R Programs for Text Book Examples

Chapter 5: Evaluating Predictive Performance

Code for accuracy measure. Prediction error metrics from a model for Toyota car prices. Training and validation (Figure 5.1)

```
####
# package forecast is required to evaluate performance
library(forecast)

# load file
toyota.corolla.df <- read.csv("ToyotaCorolla.csv")

# randomly generate training and validation sets
training <- sample(toyota.corolla.df$Id, 600)
validation <- sample(setdiff(toyota.corolla.df$Id, training), 400)

# run linear regression model
reg <- lm(Price~., data=toyota.corolla.df[,-c(1,2,8,11)], subset=training,
        na.action=na.exclude)
pred_t <- predict(reg, na.action=na.pass)
pred_v <- predict(reg, newdata=toyota.corolla.df[validation,-c(1,2,8,11)],
              na.action=na.pass)

## evaluate performance
# training
accuracy(pred_t, toyota.corolla.df[training,]$Price)
# validation
accuracy(pred_v, toyota.corolla.df[validation,]$Price)
```

Code for generating a lift chart and decile-wise lift chart. Lift chart and decile lift chart for continuous outcome variable (sales of Toyota cars) (Figure 5.3)

```
####
# remove missing Price data
toyota.corolla.df <-
  toyota.corolla.df[!is.na(toyota.corolla.df[validation,]$Price),]

# generate random Training and Validation sets
training <- sample(toyota.corolla.df$Id, 600)
validation <- sample(setdiff(toyota.corolla.df$Id, training), 400)

# regression model based on all numerical predictors
reg <- lm(Price~., data = toyota.corolla.df[,-c(1,2,8,11)], subset = training)
```

```
# predictions
pred_v <- predict(reg, newdata = toyota.corolla.df[validation,-c(1,2,8,11)])

# load package gains, compute gains (we will use package caret for categorical y later)
library(gains)
gain <- gains(toyota.corolla.df[validation,]$Price[!is.na(pred_v)], pred_v[!is.na(pred_v)])

# cumulative lift chart
options(scipen=999) # avoid scientific notation
# we will compute the gain relative to price
price <- toyota.corolla.df[validation,]$Price[!is.na(toyota.corolla.df[validation,]$Price)]
plot(c(0,gain$cume.pct.of.total*sum(price))~c(0,gain$cume.obs),
   xlab="# cases", ylab="Cumulative Price", main="Lift Chart", type="l")

# baseline
lines(c(0,sum(price))~c(0,dim(toyota.corolla.df[validation,])[1]), col="gray", lty=2)

# Decile-wise lift chart
barplot(gain$mean.resp/mean(price), names.arg = gain$depth,
     xlab = "Percentile", ylab = "Mean Response", main = "Decile-wise lift chart")
```

Confusion matrices based on cutoffs of 0.5, 0.25, and 0.75 (riding mowers example) (Figure 5.6)

```
####
library(caret)
library(e1071)

owner.df <- read.csv("ownerExample.csv")
confusionMatrix(ifelse(owner.df$Probability>0.5, 'owner', 'nonowner'), owner.df$Class)
confusionMatrix(ifelse(owner.df$Probability>0.25, 'owner', 'nonowner'), owner.df$Class)
confusionMatrix(ifelse(owner.df$Probability>0.75, 'owner', 'nonowner'), owner.df$Class)
```

Code for creating figure of accuracy and overall error as a function of the cutoff value. Plotting accuracy and overall error as a function of the cutoff value (riding mowers example) (Figure 5.7)

```
####
# replace data.frame with your own
df <- read.csv("liftExample.csv")

# create empty accuracy table
accT = c()

# compute accuracy per cutoff
for (cut in seq(0,1,0.1)){
 cm <- confusionMatrix(1 * (df$prob > cut), df$actual)
 accT = c(accT, cm$overall[1])
```

```
}

# plot accuracy
plot(accT ~ seq(0,1,0.1), xlab = "Cutoff Value", ylab = "", type = "l", ylim = c(0, 1))
lines(1-accT ~ seq(0,1,0.1), type = "l", lty = 2)
legend("topright",  c("accuracy", "overall error"), lty = c(1, 2), merge = TRUE)
```

Code for generating ROC curve and computing AUC. ROC curve for riding mowers example (Figure 5.9)

```
####
library(pROC)
r <- roc(df$actual, df$prob)
plot.roc(r)

# compute auc
auc(r)
```

Code for creating a lift chart: two options. Lift chart for the mower example using caret package and gains package (Figure 5.10))

```
####
# first option with 'caret' library:
library(caret)
lift.example <- lift(relevel(as.factor(actual), ref="1") ~ prob, data = df)
xyplot(lift.example, plot = "gain")

# Second option with 'gains' library:
library(gains)
df <- read.csv("liftExample.csv")
gain <- gains(df$actual, df$prob, groups=dim(df)[1])
plot(c(0, gain$cume.pct.of.total*sum(df$actual)) ~ c(0, gain$cume.obs),
    xlab = "# cases", ylab = "Cumulative", type="l")
lines(c(0,sum(df$actual))~c(0,dim(df)[1]), col="gray", lty=2)
```

Code for creating a decile lift chart. Decile lift chart (Figure 5.11)

```
####
# use gains() to compute deciles.
# when using the caret package, deciles must be computed manually.

gain <- gains(df$actual, df$prob)
barplot(gain$mean.resp / mean(df$actual), names.arg = gain$depth, xlab = "Percentile",
      ylab = "Mean Response", main = "Decile-wise lift chart")
```

Chapter 7: k-Nearest-Neighbors (k-NN)

Code for loading and partitioning the riding mower data, and plotting scatter plot. Scatter plot of Lot Size vs. Income for the 18 households in the training set and the new household to be classified (Figure 7.1)

```
####
mower.df <- read.csv("RidingMowers.csv")
set.seed(111)
train.index <- sample(row.names(mower.df), 0.6*dim(mower.df)[1])
valid.index <- setdiff(row.names(mower.df), train.index)
train.df <- mower.df[train.index, ]
valid.df <- mower.df[valid.index, ]
## new household
new.df <- data.frame(Income = 60, Lot_Size = 20)

## scatter plot
plot(Lot_Size ~ Income, data=train.df, pch=ifelse(train.df$Ownership=="Owner", 1, 3))
text(train.df$Income, train.df$Lot_Size, rownames(train.df), pos=4)
text(60, 20, "X")
legend("topright", c("owner", "non-owner", "newhousehold"), pch = c(1, 3, 4))
```

Code for normalizing data and finding nearest neighbors. Running -NN (Figure 7.1)

```
####
# initialize normalized training, validation data, complete data frames to originals
train.norm.df <- train.df
valid.norm.df <- valid.df
mower.norm.df <- mower.df
# use preProcess() from the caret package to normalize Income and Lot_Size.
norm.values <- preProcess(train.df[, 1:2], method=c("center", "scale"))
train.norm.df[, 1:2] <- predict(norm.values, train.df[, 1:2])
valid.norm.df[, 1:2] <- predict(norm.values, valid.df[, 1:2])
mower.norm.df[, 1:2] <- predict(norm.values, mower.df[, 1:2])
new.norm.df <- predict(norm.values, new.df)

# use knn() to compute knn.
# knn() is available in library FNN (provides a list of the nearest neighbors)
# and library class (allows a numerical output variable).
library(FNN)
nn <- knn(train = train.norm.df[, 1:2], test = new.norm.df, cl = train.norm.df[, 3], k = 3)

row.names(train.df)[attr(nn, "nn.index")]
```

Code for measuring the accuracy of different k values. Accuracy (or correct rate) of -NN predictions in validation set for various choices of .k (Figure 7.2)

```
####
library(caret)

# initialize a data frame with two columns: k, and accuracy.
accuracy.df <- data.frame(k = seq(1, 14, 1), accuracy = rep(0, 14))

# compute knn for different k on validation.
for(i in 1:14) {
  knn.pred <- knn(train.norm.df[, 1:2], valid.norm.df[, 1:2],
               cl = train.norm.df[, 3], k = i)
  accuracy.df[i, 2] <- confusionMatrix(knn.pred, valid.norm.df[, 3])$overall[1]
}
accuracy.df
```

Code for running the k-NN algorithm to classify the new household. Classifying a new household using the "best k" = 4

```
####
knn.pred.new <- knn(mower.norm.df[, 1:2], new.norm.df,
                cl = mower.norm.df[, 3], k = 4)
row.names(train.df)[attr(nn, "nn.index")]
```

Chapter 8: The Naïve Bayes Classifier

Code for running naive Bayes. Naive Bayes classifier applied to flight delays (training) data (Figure 8.1)

```
####
library(e1071)
delays.df <- read.csv("FlightDelays.csv")

# change numerical variables to categorical first
delays.df$DAY_WEEK <- factor(delays.df$DAY_WEEK)
delays.df$DEP_TIME <- factor(delays.df$DEP_TIME)
# create hourly bins departure time
delays.df$CRS_DEP_TIME <- factor(round(delays.df$CRS_DEP_TIME/100))

# Create training and validation sets.
selected.var <- c(10, 1, 8, 4, 2, 13)
train.index <- sample(c(1:dim(delays.df)[1]), dim(delays.df)[1]*0.6)
train.df <- delays.df[train.index, selected.var]
valid.df <- delays.df[-train.index, selected.var]

# run naive bayes
```

```
delays.nb <- naiveBayes(Flight.Status ~ ., data = train.df)
delays.nb
```

Pivot table of flight status by destination airport (training data) (Table 8.4)

```
####
# use prop.table() with margin = 1 to convert a count table to a proportion table,
# where each row sums up to 1 (use margin = 2 for column sums).
prop.table(table(train.df$Flight.Status, train.df$DEST), margin = 1)
```

Code for scoring data using naive Bayes. Scoring the example flight (probability and class) (Figure 8.2)

```
####
## predict probabilities
pred.prob <- predict(delays.nb, newdata = valid.df, type = "raw")
## predict class membership
pred.class <- predict(delays.nb, newdata = valid.df)

df <- data.frame(actual = valid.df$Flight.Status, predicted = pred.class, pred.prob)

df[valid.df$CARRIER == "DL" & valid.df$DAY_WEEK == 7 & valid.df$CRS_DEP_TIME == 10 &
    valid.df$DEST == "LGA" & valid.df$ORIGIN == "DCA",]
```

Code for confusion matrices. Confusion matrices for flight delay using a naive Bayes classifier (Figure 8.3)

```
####
library(caret)

# training
pred.class <- predict(delays.nb, newdata = train.df)
confusionMatrix(pred.class, train.df$Flight.Status)

# validation
pred.class <- predict(delays.nb, newdata = valid.df)
confusionMatrix(pred.class, valid.df$Flight.Status)
```

Code for creating lift chart of naive Bayes classifier applied to flight delays data (Figure 8.4)

```
####
library(gains)
gain <- gains(ifelse(valid.df$Flight.Status=="delayed",1,0), pred.prob[,1], groups=100)

plot(c(0,gain$cume.pct.of.total*sum(valid.df$Flight.Status=="delayed"))~c(0,gain$cume.obs), xlab="#
cases", ylab="Cumulative", main="", type="l")
lines(c(0,sum(valid.df$Flight.Status=="delayed"))~c(0, dim(valid.df)[1]), lty=2)
```