Insurance Customer Churn Prediction Model (Part 2)

Load packages and previous output

```
library(tidyverse)
library(tictoc)
library(pheatmap)
library(moments)
library(impute)
library(pROC)
library(caret)
library(sparklyr)
library(glmnet)
library(tensorflow)
library(keras)

# Load step 1 output
load('dat_train_cleaned_imputed.rdata')
```

Define functions

- To compute prediction measures
- To build multi-layer perceptron (MLP) models

```
caretConfusionMat_F1_Acc <- function(ref, predProb){
  # Use caret::confusionMatrix() and collect output of the confusion matrix,
  # F1-score, and accuracy.
  # Predicted labels are defined by (predProb > 0.5).
  # 'positive' is set to 'TRUE'.

  conf <- confusionMatrix(data = as.factor(predProb > 0.5),
                          reference = as.factor(ref),
                          mode = 'prec_recall',
                          positive = 'TRUE')
  F1score = conf$byClass['F1']
  Acc <- conf$overall['Accuracy']
  out <- list(conf = conf, F1score = F1score, Accuracy = Acc)
}

kerasMlp1 <- function(x_train, y_train){
  # 1 hidden layer with the same number of nodes as the input layer
  numFeat <- ncol(x_train)
  numUnits <- round(numFeat * 1)
  # Create model
  model <- keras_model_sequential()
```

```
  # Define and compile the model
  model %>%
    layer_dense(units = numUnits, activation = 'tanh', input_shape = c(numFeat)) %>%
    layer_dropout(rate = 0.5) %>%
    layer_dense(units = 1, activation = 'sigmoid') %>%
    compile(
      loss = 'binary_crossentropy',
      optimizer = optimizer_sgd(learning_rate = 0.01, momentum = 0.1),
      metrics = c('accuracy')
    )
  print(model)
  # Train
  tic()
  history <- model %>% fit(x_train, y_train, epochs = 20, batch_size = 10,
                           validation_split = 0.2, verbose = 2)
  toc(log=T)
  # Evaluate for train data
  (kerasMlpEval <- model %>% evaluate(x_train, y_train))
  fittedvalues <- model %>% predict(x_train)
  (trainAUC <- auc(y_train, c(fittedvalues)))
  out <- list(model=model, history=history, kerasMlpEval=kerasMlpEval,
              fittedvalues=fittedvalues, trainAUC=trainAUC)
  return(out)
}

kerasMlp2 <- function(x_train, y_train){
  # 2 hidden layers where the number of nodes is 3 * (num of input nodes)
  numFeat <- ncol(x_train)
  numUnits <- round(numFeat * 3)
  # Create model
  model <- keras_model_sequential()
  # Define and compile the model
  model %>%
    layer_dense(units = numUnits, activation = 'tanh', input_shape = c(numFeat)) %>%
    layer_dropout(rate = 0.5) %>%
    layer_dense(units = numUnits, activation = 'tanh') %>%
    layer_dropout(rate = 0.5) %>%
    layer_dense(units = 1, activation = 'sigmoid') %>%
    compile(
      loss = 'binary_crossentropy',
      optimizer = optimizer_sgd(learning_rate = 0.01, momentum = 0.1),
      metrics = c('accuracy')
    )
  print(model)
  # Train
  tic()
  history <- model %>% fit(x_train, y_train, epochs = 20, batch_size = 10,
                           validation_split = 0.2, verbose = 2)
  toc(log=T)
  # Evaluate for train data
  (kerasMlpEval <- model %>% evaluate(x_train, y_train))
  fittedvalues <- model %>% predict(x_train)
  (trainAUC <- auc(y_train, c(fittedvalues)))
```

```
  out <- list(model=model, history=history, kerasMlpEval=kerasMlpEval,
              fittedvalues=fittedvalues, trainAUC=trainAUC)
  return(out)
}
```

Step 2 - Build up churn models

2-1. Preliminary logistic regression with all the features (model name: lr_full)

- This initial regression with 95 features is intended for sanity check for regression, checking if the above data cleaning is working well, identifying informative features for the response, and designing modeling strategies.

```
# Split data into train and validation datasets
set.seed(0325)
validation_set_prop <- 0.25
val_set_size <- round(nrow(dat_train_sc) * validation_set_prop)
val_set_ids <- sample.int(nrow(dat_train_sc), val_set_size)
dat_train_sc_train <- dat_train_sc[-val_set_ids, ]        # for model training
dat_train_sc_val <- dat_train_sc[val_set_ids, ]           # for model evaluation and
                                                          # tuning hyper parameters
# Define features and model formulas
features <- setdiff(colnames(dat_train_sc), "y")
features
formula1 <- as.formula(paste0('y ~ ', paste0(features, collapse = ' + ')))
formula1

# Fit glm()
lr_full <- glm(formula1, data = dat_train_sc_train, family = binomial(link = 'logit'))
summary(lr_full)
indSig <- (coef(summary(lr_full))[, 4] < 0.05)
(sigFeatures <- rownames(coef(summary(lr_full)))[indSig])
# Significant features shown (P<0.05)

# Prediction performance
pred <- predict(lr_full, newdata = dat_train_sc_val, type = 'response')
(lr_full$AUC <- auc(dat_train_sc_val$y, pred))
# Area under the curve: 0.7595
out <- caretConfusionMat_F1_Acc((dat_train_sc_val$y == 1), pred)
out$conf
(lr_full$Accuracy <- out$Accuracy)
```

- Prediction AUC of 76% is not bad. This AUC will be considered as a baseline.
- But, recall is 10% which may need some improvement if possible.

In the 1st round of modeling

- Using a logistic regression with LASSO method (glmnet()), I selected candidate sets of features, which were fed into the logistic regressions, random forests, decision trees, and multilayer perceptrons in order to find an optimal model with the best AUC.
- As a result, they didn't significantly improve the baseline AUC (76%).

In the 2nd round of modeling

- I considered interaction effects between continuous features and categorical ones. But, the inclusion of some interaction effects
  didn't improve the baseline AUC.

The 3rd modeling plan

- Now I considered nonlinear effects of continuous features on the response.
- By including squared continuous features, the AUC was improved as shown below in a reference full model.
- So the plan is
  - to include squared continuous terms
  - to use logistic regression with LASSO for selection of multiple feature sets,
  - and to apply various prediction methods in order to obtained an improved prediction outcome.

2-2. Logistic regression after including squared continuous features (model name: lr_full2)

```
# Make a new data frame
df_numFeatures_squared <- (as.data.frame(df_numFeatures_sc_imputed))^2
colnames0 <- paste0(colnames(df_numFeatures_squared), "_squared")
colnames(df_numFeatures_squared) <- colnames0
df_numFeatures_squared <- as_tibble(df_numFeatures_squared)
dat_train_sc2 <- bind_cols(df_y, df_numFeatures_sc_imputed, df_numFeatures_squared, df_categorical)

dat_train_sc2_train <- dat_train_sc2[-val_set_ids, ]      # for model training
dat_train_sc2_val <- dat_train_sc2[val_set_ids, ]         # for model evaluation and
                                                          # tuning hyper parameters

# Set features and model formulas
features2 <- setdiff(colnames(dat_train_sc2), "y")
features2
formula2 <- as.formula(paste0('y ~ ', paste0(features2, collapse = ' + ')))
formula2

# Fit glm()
lr_full2 <- glm(formula2, data = dat_train_sc2_train, family = binomial(link = 'logit'))
summary(lr_full2)
indSig <- (coef(summary(lr_full2))[, 4] < 0.05)
(sigFeatures <- rownames(coef(summary(lr_full2)))[indSig])
# Significant features shown (P<0.05)
```

- Area under the curve: 0.8044 (~5% improved compared to the baseline 76%)
- The validation AUC increased significantly by adding the quadratic terms.

```
# Prediction performance
pred <- predict(lr_full2, newdata = dat_train_sc2_val, type = 'response')
(lr_full2$AUC <- auc(dat_train_sc2_val$y, pred))
```

```
## Area under the curve: 0.8044
```

- Recall is 21%, which is the double of the baseline recall (10%).

```
out <- caretConfusionMat_F1_Acc((dat_train_sc_val$y == 1), pred)
out$conf
```

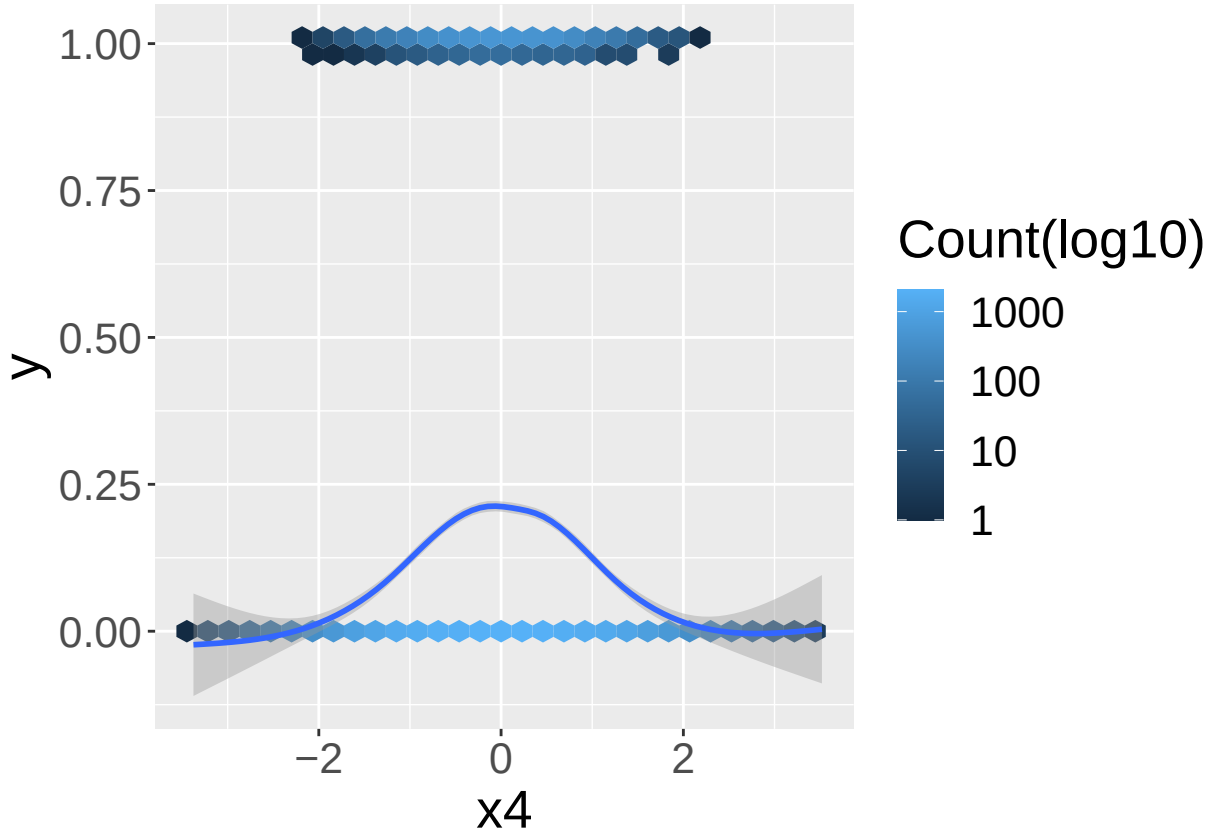
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE  8348 1153
##      TRUE    199  300
##
##                Accuracy : 0.8648
##                  95% CI : (0.8579, 0.8714)
##     No Information Rate : 0.8547
##     P-Value [Acc > NIR] : 0.00201
##
##                   Kappa : 0.2518
##
##  Mcnemar's Test P-Value : < 2e-16
##
##               Precision : 0.6012
##                  Recall : 0.2065
##                      F1 : 0.3074
##              Prevalence : 0.1453
##          Detection Rate : 0.0300
##    Detection Prevalence : 0.0499
##       Balanced Accuracy : 0.5916
##
##        'Positive' Class : TRUE
##
```

```
(lr_full2$Accuracy <- out$Accuracy)
```

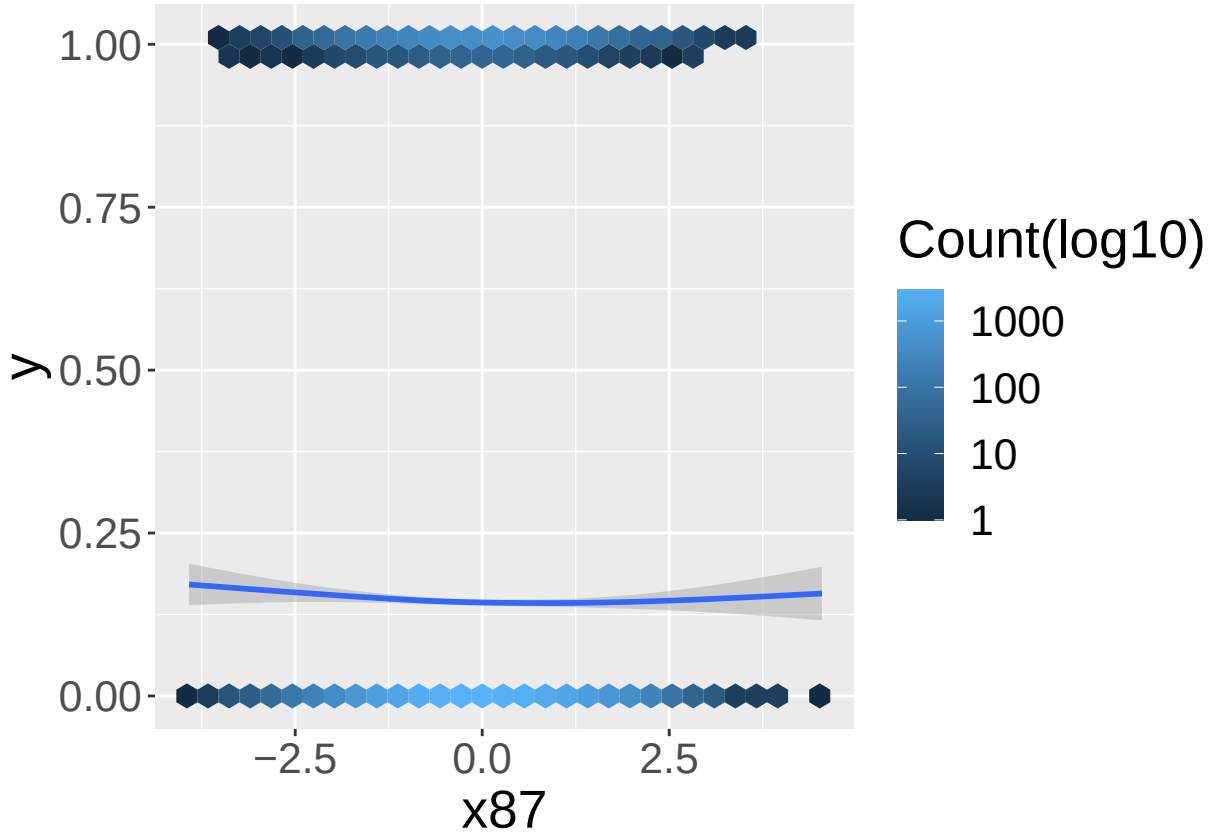
```
## Accuracy
##   0.8648
```

- In this fitting, the following 4 squared terms became significant.
  - "x4_squared" "x18_squared" "x40_squared" "x87_squared"
- Visualize the detected non-linear effects
- 'x4' shows a clear non-linear effect. The higher 'x4' is the lower 'y' is.

```
dat_train_sc2_train %>% ggplot(aes(x=x4, y=y)) +
  geom_hex() +
  scale_fill_gradient(name = 'Count(log10)', trans = "log10") +
  geom_smooth() +
  theme(text = element_text(size = 20))
```




```
dat_train_sc2_train %>% ggplot(aes(x=x87, y=y)) +
  geom_hex() +
  scale_fill_gradient(name = 'Count(log10)', trans = "log10") +
  geom_smooth() +
  theme(text = element_text(size = 20))
```



2-3. Logistic regression with LASSO penalty for feature selection (model name: lr_LASSO)

- Using this LASSO method, I will select candidate sets of features, which will be fed into the logistic regressions, random forests, decision trees, and multilayer perceptrons in order to find an optimal model with the best AUC.
- Parse categorical features into indicator features and create a design matrix for the regression

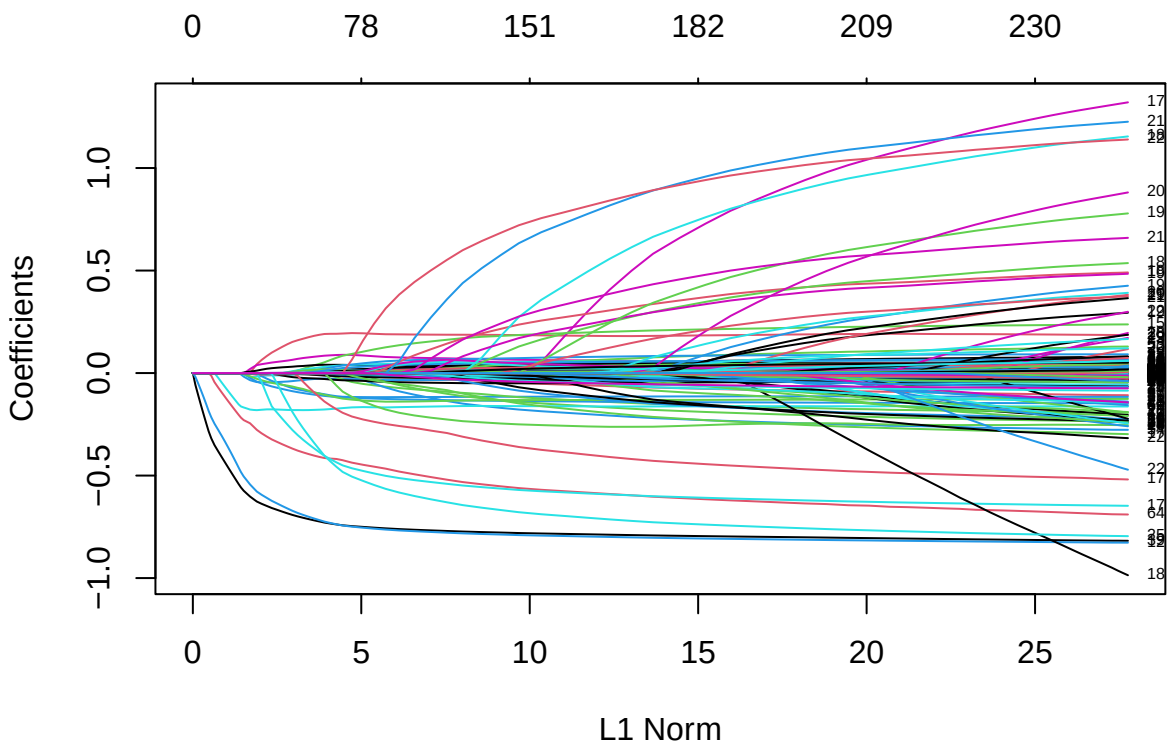
```
X <- model.matrix(formula2, data = dat_train_sc2)[, -1]
y <- dat_train_sc2$y

lr_LASSO <- glmnet(X, y, family = 'binomial', standardize = FALSE)
lr_LASSO
```

```
##
## Call:  glmnet(x = X, y = y, family = "binomial", standardize = FALSE)
##
##      Df  %Dev   Lambda
## 1    0  0.00 0.080800
## 2    2  1.42 0.073620
## 3    2  2.91 0.067080
## 4    2  4.18 0.061120
## 5    2  5.28 0.055690
## 6    2  6.23 0.050740
## 7    2  7.05 0.046240
```

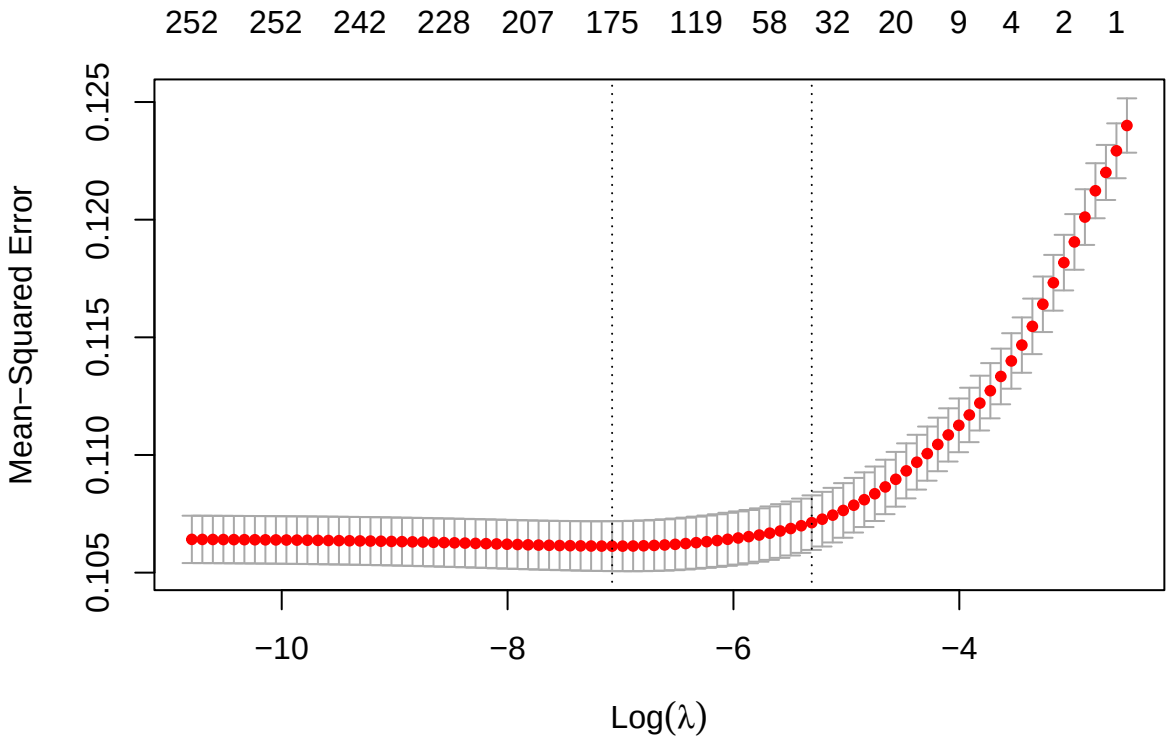
```
## 8    3  7.90 0.042130
## 9    3  8.81 0.038390
## 10   4  9.78 0.034980
## 11   4 10.62 0.031870
## 12   4 11.34 0.029040
## 13   4 11.96 0.026460
## 14   4 12.48 0.024110
## 15   4 12.93 0.021970
## 16   4 13.32 0.020010
## 17   5 13.64 0.018240
## 18   6 13.94 0.016620
## 19  10 14.25 0.015140
## 20  12 14.58 0.013800
## 21  13 14.87 0.012570
## 22  13 15.17 0.011450
## 23  16 15.43 0.010440
## 24  17 15.78 0.009509
## 25  17 16.11 0.008664
## 26  19 16.41 0.007894
## 27  19 16.70 0.007193
## 28  20 16.96 0.006554
## 29  24 17.19 0.005972
## 30  25 17.39 0.005441
## 31  26 17.58 0.004958
## 32  28 17.77 0.004517
## 33  31 17.94 0.004116
## 34  34 18.09 0.003750
## 35  38 18.23 0.003417
## 36  40 18.35 0.003114
## 37  47 18.47 0.002837
## 38  51 18.59 0.002585
## 39  58 18.70 0.002355
## 40  71 18.85 0.002146
## 41  78 19.01 0.001955
## 42  88 19.16 0.001782
## 43  94 19.30 0.001623
## 44 103 19.45 0.001479
## 45 108 19.61 0.001348
## 46 113 19.76 0.001228
## 47 114 19.90 0.001119
## 48 120 20.02 0.001020
## 49 130 20.15 0.000929
## 50 133 20.27 0.000847
## 51 141 20.37 0.000771
## 52 151 20.49 0.000703
## 53 153 20.61 0.000640
## 54 154 20.71 0.000583
## 55 160 20.80 0.000532
## 56 171 20.88 0.000484
## 57 174 20.95 0.000441
## 58 178 21.03 0.000402
## 59 182 21.10 0.000366
## 60 186 21.16 0.000334
## 61 192 21.22 0.000304
```

```
## 62 198 21.27 0.000277
## 63 200 21.32 0.000253
## 64 205 21.36 0.000230
## 65 204 21.39 0.000210
## 66 209 21.42 0.000191
## 67 210 21.45 0.000174
## 68 213 21.47 0.000159
## 69 214 21.49 0.000144
## 70 215 21.51 0.000132
## 71 218 21.52 0.000120
## 72 223 21.53 0.000109
## 73 225 21.55 0.000100
## 74 227 21.56 0.000091
## 75 228 21.56 0.000083
## 76 229 21.57 0.000075
## 77 230 21.58 0.000069
## 78 233 21.58 0.000063
## 79 233 21.59 0.000057
## 80 235 21.59 0.000052
## 81 237 21.59 0.000047
## 82 238 21.60 0.000043
## 83 238 21.60 0.000039
## 84 238 21.60 0.000036
## 85 238 21.60 0.000033
## 86 242 21.60 0.000030
## 87 243 21.60 0.000027
## 88 245 21.61 0.000025
```

```
plot(lr_LASSO, label = TRUE)
```




```
set.seed(0325)
cv.fit <- cv.glmnet(X, y, nfolds = 4)
plot(cv.fit)
```




```
log(cv.fit$lambda.min)
```

```
## [1] -7.074425
```

```
# [1] -7.074425
log(cv.fit$lambda.1se)
```

```
## [1] -5.306784
```

```
# [1] -5.306784
```

- Recommend sparsity with lambdas is between (lambda.min, lambda.1se).
- The features given by lambda.1se is typically considered as the most regularized model.
- To consider 4 candidate subsets of features, I choose the following feature sets which are in an increasing order of size.
  - (1) features given by log(lambda.1se) = -5.31
  - (2) features given by log(lambda) = average of log(lambda.1se) and log(lambda.min)
  - (3) features given by log(lambda.min) = -7.07
  - (4) features given by log(lambda) = -8.5
- Therefore, by including the full model, total 5 feature sets will be considered across various prediction models seeking for the optimal model with the best AUC.

2-4. Extract candidate feature sets from the glm-LASSO fit

```
# Prepare variables
lambdas <- list()
coeffs <- list()
sigGlmFeatures <- list()
sigOrigFeatures <- list()

# Lambdas were chosen from the output of the above cross-validation using glmnet()
lambdas[[1]] <- cv.fit$lambda.1se
lambdas[[2]] <- exp( (log(cv.fit$lambda.1se) + log(cv.fit$lambda.min))/2 )
lambdas[[3]] <- cv.fit$lambda.min
lambdas[[4]] <- exp(-8.5)

# Lambdas are matched to the corresponding features with non-zero coefficients
for (k in 1:4){
  coeffs[[k]] <- coef(lr_LASSO, s = lambdas[[k]])
  sigGlmFeatures[[k]] <- coeffs[[k]]@Dimnames[[1]][coeffs[[k]]@i + 1][-1]
  print(paste0('#### k = ', k))
  print(sigGlmFeatures[[k]])
}

# Parsed binary indicators are pooled back to original categorical features
(sigOrigFeatures[[1]] <- c(sigGlmFeatures[[1]][1:23], 'x3', 'x31', 'x93'))
(sigOrigFeatures[[2]] <- c(sigGlmFeatures[[2]][1:70], 'x3', 'x31', 'x33', 'x93'))
(sigOrigFeatures[[3]] <-
    c(sigGlmFeatures[[3]][1:112], 'x3', 'x31', 'x33', 'x60', 'x65', 'x93', 'x98'))
(sigOrigFeatures[[4]] <-
    c(sigGlmFeatures[[4]][1:148], 'x3', 'x31', 'x33', 'x60', 'x65', 'x77', 'x93', 'x98'))
(sigOrigFeatures[[5]] <- c(colnames(dat_train_sc2)[-1]))

# The number of glm features (with parsed binaries) across all 5 candidate features,
# including the full model
c(as.numeric(lapply(sigGlmFeatures, length)), ncol(X))
# [1]  26  78 133 209 255

# The number of original features across all 5 candidate features, including
# the full model
c(as.numeric(lapply(sigOrigFeatures, length)))
# [1]  26  74 119 156 179

# Save
save(lambdas, coeffs, sigGlmFeatures, sigOrigFeatures,
     file = 'glmLASSO_selectedFeatSets_includeSquared.rdata')
```

2-5. Logistic regressions with the chosen 5 feature sets and their prediction performances (model names: lr_models)

```
mdl_formulas <- list()
lr_models <- list()
lr_AUC <- list()
```

```
lr_Accuracy <- list()

# Specify model formulas
for (k in 1:length(sigOrigFeatures)){
  mdl_formulas[[k]] <- as.formula(
    paste0('y ~ ', paste0(sigOrigFeatures[[k]], collapse = ' + ')))
}

# Fit glm() and compute prediction measures for validation data
for (k in 1:length(sigOrigFeatures)){
  lr_models[[k]] <- glm(mdl_formulas[[k]], data = dat_train_sc2_train,
                        family = binomial(link = 'logit'))
  print(summary(lr_models[[k]]))
  pred <- predict(lr_models[[k]], newdata = dat_train_sc2_val, type = 'response')
  (lr_models[[k]]$AUC <- auc(dat_train_sc2_val$y, pred))
  cat(paste0("\n##  k = ", k, ", Prediction AUC: ", round(lr_models[[k]]$AUC, 4)), "\n")
  out <- caretConfusionMat_F1_Acc((dat_train_sc2_val$y == 1), pred)
  print(out$conf)
  (lr_models[[k]]$Accuracy <- out$Accuracy)
  cat(paste0("\n##  k = ", k, ", Prediction Accuracy: ",
             round(lr_models[[k]]$Accuracy, 4)), "\n")
  lr_AUC[[k]] <- lr_models[[k]]$AUC
  lr_Accuracy[[k]] <- lr_models[[k]]$Accuracy
}
```

- Validation measures

```
print(lr_AUC)                                  # (max AUC) k = 2, 0.8093
```

```
## [[1]]
## Area under the curve: 0.7986
##
## [[2]]
## Area under the curve: 0.8093
##
## [[3]]
## Area under the curve: 0.8076
##
## [[4]]
## Area under the curve: 0.8054
##
## [[5]]
## Area under the curve: 0.8044
```

```
print(lr_Accuracy)                             # (max Accuracy) k = 2, 0.8685
```

```
## [[1]]
## Accuracy
##   0.8631
##
## [[2]]
## Accuracy
```

```
##   0.8685
##
## [[3]]
## Accuracy
##   0.8671
##
## [[4]]
## Accuracy
##   0.8649
##
## [[5]]
## Accuracy
##   0.8648
```

- The optimal feature set (k = 2)

```
print(length(sigOrigFeatures[[2]]))
print(sigOrigFeatures[[2]])

# Among 5 considered feature sets, the following set is selected by GLM.
#[1] "x11"           "x14"           "x16"           "x26"           "x41"
#[6] "x42"           "x52"           "x54"           "x61"           "x67"
#[11] "x68"           "x75"           "x83"           "x89"           "x96"
#[16] "x1"            "x7"            "x9"            "x15"           "x18"
#[21] "x19"           "x28"           "x36"           "x40"           "x46"
#[26] "x47"           "x48"           "x51"           "x56"           "x62"
#[31] "x66"           "x81"           "x97"           "x100"          "x5_squared"
#[36] "x16_squared"   "x22_squared"   "x38_squared"   "x45_squared"   "x55_squared"
#[41] "x63_squared"   "x64_squared"   "x74_squared"   "x75_squared"   "x80_squared"
#[46] "x83_squared"   "x85_squared"   "x89_squared"   "x94_squared"   "x4_squared"
#[51] "x8_squared"    "x18_squared"   "x19_squared"   "x21_squared"   "x32_squared"
#[56] "x34_squared"   "x37_squared"   "x40_squared"   "x43_squared"   "x46_squared"
#[61] "x47_squared"   "x50_squared"   "x53_squared"   "x69_squared"   "x71_squared"
#[66] "x72_squared"   "x73_squared"   "x82_squared"   "x84_squared"   "x100_squared"
#[71] "x3"            "x31"           "x33"           "x93"

# Save
save(lr_models, lr_AUC, lr_Accuracy,
     file = "lr_models_AUC_Acc.rdata")
```

2-6. Random forests with the chosen 5 feature sets and their prediction performances (model names: rf_models)

```
rf_models <- list()
mdl_formulas_wDummy <- list()
rf_Accuracy <- list()

# Because of an error in parsed model formula, 'x33' (state) variable's factor levels
# are re-coded by removing a space in the state names.
dat_train_sc2_x33recoded <- dat_train_sc2
x33 <- dat_train_sc2_x33recoded$x33
```

```
level0 <- levels(x33)
level1 <- level0
for (i in 1:length(level0)){
  level1[i] <- paste0(unlist(strsplit(level0[i], split=" ")), collapse="")
}
levels(x33) <- level1
dat_train_sc2_x33recoded$x33 <- x33

# Generate a new df's where categorical features are parsed into binary features
X_x33recoded <- model.matrix(formula2, data = dat_train_sc2_x33recoded)[, -1]
dat_train_sc2_dummy <- as_tibble(cbind(y = dat_train_sc2_x33recoded$y, X_x33recoded))
dat_train_sc2_dummy_train <- dat_train_sc2_dummy[-val_set_ids, ]
dat_train_sc2_dummy_val <- dat_train_sc2_dummy[val_set_ids, ]
```

- Using Sparklyr (v3.3.1)

```
sc <- spark_connect(master = 'local')
dat_train_sc2_dummy_train_tbl <-
  copy_to(sc, dat_train_sc2_dummy_train, 'dat_train_sc2_dummy_train_tbl', overwrite = T)
dat_train_sc2_dummy_val_tbl <-
  copy_to(sc, dat_train_sc2_dummy_val, 'dat_train_sc2_dummy_val_tbl', overwrite = T)

# Generate new model formulas according to the new df's with the recoded x33
for (k in 1:length(mdl_formulas)){
  tmpMdlMat <- model.matrix(mdl_formulas[[k]], data = dat_train_sc2_x33recoded)[, -1]
  mdl_formulas_wDummy[[k]] <-
    as.formula(paste0('y ~ ', paste0(cbind(colnames(tmpMdlMat)), collapse = ' + ')))
}

# Fit ml_random_forest() and compute prediction measures for validation data
for (k in 1:length(mdl_formulas_wDummy)){
  rf_models[[k]] <- ml_random_forest(
    dat_train_sc2_dummy_train_tbl, mdl_formulas_wDummy[[k]], type = "classification")
  print(rf_models[[k]])
  mlEval <- ml_evaluate(rf_models[[k]], dat_train_sc2_dummy_val_tbl)   # Val Accuracy
  rf_Accuracy[[k]] <- mlEval
}
```

- rf_Accuracies are ~0.788, which are lower than lr_full2 model (~0.86).

```
print(rf_Accuracy)
```

```
## [[1]]
## # A tibble: 1 x 1
##   Accuracy
##      <dbl>
## 1    0.788
##
## [[2]]
## # A tibble: 1 x 1
##   Accuracy
##      <dbl>
```

```
## 1    0.788
##
## [[3]]
## # A tibble: 1 x 1
##   Accuracy
##      <dbl>
## 1    0.788
##
## [[4]]
## # A tibble: 1 x 1
##   Accuracy
##      <dbl>
## 1    0.788
##
## [[5]]
## # A tibble: 1 x 1
##   Accuracy
##      <dbl>
## 1    0.788
```

2-7. Decision trees with the chosen 5 feature sets and their prediction performances (model names: dt_models)

```
dt_models <- list()
dt_Accuracy <- list()

# Fit ml_decision_tree() and compute prediction measures for validation data
for (k in 1:length(mdl_formulas_wDummy)){
  dt_models[[k]] <- ml_decision_tree(dat_train_sc2_dummy_train_tbl, mdl_formulas_wDummy[[k]],
                                     type = "classification")
  print(dt_models[[k]])
  mlEval <- ml_evaluate(dt_models[[k]], dat_train_sc2_dummy_val_tbl)   # Val Accuracy
  dt_Accuracy[[k]] <- mlEval
}
```

- dt_Accuracies are <0.809, which are lower than lr_full2 model (~0.86).

```
print(dt_Accuracy)
```

```
## [[1]]
## # A tibble: 1 x 1
##   Accuracy
##      <dbl>
## 1    0.808
##
## [[2]]
## # A tibble: 1 x 1
##   Accuracy
##      <dbl>
## 1    0.809
##
```

```
## [[3]]
## # A tibble: 1 x 1
##   Accuracy
##      <dbl>
## 1    0.809
##
## [[4]]
## # A tibble: 1 x 1
##   Accuracy
##      <dbl>
## 1    0.800
##
## [[5]]
## # A tibble: 1 x 1
##   Accuracy
##      <dbl>
## 1    0.800
```

2-8. Multi-layer Perceptron (MLP) with the chosen 5 feature sets and their prediction performances (model names: mlp1_models, mlp2_models)

- For model specifications, I chose to run two MLP models.
- The first has 1-hidden layer with the same number of nodes as the input layer.
- The second has 2-hidden layers where the layer size is 3*(the number of input layer), which can hopefully approximate non-linear relations between X and y if there is any.
- Through preliminary fitting with MLPs, I selected a set of tuning parameters (learning rate, momentum, epochs, batch_size, and dropout rate).
- Their values are specified in defined functions of kerasMlp1() and kerasMlp2().

- Fit kerasMlp1(), and compute prediction measures

```
mlp1_models <- list()
mlp1_AUC <- list()
mlp1_Accuracy <- list()
mlp2_models <- list()
mlp2_AUC <- list()
mlp2_Accuracy <- list()

for (k in 1:length(mdl_formulas)){
  x_train <- model.matrix(mdl_formulas[[k]], data = dat_train_sc2_train)[, -1]
  y_train <- dat_train_sc2_train$y
  mlp1_models[[k]] <- kerasMlp1(x_train, y_train)
  # Validation data
  x_val <- model.matrix(mdl_formulas[[k]], data = dat_train_sc2_val)[, -1]
  y_val <- dat_train_sc2_val$y
  pred <- mlp1_models[[k]]$model %>% predict(x_val)
  (mlp1_AUC[[k]] <- auc(y_val, c(pred)))
  cat(paste0("\n##  k = ", k, ", Prediction AUC: ", round(mlp1_AUC[[k]], 4)), "\n")
```

```
  out <- caretConfusionMat_F1_Acc((dat_train_sc2_val$y == 1), c(pred))
  print(out$conf)
  mlp1_Accuracy[[k]] <- out$Accuracy
  cat(paste0("\n##  k = ", k, ", Prediction Accuracy: ",
             round(mlp1_Accuracy[[k]], 4)), "\n")
}
```

- Maximum AUC = 0.7978 (k=2), which is lower than lr_AUC (0.8093).
- Maximum Accuracy = 0.8635 (k=2), which is lower than lr_Accuracy (0.8685).

```
# Prediction measures
print(mlp1_AUC)
```

```
## [[1]]
## Area under the curve: 0.7964
##
## [[2]]
## Area under the curve: 0.7982
##
## [[3]]
## Area under the curve: 0.7947
##
## [[4]]
## Area under the curve: 0.7915
##
## [[5]]
## Area under the curve: 0.7916
```

```
print(mlp1_Accuracy)
```

```
## [[1]]
## Accuracy
##   0.8635
##
## [[2]]
## Accuracy
##   0.8638
##
## [[3]]
## Accuracy
##   0.8595
##
## [[4]]
## Accuracy
##   0.8617
##
## [[5]]
## Accuracy
##   0.8629
```

- Fit kerasMlp2(), and compute prediction measures

```r
for (k in 1:length(mdl_formulas)){
  x_train <- model.matrix(mdl_formulas[[k]], data = dat_train_sc2_train)[, -1]
  y_train <- dat_train_sc2_train$y
  mlp2_models[[k]] <- kerasMlp2(x_train, y_train)
  # Validation data
  x_val <- model.matrix(mdl_formulas[[k]], data = dat_train_sc2_val)[, -1]
  y_val <- dat_train_sc2_val$y
  pred <- mlp2_models[[k]]$model %>% predict(x_val)
  (mlp2_AUC[[k]] <- auc(y_val, c(pred)))
  cat(paste0("\n##  k = ", k, ", Prediction AUC: ", round(mlp2_AUC[[k]], 4)), "\n")
  out <- caretConfusionMat_F1_Acc((dat_train_sc2_val$y == 1), c(pred))
  print(out$conf)
  mlp2_Accuracy[[k]] <- out$Accuracy
  cat(paste0("\n##  k = ", k, ", Prediction Accuracy: ",
             round(mlp2_Accuracy[[k]], 4)), "\n")
}
```

- Maximum AUC = 0.7994 (k=2), which is close to but lower than lr_AUC (0.8093).
- Maximum Accuracy = 0.8635 (k=1), which is close to but lower than lr_Accuracy (0.8685).

```r
# Prediction measures
print(mlp2_AUC)
```

```
## [[1]]
## Area under the curve: 0.796
##
## [[2]]
## Area under the curve: 0.7987
##
## [[3]]
## Area under the curve: 0.7958
##
## [[4]]
## Area under the curve: 0.7889
##
## [[5]]
## Area under the curve: 0.7895
```

```r
print(mlp2_Accuracy)
```

```
## [[1]]
## Accuracy
##   0.8644
##
## [[2]]
## Accuracy
##    0.863
##
## [[3]]
## Accuracy
##   0.8615
##
```

```
## [[4]]
## Accuracy
##   0.8559
##
## [[5]]
## Accuracy
##   0.8601
```

```
# Save
save(mlp1_models, mlp1_AUC, mlp1_Accuracy, mlp2_models, mlp2_AUC, mlp2_Accuracy,
     file = "MLP_models_AUC_Acc.rdata")
save_model_hdf5(mlp1_models[[2]]$model, 'mlp1_model_k2.h5')
save_model_hdf5(mlp2_models[[2]]$model, 'mlp2_model_k2.h5')
```

- As an optimal MLP model, I select the 1-hidden layer model with k=2 (mlp1_models[[2]]).
- Although a 2-hidden layer model showed a better AUC (0.7994), the difference from the AUC of mlp1_models[[2]] (0.7978) is negligible.
  Thus, a more parsimonious model is preferred.
- Just for record, the optimal MLP model (mlp1_models with k=2 feature set) is defined and trained as follows.

```
k = 2

x_train <- model.matrix(mdl_formulas[[k]], data = dat_train_sc2_train)[, -1]
y_train <- dat_train_sc2_train$y
mlp1_models[[k]] <- kerasMlp1(x_train, y_train)
```

```
## Model: "sequential_10"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  dense_26 (Dense)                   (None, 129)                     16770
##  dropout_15 (Dropout)               (None, 129)                     0
##  dense_25 (Dense)                   (None, 1)                       130
## =========================================================================
## Total params: 16,900
## Trainable params: 16,900
## Non-trainable params: 0
## _________________________________________________________________________
## 65.31 sec elapsed
```

```
# Validation data
x_val <- model.matrix(mdl_formulas[[k]], data = dat_train_sc2_val)[, -1]
y_val <- dat_train_sc2_val$y
pred <- mlp1_models[[k]]$model %>% predict(x_val)
(mlp1_AUC[[k]] <- auc(y_val, c(pred)))
```

```
## Area under the curve: 0.7981
```

```
cat(paste0("\n##  k = ", k, ", Prediction AUC: ", round(mlp1_AUC[[k]], 4)), "\n")
```

```
##
## ##  k = 2, Prediction AUC: 0.7981
```

```
out <- caretConfusionMat_F1_Acc((dat_train_sc2_val$y == 1), c(pred))
print(out$conf)
```

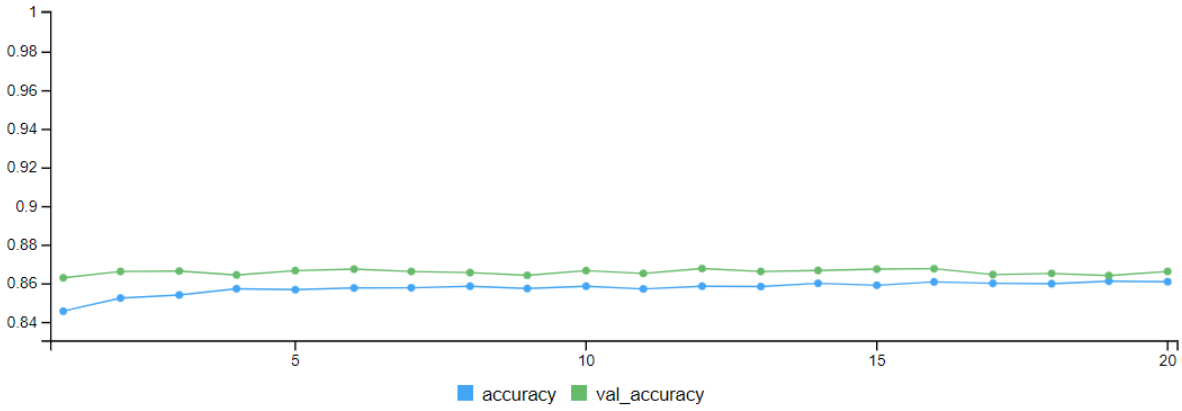
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE  8453 1271
##      TRUE     94  182
##
##                Accuracy : 0.8635
##                  95% CI : (0.8566, 0.8702)
##     No Information Rate : 0.8547
##     P-Value [Acc > NIR] : 0.006196
##
##                   Kappa : 0.1721
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##               Precision : 0.6594
##                  Recall : 0.1253
##                      F1 : 0.2105
##              Prevalence : 0.1453
##          Detection Rate : 0.0182
##    Detection Prevalence : 0.0276
##       Balanced Accuracy : 0.5571
##
##        'Positive' Class : TRUE
##
```

```
mlp1_Accuracy[[k]] <- out$Accuracy
cat(paste0("\n##  k = ", k, ", Prediction Accuracy: ",
           round(mlp1_Accuracy[[k]], 4)), "\n")
```

```
##
## ##  k = 2, Prediction Accuracy: 0.8635
```

- The learning curve


0.4
0.39
0.38
0.37
0.36
0.35
0.34
0.33
5
10
15
20
loss
val_loss


1
0.98
0.96
0.94
0.92
0.9
0.88
0.86
0.84
5
10
15
20
accuracy
val_accuracy