Training and test sets after preprocessing
data.train <- pp.result$train
data.test <- pp.result$test
data.train = data.train[, 2:18]
```

We simply chose to delete the missing values in the training set, as there didn't seem to be much difference with or without the missing values. We also deleted observations that did not make sense (age > 100).

```r
# Delete NA values in training data
data.train = na.omit(data.train)
# Delete observations with age > 100 in training data
data.train = data.train[data.train$ni.age < 100, ]
```

In the test data, we imputed the missing values by the median for numeric variables, and the mode for categorical variables. In the test data, there was also a 'Landlord' type that did not appear in the training data, so we simply treated it as 'House' like in the training data.

```r
# Impute NAs by median for numeric variables or mode for factors
for (i in 1:16) {
  idx = which(is.na(data.test[, i]))
  if (class(data.test[, i]) == "numeric" | class(data.test[,i]) == "integer") {
    med <- median(data.test[, i], na.rm = T)
    data.test[idx, i] = med
  }
  else if (class(data.test[, i]) == "factor") {
    mod <- labels(which.max(summary(data.test[, i])))
    data.test[idx, i] = mod
  }
}
# Treat 'Landlord' variable in test data as 'House' like in training set
data.test$dwelling.type[data.test$dwelling.type == "Landlord"] = "House"
```

Metrics

We evaluated our models by two criteria: C-statistic, and misclassification error. In order to penalize false positive errors, we assigned a loss of weight 5 to this type of error, and a loss of weight 1 to the other type of error. Under this asymmetric loss function, we decided to pick a cut-off point of 1/6 for the class label prediction to minimize the loss function.

```
# This function calculates misclassification error
misclassification = function(obs, pred) {
  temp = ifelse(obs == pred, 0, ifelse(obs == 1, 5, 1))
  mean(temp)
}
```

For each method, we divided our training data into subsets and performed repeated 10-fold cross-validation to average for the C-stastistic and misclassification error under the asymmetric loss. The final model was chosen as the one that outperformed in both criteria

Logistic Regression

First we selected only main effects variables. We used L0 penalization for variable selection in order to reduce over-fitting and improve the prediction performance. We tuned the regularization parameter using cross-validation with 20% as the training data, repeated 50 times.

```
# This function returns optimal penalization parameter 2:log(n) by CV
L0Selection.tune = function(cv.k = 100, cv.ratio = 0.2, lambda.step = 0.1,
                            lambda.l = 1, lambda.u = 7.2) {
  n.data = nrow(data.train)
  lambda = seq(from = lambda.l, to = lambda.u, by = lambda.step)
  cv.err = numeric(length(lambda))
  cv.auc = numeric(length(lambda))
  for (j in 1:length(lambda)) {
    err = numeric(cv.k)
    cstat = numeric(cv.k)
    for (i in 1:cv.k) {
      train = sample(1:n.data)[1:round(n.data * cv.ratio)]
      logit = glm(cancel ~ factor(claim.ind) + sales.channel +
                    credit + factor(zip.code) + n.adults + n.children +
                    tenure + len.at.res + ni.age + ni.marital.status +
                    premium + ni.gender + house.color + year + sales.channel +
                    coverage.type, data = data.train[train, ], family = binomial(link = "logit"))
      temp = step(logit, direction = "both", trace = 0,
                  k = lambda[j])
      logit = eval(temp$call)
      pred = predict(logit, data.train[-train, ], type = "response")
      input = prediction(pred, data.train[-train, ]$cancel)
      auc = performance(input, "auc")
      cstat[i] = unlist(auc@y.values)
      pred = ifelse(pred > 1/6, 1, 0)
      err[i] = with(data.train[-train, ], misclassification(cancel,pred))
    }
    cv.err[j] = mean(err)
    cv.auc[j] = mean(cstat)
  }
  idx = which.min(cv.err - cv.auc)
```

```r
  return(lambda[idx])
}
```

After tuning for the optimal regularization parameter, we counted the appearances of the predictors, repeated 100 times using cross-validation. We selected the top 10 most frequently appeared variables as our main effects predictors.

```r
# This function returns the counts of appearances of each predictors
L0Selection.count = function(cv.k = 500, cv.ratio = 0.2, lambda = 4) {
  c = numeric(16)
  names(c) = names(data.train)[2:17]
  for (i in 1:cv.k) {
    train = sample(1:n.data)[1:round(n.data * cv.ratio)]
    logit = glm(cancel ~ factor(claim.ind) + sales.channel +
                  credit + factor(zip.code) + n.adults + n.children +
                  tenure + len.at.res + ni.age + ni.marital.status +
                  premium + ni.gender + house.color + year +
                  sales.channel + coverage.type,
                data = data.train[train, ],
                family = binomial(link = "logit"))
    temp = step(logit, direction = "both", trace = 0, k = lambda)
    str = paste(deparse(temp$call), collapse = "")
    str = sub("~", "@", str)
    str = sub(",", "@", str)
    str = strsplit(str, split = "@")[[1]][2]
    str = stri_replace_all_charclass(str, "\\p{WHITE_SPACE}","")
    str = unlist(strsplit(str, split = "[+]"))
    for (j in 1:16) if (names(c)[j] %in% str)
      c[j] = c[j] + 1
  }
  return(c)
}
```

Then we added interaction terms within continuous variables and categorical variables and chose the final model that minimized the difference between the misclassification error and the C-statistic, using cross-validation with 20% as the training data, repeated 100 times.

```r
V0 = c("factor(claim.ind):sales.channel", "factor(claim.ind):credit",
       "factor(claim.ind):factor(zip.code)", "factor(claim.ind):ni.marital.status",
       "sales.channel:credit", "sales.channel:factor(zip.code)",
       "sales.channel:ni.marital.status", "credit:factor(zip.code)",
       "credit:ni.marital.status", "factor(zip.code):ni.marital.status",
       "n.adults:n.children", "n.adults:tenure", "n.adults:len.at.res",
       "n.adults:ni.age", "n.children:tenure", "n.children:len.at.res",
       "n.children:ni.age", "tenure:len.at.res", "tenure:ni.age",
       "len.at.res:ni.age")
M0.fm = "n.adults + n.children +
tenure + sales.channel + credit +
factor(zip.code) + len.at.res + ni.age +
factor(claim.ind) + ni.marital.status"

# This function returns a formula of the best model with two way interactions
InteractSelection = function(V0, M0.fm, M0.err = 0.75, M0.auc = 0.5, cv.k = 500, cv.ratio = 0.2) {
  express = c("glm(cancel ~", ",data = data.train[train,],
              family = binomial)")
  while (TRUE) {
    R = array(NA, dim = c(length(V0), 2))
    for (i in 1:length(V0)) {
      n.data = nrow(data.train)
      err = numeric(cv.k)
      cstat = numeric(cv.k)
      for (j in 1:cv.k) {
        train = sample(1:n.data)[1:round(n.data * cv.ratio)]
        M1 = eval(parse(paste(express[1], "+", M0.fm,
                              V0[i], express[2])))
        pred.p = predict(M1, data.train[-train, ], type = "response")
        input = prediction(pred.p, data.train[-train,]$cancel)
        auc = performance(input, "auc")
        cstat[j] = unlist(auc@y.values)
        pred.c = ifelse(pred.p > 1/6, 1, 0)
        err[j] = with(data.train[-train, ],
                      misclassification(cancel,pred.c))
      }
      M1.err = mean(err)
      M1.auc = mean(cstat)
      if (M1.err < M0.err & M1.auc > M0.auc) {
        R[i, 1] = M1.err
        R[i, 2] = M1.auc
      }
    }
    if (length(na.omit(R[, 1])) > 0) {
      idx = which.min(R[, 1] - R[, 2])
      M0.fm = paste(M0.fm, "+", V0[idx])
      M0.err = R[idx, 1]
      M0.auc = R[idx, 2]
      V0 = V0[-idx]
    } else break
  }
  return(M0.fm)
}
```

```r
err = numeric(100)
cstat = numeric(100)
n.data = nrow(data.train)

# Repeat cross-validation 100 times
for (i in 1:100) {
  train = sample(1:n.data)[1:round(n.data * 0.2)]
  # the best logit model M0
  logit0 = glm(cancel ~ factor(claim.ind) + sales.channel +
                 credit + factor(zip.code) + n.adults * n.children +
                 tenure + len.at.res * ni.age + ni.marital.status,
               data = data.train[train,],
               family = binomial(link = "logit"))
  pred0.p = predict(logit0, data.train[-train, ], type = "response")
  input0 = prediction(pred0.p, data.train[-train, ]$cancel)
  auc0 = performance(input0, "auc")
  cstat[i] = unlist(auc0@y.values)
  pred0.c = ifelse(pred0.p > 1/6, 1, 0)
  err[i] = with(data.train[-train, ], misclassification(cancel,
                                                       pred0.c))
}

mean(err)
```

```
## [1] 0.6225543
```

```r
mean(cstat)
```

```
## [1] 0.7199287
```

Then we used the best performing model to perform prediction on the test data

```r
# Final logistic model
logit = glm(cancel ~ factor(claim.ind) + sales.channel + credit
            + factor(zip.code) + n.adults * n.children
            + tenure + len.at.res * ni.age + ni.marital.status,
            data = data.train, family = binomial(link = "logit"))
# Probability prediction from test data
pred.prob = predict(logit, data.test, type = "response")
# Classification with 1/6 cut-off threshold
pred.class = ifelse(pred.prob > 1/6, 1, 0)
output = list(id = data.test$id, prob = pred.prob, class = pred.class)
write.csv(output, file = "predict.csv")
```