

Classification: Data Preparation

CITS4009 Computational Data Analysis

Unit Coordinator: Dr Du Huynh

Department of Computer Science and Software Engineering
The University of Western Australia

Semester 2, 2022

KDD Cup 2009 Data

The KDD Cup 2009 provided a dataset about customer relationship management.

- The contest supplied 230 facts (features) about 50,000 credit card accounts.
- The goal was to predict
 - account cancellation (called *churn*),
 - the innate tendency to use new products and services (called *appetency*), and
 - willingness to respond favorably to marketing pitches (called *upselling*)

Data Available at:

<https://github.com/WinVector/zmPDSwR>

Why Three-way splitting

This problem has

- a large number of variables, many of which have a large number of possible levels.
- The competition's **AUC** (*area under curve*) measure, isn't particularly resistant to *overfitting* (not having built-in model complexity or chance corrections).

Because of this concern, we'll split our data into three sets: training, calibration, and test.

- Use the training set for most of our work, and we'll never look at the test set (we'll reserve it for our final report of model performance).
- The calibration set is used to simulate the unseen test set during modeling-
 - performance on the calibration set is used to estimate overfitting.

KDD2009 Data Pre-processing

```
path <- '../..'/data_v2/KDD2009/'
d <- read.table(paste0(path, 'orange_small_train.data.gz'),
               header=T, sep='\t', na.strings=c('NA',''))

churn <- read.table(
  paste0(path, 'orange_small_train_churn.labels.txt'),
  header=F, sep='\t')
d$churn <- churn$V1           # ___churn___

appetency <- read.table(
  paste0(path, 'orange_small_train_appetency.labels.txt'),
  header=F, sep='\t')
d$appetency <- appetency$V1   # ___appetency___

upselling <- read.table(
  paste0(path, 'orange_small_train_upselling.labels.txt'),
  header=F, sep='\t')
d$upselling <- upselling$V1   # ___upselling___

# d - data frame having 5000 rows and 233 (=230+3) columns;
# churn, appetency, upselling - all having 5000 rows and 1 column.
```

KDD2009 Data Pre-processing (Cont.)

```
# do a 90/10 split to form the training and test sets.
set.seed(729375)
d$rgroup <- runif(dim(d)[1])
dTrainAll <- subset(d, rgroup<=0.9)
dTest <- subset(d, rgroup>0.9)
outcomes <- c('churn', 'appetency', 'upselling')
# names of columns that are categorical type and numerical type
vars <- setdiff(colnames(dTrainAll), c(outcomes, 'rgroup'))
catVars <- vars[sapply(dTrainAll[, vars], class) %in%
                  c('factor', 'character')]
numericVars <- vars[sapply(dTrainAll[, vars], class) %in%
                  c('numeric', 'integer')]
# remove the original tables
rm(list=c('d', 'churn', 'appetency', 'upselling'))

# split dTrainAll into a training set and a validation (or calibration) set
useForCal <- rbinom(n=dim(dTrainAll)[1], size=1, prob=0.1)>0
dCal <- subset(dTrainAll, useForCal)
dTrain <- subset(dTrainAll, !useForCal)
```

References

- **Practical Data Science with R**, *Nina Zumel, John Mount*, Manning, 2nd Ed., 2020 (Section 8.1-8.2)
- See <https://github.com/WinVector/PDSwR2> for all the datasets provided by the authors Zumel et al for the book above.
- See <https://github.com/WinVector/PDSwR2/raw/master/CodeExamples.zip> for all the R code that produces all the results and almost all the plots in the book above.

Single Variable Classification – Categorical

CITS4009 Computational Data Analysis

Unit Coordinator: Dr Du Huynh

Department of Computer Science and Software Engineering
The University of Western Australia

Semester 2, 2022

Classification and Scoring methods in data science

- The simplest classification method is one that always returns the majority category (a typical Null model)
- The simplest scoring (regression) method is one that always returns the average value of a subset of the original training data (a typical Null model)
- Other simpler methods (can be used for classification or scoring) include:
 - the single variable method (similar to the analyst's pivot table)
 - the nearest neighbor method
 - the Naïve Bayes method.

Section 1

Building Single Variable Models - Categorical

Single categorical variable model using `table()`

A single-variable model based on categorical features is easiest to describe as a table.

- Business analysts use what's called a *pivot table* (which promotes values or levels of a feature to be families of new columns) and
- statisticians use what's called a *contingency table* (where each possibility is given a column name).

In either case, the R command to produce a table is `table()`.

Single categorical variable model – example

```
outcome <- 'churn' # We can also try the 'appetency' and 'upselling' columns.
pos <- '1'         # We are interested in when 'churn' is positive. We need
                  # to put quotes around number 1 as it is the column name
                  # of the table created.

# Column 'Var218' is a categorical column containing 'cJvF' and 'UYBR'.
table218 <- table(Var218=dTrain[, 'Var218'], churn=dTrain[, outcome], useNA='ifany')
kable(table218)
```

	-1	1
cJvF	19245	1220
UYBR	17860	1618
NA	423	152

null model 代码

```
print(table218[,2] / (table218[,1] + table218[,2]))
##           cJvF           UYBR           <NA>
## 0.05961398 0.08306808 0.26434783
```

What does this mean when we use **Var218** to predict the outcome **churn**? — if the value of **Var218** is equal to **cJvF** then the predictor has **0.0596** probability of outputting a 1.

Function to repeat model building

```

# outCol: vector holding the values (known in the training step) of the
#         output column that we want to predict, e.g., the 'churn' column.
# varCol: the single variable column that is of interest. Can we use this
#         column alone to predict outCol?
# appCol: after building the model, we can apply it to this column (same
#         as varCol but may come from the calibration or test set).
mkPredC <- function(outCol, varCol, appCol) {
  pPos <- sum(outCol == pos) / length(outCol)
  naTab <- table(as.factor(outCol[is.na(varCol)]))
  pPosWna <- (naTab/sum(naTab))[pos]
  vTab <- table(as.factor(outCol), varCol)
  pPosWv <- (vTab[pos, ] + 1.0e-3*pPos) / (colSums(vTab) + 1.0e-3)
  pred <- pPosWv[appCol]
  pred[is.na(appCol)] <- pPosWna
  pred[is.na(pred)] <- pPos
  pred
}

```

Code with annotation

```

mkPredC <- function(outCol,varCol,appCol) {
  pPos <- sum(outCol==pos)/length(outCol)
  naTab <- table(as.factor(outCol[is.na(varCol)]))
  pPosWna <- (naTab/sum(naTab))[pos]
  vTab <- table(as.factor(outCol),varCol)
  pPosWv <- (vTab[pos,]+1.0e-3*pPos)/(colSums(vTab)+1.0e-3)
  pred <- pPosWv[appCol]
  pred[is.na(appCol)] <- pPosWna
  pred[is.na(pred)] <- pPos
  pred
}

```

Get stats on how often outcome is positive during training.

Make predictions by looking up levels of appCol.

Given a vector of training outcomes (outCol), a categorical training variable (varCol), and a prediction variable (appCol), use outCol and varCol to build a single-variable model and then apply the model to appCol to get new predictions.

Get stats on how often outcome is positive for NA values of variable during training.

Get stats on how often outcome is positive, conditioned on levels of training variable.

Add in predictions for NA levels of appCol.

Add in predictions for levels of appCol that weren't known during training.

Return vector of predictions.

Predict for all categorical variables

```
# call the mkPredC() function for all the categorical columns
for(v in catVars) {
  pi <- paste('pred', v, sep='')
  dTrain[,pi] <- mkPredC(dTrain[,outcome], dTrain[,v], dTrain[,v])
  dCal[,pi] <- mkPredC(dTrain[,outcome], dTrain[,v], dCal[,v])
  dTest[,pi] <- mkPredC(dTrain[,outcome], dTrain[,v], dTest[,v])
}
```

Recall that `outcome` was set to the value `churn` in an earlier slide. So we try to use each categorical column in `catVars` to predict when `churn` is positive (i.e., equal to 1).

Predict for all categorical variables (cont.)

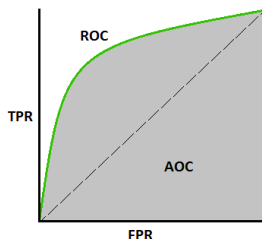
```
# We can inspect a few rows of the output column that has been added
# to dTrain for the categorical variable 'Var194'.
factor(dTrain[, "Var194"])["Levels"]  # how many levels are there?
## [1] <NA>
## Levels: CTUH lvza SEuy

rows <- c(620, 725, 9502, 40310)
dTrain[rows, c("Var194", "predVar194")]
##      Var194 predVar194
## 767      CTUH 0.06451643
## 895      lvza 0.06521745
## 11712     SEuy 0.06674519
## 49743    <NA> 0.07622426

rows <- c(842, 2885, 4507, 4510)
dCal[rows, c("Var194", "predVar194")]
##      Var194 predVar194
## 9040      CTUH 0.06451643
## 31804     lvza 0.06521745
## 49966     SEuy 0.06674519
## 49983    <NA> 0.07622426
```

What are the measures that we can use to evaluate the predictions above?

Area under ROC curve (AUC)



$$TPR/Recall/Sensitivity = \frac{TP}{TP + FN}$$

$$Specificity = \frac{TN}{TN + FP} \text{ Specificity is also known as } TNR)$$

$$FPR = 1 - Specificity = \frac{FP}{TN + FP}$$

Evaluate

Every classifier evaluation using ROCR starts with creating a `prediction` object created using the `prediction()` function.

The `performance()` function takes a prediction object and an evaluation metric to work out the relevant measurement.

```
library('ROCR')
calcAUC <- function(predcol,outcol) {
  perf <- performance(prediction(predcol,outcol==pos), 'auc')
  as.numeric(perf@y.values)
}
```

Evaluate (cont.)

```
for(v in catVars) {
  pi <- paste('pred', v, sep='')
  aucTrain <- calcAUC(dTrain[,pi], dTrain[,outcome])
  if (aucTrain >= 0.8) {
    aucCal <- calcAUC(dCal[,pi], dCal[,outcome])
    print(sprintf(
      "%s: trainAUC: %4.3f; calibrationAUC: %4.3f",
      pi, aucTrain, aucCal))
  }
}
```

```
## [1] "predVar200: trainAUC: 0.830; calibrationAUC: 0.565"
## [1] "predVar202: trainAUC: 0.827; calibrationAUC: 0.525"
## [1] "predVar214: trainAUC: 0.830; calibrationAUC: 0.565"
## [1] "predVar217: trainAUC: 0.897; calibrationAUC: 0.553"
```

100-fold cross-validation

Let's inspect the AUC values of two example categorical columns a bit more.

```
vars <- c('Var200', 'Var217')
for (var in vars) {
  aucs <- rep(0,100)
  for (rep in 1:length(aucs)) {
    useForCalRep <- rbinom(n=nrow(dTrainAll), size=1, prob=0.1) > 0
    predRep <- mkPredC(dTrainAll[!useForCalRep, outcome],
                      dTrainAll[!useForCalRep, var],
                      dTrainAll[useForCalRep, var])
    aucs[rep] <- calcAUC(predRep, dTrainAll[useForCalRep, outcome])
  }
  print(sprintf("%s: mean: %4.3f; sd: %4.3f", var, mean(aucs), sd(aucs)))
}
## [1] "Var200: mean: 0.549; sd: 0.014"
## [1] "Var217: mean: 0.554; sd: 0.014"
```

k-fold交叉验证，整个数据集被划分为 K 个 fold，其中1个用作验证集，其余 K-1 个用作训练集。该技术重复 K 次，直到每个fold都做过一次验证集，取K次的平均精度作为最终精度。不适用于不平衡的数据集

Messages from the cross-validation

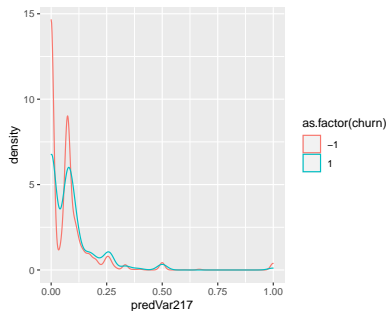
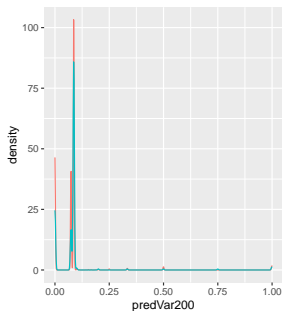
The 100-fold repeated estimates of the AUC give a mean of 0.549 for Var200 and 0.554 for Var217, both of which has standard deviation equal to 0.014.

- So the original AUC estimate of 0.565 for Var200 was a bit high, but our original AUC estimate of 0.553 for Var217 was very good.
- In some modelling circumstances, training set estimations are good enough (linear regression is often such an example).
- In many other circumstances, estimations from a single calibration set are good enough.
- In extreme cases (such as fitting models with very many variables or level values), you're well advised to use replicated cross-validation estimates of variable utilities and model fits.

Automatic cross-validation is extremely useful in all modeling situations, so it's critical that you automate your modelling steps so that you can perform

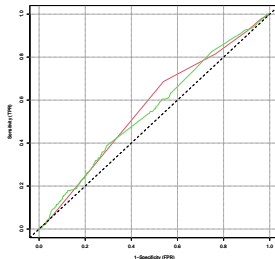
Double density plot – useful for inspecting the predicted probabilities

```
str(factor(dTrain[, "Var200"]))
## Factor w/ 13442 levels "_84etK_", "_9bT0Wp", ...: NA NA 11937 NA 401 NA 13079 2702 2449 11820 ...
str(factor(dTrain[, "Var217"]))
## Factor w/ 12481 levels "__VZ", "_00E", ...: 9492 NA 2784 4033 5109 2115 NA 2645 10407 11871 ...
fig1 <- ggplot(dCal) + geom_density(aes(x=predVar200, color=as.factor(churn)))
fig2 <- ggplot(dCal) + geom_density(aes(x=predVar217, color=as.factor(churn)))
grid.arrange(fig1, fig2, ncol=2)
```



Plotting the ROC curve

```
library(ROCit)
# colour_id 1-7 are: black,red,green,blue,cyan,purple,gold
plot_roc <- function(predcol, outcol, colour_id=2, overlaid=F) {
  ROCit_obj <- rocit(score=predcol, class=outcol==pos)
  par(new=overlaid)
  plot(ROCit_obj, col = c(colour_id, 1),
       legend = FALSE, YIndex = FALSE, values = FALSE)
}
plot_roc(dCal$predVar200, dCal[,outcome]) #red
plot_roc(dCal$predVar217, dCal[,outcome], colour_id=3, overlaid=T) # green
```



Interpret and choose the best single variable model

As expected, each variable's training AUC is inflated compared to its calibration AUC.

- This is because many of these variables have thousands of levels.
 - For example, `length(unique(dTrain$Var217))` is 12,434
- A good trick to work around this is to sort the variables by their AUC score on the calibration set (not seen during training), which is a better estimate of the variable's true utility.
- In our case, the most promising variable is variable 206, which has both training and calibration AUCs of 0.59.
- The winning KDD entry, which was a model that combined evidence from multiple features, had a much larger AUC of 0.76.

References

- **Practical Data Science with R**, *Nina Zumel, John Mount*, Manning, 2nd Ed., 2020 (Section 8.1-8.2)
- See <https://github.com/WinVector/PDSwR2> and e <https://github.com/WinVector/PDSwR2/raw/master/CodeExamples.zip> for all the datasets and code provided by the authors Zumel et al for the book above.

Single Variable Classification – Numerical

CITS4009 Computational Data Analysis

Unit Coordinator: Dr Du Huynh

Department of Computer Science and Software Engineering
The University of Western Australia

Semester 2, 2022

Discretisation: Binning numeric into categorical

- Bin the numeric feature into a number of ranges and then use the range labels as a new categorical variable.
- R can do this quickly with its `quantile()` and `cut()` commands,
- Let's look at how `quantile()` and `cut()` work on two example numeric columns.

```
(q1 <- quantile(dTrain[, "Var1"], probs=seq(0, 1, 0.1), na.rm=T))
##  0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
##   0    0    0    0    0    0    8    8   16   24  680
(q6 <- quantile(dTrain[, "Var6"], probs=seq(0, 1, 0.1), na.rm=T))
##   0%   10%   20%   30%   40%   50%   60%   70%   80%   90%  100%
##   0   161   406   623   763   861  1008  1260  1659  2520 114079

# dis.Var1 and dis.Var6 are the discretised version of dTrain[, "Var1"] and dTrain[, "Var6"]
dis.Var1 <- cut(dTrain[, "Var1"], unique(q1))
dis.Var6 <- cut(dTrain[, "Var6"], unique(q6))

# inspect the number of levels
dis.Var1["Levels"]
## [1] <NA>
## Levels: (0,8] (8,16] (16,24] (24,680]
dis.Var6["Levels"]
## [1] <NA>
## 10 Levels: (0,161] (161,406] (406,623] (623,763] (763,861] ... (2.52e+03,1.14e+05]
```

Discretisation: Binning numeric into categorical (cont.)

- Put all these operations into a function.

```
mkPredN <- function(outCol, varCol, appCol) {  
  # compute the cuts  
  cuts <- unique(  
    quantile(varCol, probs=seq(0, 1, 0.1), na.rm=T))  
  # discretize the numerical columns  
  varC <- cut(varCol, cuts)  
  appC <- cut(appCol, cuts)  
  
  mkPredC(outCol, varC, appC)  
}
```

Processing all numerical variables

```
for(v in numericVars) {  
  pi <- paste('pred', v, sep='')  
  dTrain[,pi] <- mkPredN(dTrain[,outcome], dTrain[,v], dTrain[,v])  
  dCal[,pi] <- mkPredN(dTrain[,outcome], dTrain[,v], dCal[,v])  
  dTest[,pi] <- mkPredN(dTrain[,outcome], dTrain[,v], dTest[,v])  
  aucTrain <- calcAUC(dTrain[,pi], dTrain[,outcome])  
  
  if(aucTrain >= 0.55) {  
    aucCal <- calcAUC(dCal[,pi], dCal[,outcome])  
    print(sprintf(  
      "%s: trainAUC: %4.3f; calibrationAUC: %4.3f",  
      pi, aucTrain, aucCal))  
  }  
}
```

```
## [1] "predVar6: trainAUC: 0.557; calibrationAUC: 0.554"  
## [1] "predVar7: trainAUC: 0.555; calibrationAUC: 0.565"  
## [1] "predVar13: trainAUC: 0.568; calibrationAUC: 0.553"  
## [1] "predVar73: trainAUC: 0.608; calibrationAUC: 0.616"  
## [1] "predVar74: trainAUC: 0.574; calibrationAUC: 0.566"  
## [1] "predVar81: trainAUC: 0.558; calibrationAUC: 0.542"
```

100-fold cross-validation

Let's inspect the AUC values of two example numerical columns a bit more.

```
vars <- c('Var7', 'Var13', 'Var73')

for (var in vars) {
  aucs <- rep(0,100)
  for (rep in 1:length(aucs)) {
    useForCalRep <- rbinom(n=nrow(dTrainAll), size=1, prob=0.1) > 0
    predRep <- mkPredN(dTrainAll[!useForCalRep, outcome],
                      dTrainAll[!useForCalRep, var],
                      dTrainAll[useForCalRep, var])
    aucs[rep] <- calcAUC(predRep, dTrainAll[useForCalRep, outcome])
  }
  print(sprintf("%s: mean: %4.3f; sd: %4.3f", var, mean(aucs), sd(aucs)))
}

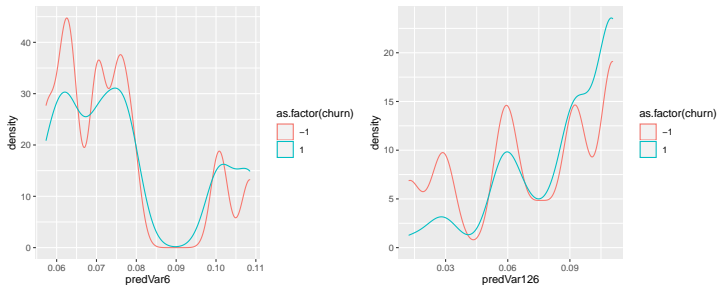
## [1] "Var7: mean: 0.555; sd: 0.015"
## [1] "Var13: mean: 0.563; sd: 0.016"
## [1] "Var73: mean: 0.605; sd: 0.015"
```

Messages from the cross-validation

- On slide 4/11, the predictions using **Var7** and **Var73** have slightly higher calibrationAUC values than the trainAUC values.
- The 100-fold repeated estimates of the AUC show that that happened simply by chance, as on slide 5/11, all the **calibrationAUC** values are slightly lower than the **trainAUC** values on slide 4/11.

Double density plot – useful for inspecting the predicted probabilities

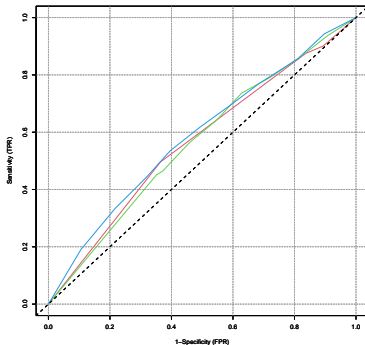
```
calcAUC(dTrain[, "predVar126"], dTrain[, outcome]); calcAUC(dCal[, "predVar126"], dCal[, outcome])  
## [1] 0.6349584  
## [1] 0.6288453  
fig1 <- ggplot(dCal) + geom_density(aes(x=predVar6, color=as.factor(churn)))  
fig2 <- ggplot(dCal) + geom_density(aes(x=predVar126, color=as.factor(churn)))  
grid.arrange(fig1, fig2, ncol=2)
```



We can deduce that churning (i.e., **churn=1**) is unlikely when the value of **predVar126** is low (the graph is read by comparing the areas under the curves).

Comparing the ROC curves for the test set

```
# call calcAUC() to calculate the AUC of each variable (code omitted)
cat("Var7's AUC:", auc.predVar7, "; Var13's AUC:", auc.predVar13, "; Var73's AUC:",
## Var7's AUC: 0.5658961 ; Var13's AUC: 0.5636032 ; Var73's AUC: 0.5854976
plot_roc(dTest$predVar7, dTest[,outcome]) #red
plot_roc(dTest$predVar13, dTest[,outcome], colour_id=3, overlaid=T) #green
plot_roc(dTest$predVar73, dTest[,outcome], colour_id=4, overlaid=T) #blue
```



How about the Null model?

```
(Npos <- sum(dTrain[,outcome] == 1))  
## [1] 2990  
pred.Null <- Npos / nrow(dTrain)  
cat("Proportion of outcome == 1 in dTrain:", pred.Null)  
## Proportion of outcome == 1 in dTrain: 0.07379436
```

We have a classification task where `churn` takes on the value `1` or `-1`. The simplest Null model is to always output `pred.Null` as the predicted probability. How does this model perform on the calibration set?

```
TP <- 0; TN <- sum(dCal[,outcome] == -1); # using threshold 0.5  
FP <- 0; FN <- sum(dCal[,outcome] == 1); # using threshold 0.5  
cat("nrow(dCal):", nrow(dCal), "TP:", TP, "TN:", TN, "FP:", FP, "FN:", FN)  
## nrow(dCal): 4510 TP: 0 TN: 4181 FP: 0 FN: 329  
(accuracy <- (TP + TN) / nrow(dCal))  
## [1] 0.927051  
(precision <- TP/(TP + FP))  
## [1] NaN  
(recall <- TP/(TP + FN))  
## [1] 0  
(AUC <- calcAUC(pred.Null, dCal[,outcome]))  
## [1] 0.5
```

Take home messages

- How to deal with categorical variables
- How to deal with numerical variables
- Single Variable Models
- How to select models based on AUC values

References

- **Practical Data Science with R**, *Nina Zumel, John Mount*, Manning, 2nd Ed., 2020 (Sections 8.1-8.2)
- **Decision Tree Algorithm**: <https://medium.com/deep-math-machine-learning-ai/chapter-4-decision-trees-algorithms-b93975f7a1f1>
- **Best Machine Learning Packages in R**: <https://www.r-bloggers.com/what-are-the-best-machine-learning-packages-in-r/>