Study Note: Model Validation and Holdout Data

1/13/2019

The importance of testing models on out-of-sample data should not be underestimated. Testing a model on data that has been used to fit the model's parameters will provide an overly optimistic assessment of its predictive power. This study note provides some general remarks on model validation, the importance of using out-of-sample data, and examples of diagnostic tools such as:

- the Gini coefficient,
- the Lorenz curve,
- lift charts, and
- the receiver operating characteristic curve.

General Remarks

The proper testing of models on out-of-sample data is absolutely crucial. However, there are different types of testing that accomplish different things. For example, it is common to divide one's data into three parts: (1) a training set, used for building models; (2) a test set, used frequently during the model build process to test those models; and (3) a holdout set, saved until the end of the modeling process, to provide an objective metric of goodness of fit that can be reported to management. (The term "validation set" is sometimes used, but since it is sometimes used for (2) and sometimes for (3), we have avoided this term.) The metric used could be a Gini coefficient, the area under the receiver operating characteristic (ROC) curve, the squared error (or other cost function), a gains chart, or a lift curve, in each case applied to the holdout data the model has not yet seen. In order to ensure that the holdout test is objective, one should separate the holdout data as early in the process as practicable[1], and store it in a separate file not accessible during the modeling process.

There is no hard-and-fast rule for the sizes of these sets across all types of models, but for many types of models, it is usual to assign one-third of the data at random to each set. It may be tempting to make the training set larger and the test and holdout datasets smaller, but (a) building on a smaller dataset is a good way to avoid a natural tendency toward overfit models with too many parameters and (b) if your test and holdout datasets are too small, one will be tempted not to trust the out-of-sample tests, which rather defeats their purpose.

The reason that the test set will not serve the purpose of the objective test is that, since one has repeatedly compared to it, one has in some sense fit the model to the test data as well as to the training data. In fact, one can make a virtue of this and swap the roles of the training and test data during the course of the modeling process, as one zeroes in on the best model.

We are often asked why one needs holdout data when one can get an out-of-sample test using cross-validation, where a dataset is divided into K folds and the goodness of fit calculated by using the model fit to all folds but the i-th one to make predictions for the observations in the i-th fold. This can indeed work if the model building

[1] Typically one does some level of data profiling before separating the holdout data in order to: (a) match control totals to ensure the dataset is complete and correct; and (b) ensure that all possible values are known for each categorical field.

process is completely automated. However, when human beings are building the model, the choice of model (including type of model, which variables were candidates to final feature selection, which derived features were created) has typically been impacted by all the data the modeler has worked with…the whole dataset. Even though one may be fitting parameters and even performing final feature selection separately for each collection of K-1 folds, those models will have been influenced by all the data. This does not mean that cross-validation is useless for human-built models. It merely means that the proper use of cross-validation is as a substitute for the training/test split, not as a substitute for having true holdout data. (In fact, the suggestion above to swap training and test data during the modeling process is a form of two-fold cross-validation that is very tractable for human modelers.) Also, note that nothing is lost in terms of the final model by using the holdout approach. It is familiar in the cross-validation approach that the union of the cross-validation models provides the goodness of fit metrics whilst the model to be used is fit on all the data at once. The same is true in the holdout approach. After testing on holdout data to obtain the objective quality of the model, one can (and usually does) refit on the complete data to obtain the final model. One can even change the model if adding the holdout data makes it clear that one should. What one cannot do is claim a better goodness of fit than one has already measured.

In order to ensure independence of the training, test, and validation data, it is good practice to ensure the correlated observations (i.e., ones that are still correlated even after conditioning on the model variables) are all put into the same group. For example, if the unit of observation is one policy for one year, the series of observations that correspond to a single policyholder over that period of time would be put into the same group. Or if the model is a severity model for weather claims, claims from the same storm might be put into the same group. If time is the main driver of correlation then one might use a contiguous block of time for each group, known as out-of-time validation. For many insurance problems, however, time is not the main dimension, and splitting on the time variable may actually be undesirable because one may want to incorporate time explicitly in the model or to test consistency of effects across time within the training data as one of the criteria for including a variable. This latter is actually straightforward to do: Look at residuals across the levels of the variable in question for the earlier half (temporally) of the data and for the latter half of the data, and compare.

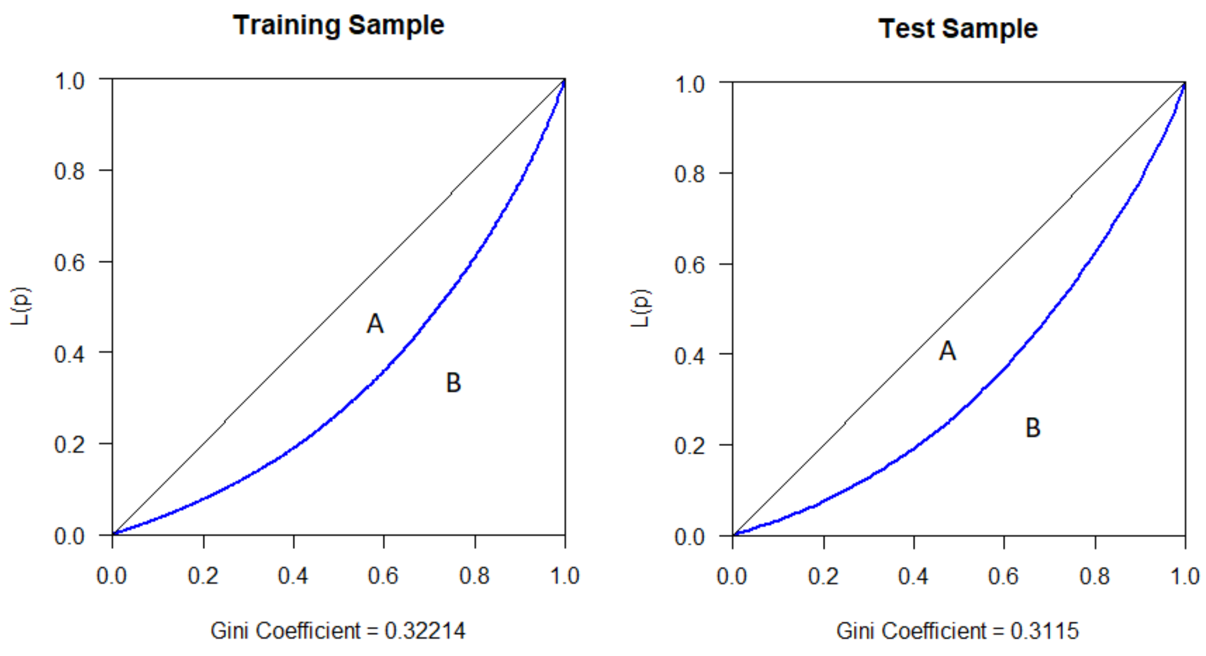
Finally, one practical aspect of splitting data for testing and validation is that datasets evolve. Perhaps, for example, it turns out that a small set of observations were erroneously included, or the business identifies certain observations as not relevant to going-forward strategy and asks that they be omitted. If you have used a random number generator and appended a string of random numbers to a list of policy numbers in order to split the data into thirds, now you not only need to have stored the random number seed and have access to the exact same random number generator to reproduce the random split, and to have sorted your data the precise way you had done previously, you also need to match at precisely the right midway point in your data filtering process. Thus, it can be much simpler to use a hash of the relevant unit information (e.g., policy number and policyholder name) rather than a random number generator in order to assign an observation to a group. However, the selection of hash function is critical, since one needs to avoid assigning similar hashes to similar initial values, and most hash functions will fail this test. However, the hash function MD5 is considered suitable for this purpose[2].

[2] The interested reader will find additional information in section 4.1 of Kohavi, Henne, and Sommerfield, "Practical Guide to Controlled Experiments on the Web," KDD2007, currently available at https://ai.stanford.edu/~ronnyk/2007GuideControlledExperiments.pdf

The Gini Coefficient and the Lorenz Curve

The Gini Coefficient is named after Corrado Gini, an Italian economist who developed as a measure of statistical dispersion of income or wealth within the population of a nation. For more details, see the following Wikipedia article: https://en.wikipedia.org/wiki/Gini_coefficient. The Lorenz Curve is related to the Gini coefficient, and it shows the distribution of income or wealth graphically. See Wikipedia article here: https://en.wikipedia.org/wiki/Lorenz_curve.

The following example illustrates the relationship between the Lorenz Curve and the Gini Coefficient using chick weight data from the ChickWeight data set in the R datasets package. The horizontal axis is the percentage of total chickens, and the vertical axis is the percentage of total weight in a training sample. The diagonal line is the "line of equality." If all chickens were the same weight, as the cumulative count of chickens increased, the cumulative sum of their weight would increase in the same proportion. However, when you sort records in increasing order of weight, cumulative weight increases slowly at first, so the Lorenz Curve, in blue, increases slowly. As more and more larger chick weights are added, the percentage of total chick weight increases more rapidly that the percentage of total chick count, and the curve becomes steeper. In the graph, the area between the line of equality and the Lorenz Curve is labeled A, and the area below the Lorenz Curve is labeled B. The Gini coefficient is the ratio A to the sum of A and B. Thus, Gini = A / (A + B) = 2A, since A + B = 1/2.



Note there is a slight difference between the training and test samples. The training sample has a slightly higher Gini coefficient, indicating there is slightly more variation in the data. I produced the plots above using the gini function from the reldist package, and the Lc function from the ineq package, as follows:

```
#
# use ChickWeight data from datasets package
#
cw <- as.data.frame(ChickWeight)
set.seed(123)
row_count <- dim(cw)[1]
cw$count <- as.numeric(rep(1,row_count))
cw$sample <- as.factor(sample(c("Test","Train"),
                                row_count, replace=TRUE,
                                prob = c(0.3,0.7)))
#
# check row counts and percentages in resulting samples
#
summary(cw$sample)
summary(cw$sample) / row_count
#
# Gini Coefficient and Lorenz Curve on training data
#
library(ineq)
library(reldist)
cw_train <- subset(cw, sample == "Train")
train_gini <- gini(cw_train$weight)
xtitle <- paste("Gini Coefficient =", round(train_gini,5))
plot(Lc(cw_train$weight), col = "blue", lwd = 2,
     main = "Training Sample", xlab = xtitle)
round(with(cw_train, WeightedGini(weight, count, weight)), 5)
#
# Gini Coefficient and Lorenz Curve on test data
#
cw_test <- subset(cw, sample == "Test")
test_gini <- gini(cw_test$weight)
xtitle <- paste("Gini Coefficient =", round(test_gini,5))
plot(Lc(cw_test$weight), col = "blue", lwd = 2,
     main = "Test Sample", xlab = xtitle)
round(with(cw_test, WeightedGini(weight, count, weight)), 5)
```

To produce a Gini coefficient and a Lorenz curve plot for a model, it is necessary to modify the functions. Instead of sorting based on actual data, one sorts by predicted values. It is also advisable to include a random number column as a secondary sort key. If a model produces the same predicted value for large segments of the data, and if those segments had been previously sorted in ascending order of actual values, and if the sort function preserves that order, the Gini coefficient and Lorenz curve might look better than they ought to be. Introducing a random number column as a secondary sort key avoids that issue.

Normalized Gini Coefficient

A good model should reflect the heterogeneity of the data. However, if it reflects too much of the heterogeneity of the data, it is overfit, and it will not generalize well. A measure of how well a model fits the data is the ratio of the Gini coefficient for data sorted by predicted values (model Gini) to the ratio of the Gini coefficient for data sorted by actual values (data Gini), such as NormalizedWeightedGini function in https://www.kaggle.com/c/liberty-mutual-fire-peril/discussion/9880

Normalized Gini = Model Gini / Data Gini

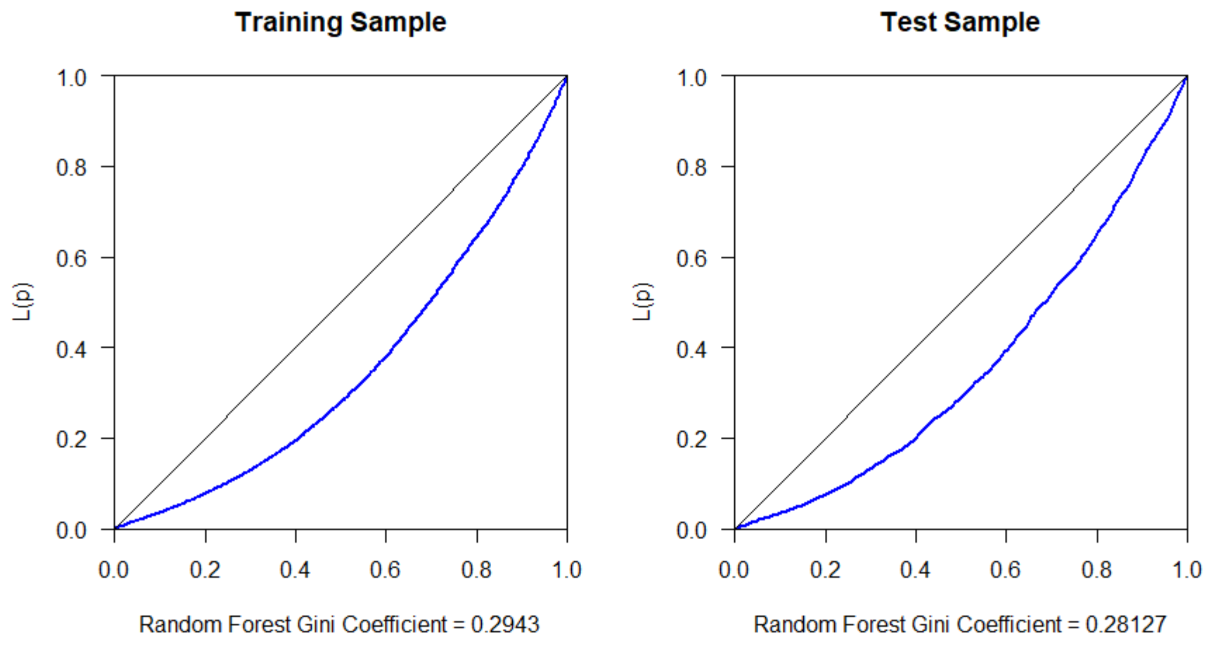
The normalized Gini coefficient adjusts for a possible difference between the Gini coefficient of the training sample versus the Gini coefficient for a testing or holdout sample.

Considerations

1. A saturated model, one with as many parameters as there are observations, and where the predicted value is equal to the actual value, would have a normalized Gini equal to unity. Such a model, however, would not generalize well. Indeed, how would one score a different data set?
2. A model where the predicted value is a constant, would have a normalized Gini equal to zero.
3. A model generalizes well if its normalized Gini in a testing or holdout sample is reasonably close to its normalized Gini in the training sample.

Thus, evaluating the normalized Gini coefficient of a model is a bit of an art. The model should have a reasonably high normalized Gini, and the normalized Gini when testing the model on a testing or holdout sample should be reasonably close to the normalized Gini when testing on the training sample.

As an example, we use the normalized Gini coefficient to compare two different models (a random forest and an ordinary linear regression) fitted to the chick weight data. The following graphs show the Gini coefficient for the random forest model.



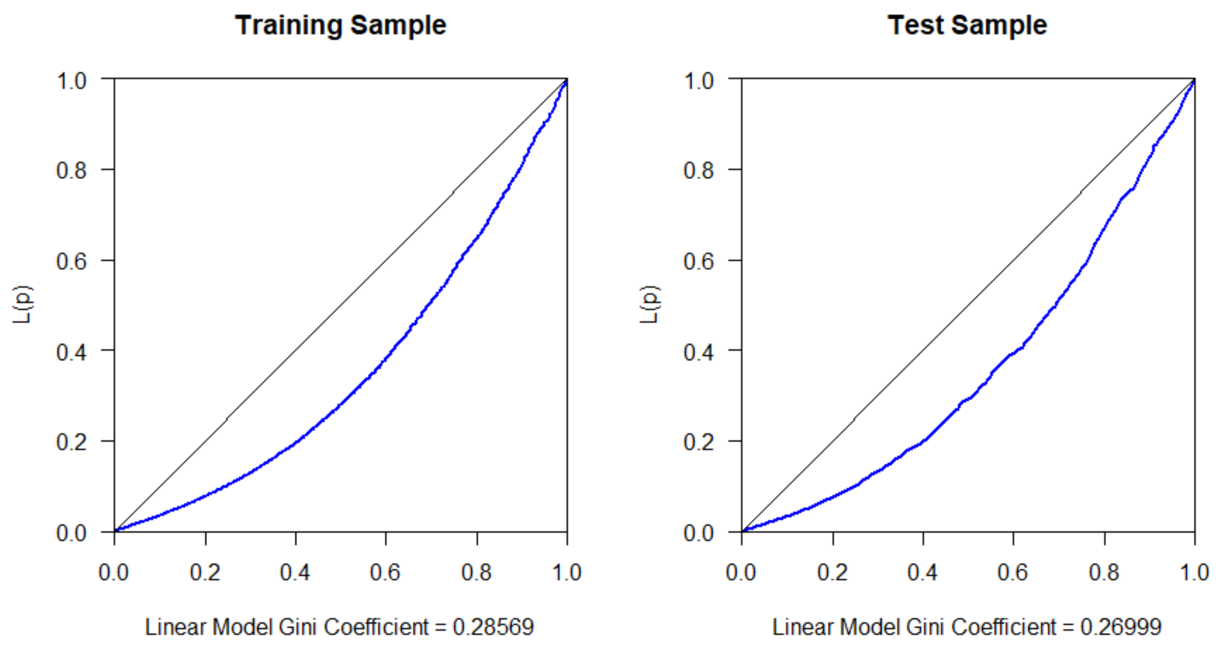
The normalized Gini coefficient for this model on the training and test samples is:

Normalized Gini on training sample = 0.2943 / 0.32214 = 0.91358

Normalized Gini on test sample = 0.28127 / 0.3115 = 0.90295

Difference in normalized Gini = 0.91358 - 0.90295 = 0.01063

The following two graphs show the same information as the previous two but for the ordinary linear regression model.



Similarly, the normalized Gini coefficient for this model is:

Normalized Gini on training sample = 0.28569 / 0.32214 = 0.88685

Normalized Gini on test sample = 0.26999 / 0.3115 = 0.86674

Difference in normalized Gini = 0.88685 - 0.86674 = 0.02011

Note that the normalized Gini for the random forest model is higher on both the training and testing samples than the normalized Gini for the linear regression model. Furthermore, the difference in normalized Gini between the training and testing sample is smaller for the random forest than it is for the linear model. Thus, in this example, the random forest model has a better fit, and generalizes better than the linear model.

The R code used to generate the plots above follows.

```
#

# Fit random forest model on training data
#
library(randomForest)
cw_train <- subset(cw, sample == "Train")
set.seed(456)
m1.cw <- randomForest(weight ~ Diet + Time,
                               data = cw_train)
# Predicted on Training Sample
cw_train$weight_p <- predict(m1.cw, cw_train)
train_pgin <- round(with(cw_train,
                   WeightedGini(weight, count, weight_p)), 5)
xtitle <- paste("Random Forest Gini Coefficient =",
                round(train_pgin,5))
with(cw_train, plot(LorenzCurve2(weight, count, weight_p),
     col="blue", lwd=2,
     main="Training Sample", xlab=xtitle))
# Predicted on Test Sample
cw_test$weight_p <- predict(m1.cw, cw_test)
test_pgin <- round(with(cw_test,
                         WeightedGini(weight, count, weight_p)), 5)
xtitle <- paste("Random Forest Gini Coefficient =",
                round(test_pgin,5))
with(cw_test, plot(LorenzCurve2(weight, count, weight_p),
                    col="blue", lwd=2,
                    main="Test Sample", xlab=xtitle))
#
# Fit linear regression training data
#
cw_train <- subset(cw, sample == "Train")
m3.cw <- lm(weight ~ Time,
               data = cw_train)
# Predicted on Training Sample
cw_train$weight_p <- predict(m3.cw, cw_train)
train_pgin <- round(with(cw_train,
                         WeightedGini(weight, count, weight_p)), 5)
xtitle <- paste("Linear Model Gini Coefficient =",
                round(train_pgin,5))
with(cw_train, plot(LorenzCurve2(weight, count, weight_p),
                    col="blue", lwd=2,
                    main="Training Sample", xlab=xtitle))
# Predicted on Test Sample
cw_test$weight_p <- predict(m3.cw, cw_test)
test_pgin <- round(with(cw_test,
                        WeightedGini(weight, count, weight_p)), 5)
xtitle <- paste("Linear Model Gini Coefficient =",
                round(test_pgin,5))
with(cw_test, plot(LorenzCurve2(weight, count, weight_p),
                   col="blue", lwd=2,
                   main="Test Sample", xlab=xtitle))
```

The following table shows the gini function from the reldist package, versus the WeightedGini function posted by William Cukierski on the Kaggle website at https://www.kaggle.com/c/liberty-mutual-fire-peril/discussion/9880, with some modifications for clarity and to add a random number key to break the tie in cases where the model produces the same prediction. Some sort routines keep the data's original order when records have the same value of the sort key. If the data set had been previously sorted by actual values, this could cause the value of the Gini coefficient to be higher than it should be.

The table also shows the original Lc (Lorenz curve) function from the ineq package as well as a version modified to allow sorting by predicted values.

`gini()`

```
function (x, weights = rep(1, length = length(x)))
{
    ox <- order(x)
    x <- x[ox]
    weights <- weights[ox]/sum(weights)
    p <- cumsum(weights)
    nu <- cumsum(weights * x)
    n <- length(nu)
    nu <- nu/nu[n]
    sum(nu[-1] * p[-n]) - sum(nu[-n] * p[-1])
}
```

`WeightedGini()`

```
function(actual, weights, predicted){
      # Modification of code posted in Kaggle by William Cukierski
      # https://www.kaggle.com/c/liberty-mutual-fire-peril/discussion/9880
      # actual = actual frequency, severity, loss cost
      # corresponding weights = exposure, claim count, exposure
      # predicted = predicted
      df = data.frame(actual = actual, weights = weights, predicted =
predicted)
      # create random number sort key so ties will be resolved in random order
      k <- length(df$actual)
      df$rkey <- runif(k)
      df <- df[order(df$predicted, df$rkey),]
      df$random = cumsum((df$weights/sum(df$weights)))
      totalPositive <- sum(df$actual * df$weights)
      df$cumPosFound <- cumsum(df$actual * df$weights)
      df$Lorentz <- df$cumPosFound / totalPositive
      n <- nrow(df)
      gini <- sum(df$Lorentz[-1]*df$random[-n]) - sum(df$Lorentz[-n]
* df$random[-1])
      return(gini)
}
```

`Lc`

```
function (x, n = rep(1, length(x)), plot = FALSE)
{
    ina <- !is.na(x)
    n <- n[ina]
    x <- as.numeric(x)[ina]
    k <- length(x)
    o <- order(x)
    x <- x[o]
    n <- n[o]
    x <- n * x
    p <- cumsum(n)/sum(n)
    L <- cumsum(x)/sum(x)
    p <- c(0, p)
    L <- c(0, L)
    L2 <- L * mean(x)/mean(n)
    Lc <- list(p, L, L2)
    names(Lc) <- c("p", "L", "L.general")
    class(Lc) <- "Lc"
    if (plot)
        plot(Lc)
    Lc
}
```

LorenzCurve2

```
# Lorenz Curve function that uses predicted values for sorting
LorenzCurve2 <- function (x, n = rep(1, length(x)), xhat, plot = FALSE)
{
      # Modification of Lc function in ineq package
      # x = actual (frequency, severity, loss cost)
      # xhat = predicted (frequency, severity, loss cost)
      # n = weights (exposure, claims, exposure)
      # data will be sorted in ascending order of prediction
      # Lorenz Curve will be plotted based on actual values
      ina <- !is.na(x)
      n <- n[ina]
      x <- as.numeric(x)[ina]
      xhat <- as.numeric(xhat)[ina]
      k <- length(x)
      # create random number sort key so ties will be resolved in random order
      #rkey <- runif(k)
      #o <- order(xhat, rkey)
      o <- order(xhat)
      x <- x[o]
      n <- n[o]
      x <- n * x
      p <- cumsum(n)/sum(n)
      L <- cumsum(x)/sum(x)
      p <- c(0, p)
      L <- c(0, L)
      L2 <- L * mean(x)/mean(n)
      Lc <- list(p, L, L2)
      names(Lc) <- c("p", "L", "L.general")
      class(Lc) <- "Lc"
      if (plot)
            plot(Lc)
      Lc
}
```

Calculation of lift charts

Lift charts and double lift charts are typically used to compare the relative performance of two or more models. One of the models is usually the one you are currently using in your production environment. The other model, sometimes called the challenger model, you hope will be a better model. Lift charts will give us a graphical way to compare the two models. For lift charts you create one chart per model and you can gain some valuable insights even if you only have a single model that you are working with. For double lift charts you create one chart per pair of models that you want to compare.

The procedure to calculate a lift chart for model "X" is as follows:

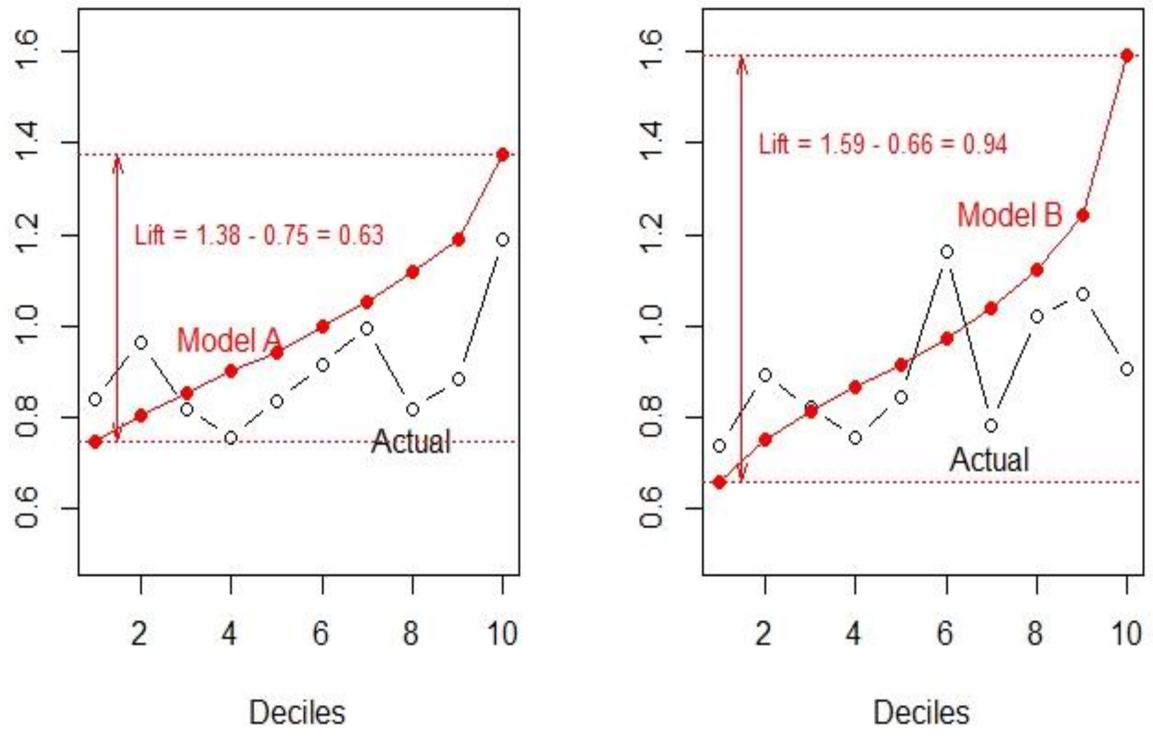
1. Calculate predicted values based on model "X" and sort the data from lowest to highest based on these predicted values.
2. Group your data in bins based on Step #1 and putting the same volume of exposure in each bin. Using five, ten, or twenty bins is typical.
3. For each bin, calculate the average predicted value and the average actual value.
4. Plot the results. On the horizontal axis you have the bins (from lowest to highest). On the vertical axis you have the average predicted value and the average actual value.

To compare model lift between two models, say "A" and "B," we apply the above procedure to each model and then place the charts side by side. We always calculate lift on holdout data.

Keep in mind that the records that make up each of the bins for the model "A" chart need not be the same as the records used in the model "B" chart. The bins in each chart depend on the predictions made by each model.

For each chart, the average predicted values increase as we move from left to right because we ordered the data from lowest to highest based on the predicted value. The actual average values need not follow this pattern, but if our model accurately predicts the actual values they would also increase (some small reversals in the pattern are fine) as we move from left to right. Large deviations would suggest that our model is not picking up something in the actual experience.

In the following graph we have the results for models "A" and "B."



In these graphs, we have normalized the plotted values (predicted and actuals) by dividing by the overall (across the entire holdout dataset) average predicted value.

On the left-hand side, we can see that model "A" average predicted value for bin #1 is about 0.75 of the overall average predicted value. At the other extreme, in bin #10, the average predicted value is about 1.38 times the overall average predicted value. Therefore, the lift that model "A" provides is equal to 0.63 = 1.38 – 0.75; namely, the distance between bin #1 and bin #10 is about 63% of the overall average predicted value. In practice, you will also see the lift of a model reported as the ratio of the highest bin to the lowest bin. In our example, this ratio would be 1.84 = 1.38 / 0.75 and we would say that our model provides a lift of 1.84.

Note that while our model somewhat tracks the actual experience the agreement is not very good (especially in bins 8, 9, and 10).

For model "B", shown on the left-hand panel, the lift is equal to 0.94 = 1.59 – 0.66. If we calculate lift as the ratio of highest bin to lowest bin we would get 2.41 = 1.59 / 0.66. Model "B" is able to separate the risks more than model "A." But also note that model "B" over predicts the actual experience in bins 7 through 10.

The R code to produce the above graphs is presented next.

```
#
# Using the car.csv data from the book
# Generalized Linear Models for Insurance Data
# by Piet de Jong and Gillian Heller
# http://www.acst.mq.edu.au/GLMsforInsuranceData/
#
dta <- read.csv("car.csv")
dta <- dta[,1:10]
dta$veh_age <- as.factor(dta$veh_age)
dta$agecat <- as.factor(dta$agecat)

# separate all records with positive claims cost
idx <- dta$claimcst0 > 0
dtb <- dta[idx,]
dtb$obs <- 1:(dim(dtb)[1])

rm(idx, dta)

# add random numbers and split data into training and holdout
set.seed(189763)
dtb$rnd <- runif(dim(dtb)[1])
idx <- dtb$rnd < 0.6
dt.train <- dtb[idx,]
dt.holdout <- dtb[!idx,]
rm(idx, dtb)

#
# Fit a simple gamma GLM with two predictors
# This will be our current base model
#
fit.base <- glm(claimcst0 ~ area + veh_age, data = dt.train,
                family = Gamma(link = "log"))
dt.holdout$base <- predict(fit.base, newdata = dt.holdout, type = "response")
#
# Fit a "superior" model by including a new variable
# Store our model predictions in the dataset
#
fit.A <- glm(claimcst0 ~ area + veh_age + gender, data = dt.train,
             family = Gamma(link = "log"))
dt.holdout$mod.A <- predict(fit.A, newdata = dt.holdout, type = "response")

#
# Calculate bins of equal exposure based on sorting
# by model A predictions
#
set.seed(13472)
ord <- order(dt.holdout$mod.A, runif(length(dt.holdout$mod.A)))
dt.holdout <- dt.holdout[ord,]
cum.expo <- cumsum(dt.holdout$exposure)
total.exposure <- sum(dt.holdout$exposure)
bks <- c(0, 1:9 * total.exposure/10, 10.2 * total.exposure/10)
dt.holdout$bin.A <- cut(cum.expo, breaks = bks, labels = 1:10)
dt.holdout <- dt.holdout[order(dt.holdout$obs),]
```

```r
rm(ord, cum.expo, total.exposure, bks)

# Check that summing exposure by bin number gives about
# equal exposure in each bin
tapply(dt.holdout$exposure, dt.holdout$bin.A, sum)

#
# Now let's calculate the lift that the new model provides
# compared to the current base model using deciles
#
avg.mod.A <- tapply(dt.holdout$mod.A,
                    dt.holdout$bin.A, mean)/mean(dt.holdout$mod.A)
avg.actual <- tapply(dt.holdout$claimcst0,
                     dt.holdout$bin.A, mean)/mean(dt.holdout$mod.A)

par(mar = c(4,4,1,1)+0.1)
lim = c(0.5, 1.65)
plot(x = 1:10, y = avg.actual, type = "b", ylim = lim,
     xlab = "Deciles", ylab = "")
points(x = 1:10, y = avg.mod.A, pch = 16, col = "red")
lines(x = 1:10, y = avg.mod.A, col = "red")
text(x = 8, y = avg.actual[8], labels = "Actual", pos = 1)
text(x = 4, y = avg.mod.A[4], labels = "Model A", pos = 3, col = "red")
abline(h = avg.mod.A[c(1,10)], lty = 3, col = "red")
arrows(x0 = 1.5, y0 = avg.mod.A[1], x1 = 1.5, y1 = avg.mod.A[10],
       code = 3, col = "red", angle = 15, length = 0.1)
text(x = 1.5, y = 1.2,
     labels = paste(c("Lift = ",
                      round(avg.mod.A[10],2), " - ",
                      round(avg.mod.A[1],2), " = ",
                      round(diff(avg.mod.A[c(1,10)]),2)),
                    collapse = ""),
     pos = 4, cex = 0.8, col = "red")

rm(avg.mod.A, avg.actual, lim)

#
# Fit a complete model by including all variables
#
fit.B <- glm(claimcst0 ~ area + veh_age + gender + agecat +
                         veh_value + veh_body,
             data = dt.train, family = Gamma(link = "log"))
dt.holdout$mod.B <- predict(fit.B, newdata = dt.holdout, type = "response")

#
# Calculate bins of equal exposure based on sorting
# by model B predictions
#
set.seed(13472)
ord <- order(dt.holdout$mod.B, runif(length(dt.holdout$mod.B)))
dt.holdout <- dt.holdout[ord,]
cum.expo <- cumsum(dt.holdout$exposure)
total.exposure <- sum(dt.holdout$exposure)
bks <- c(0, 1:9 * total.exposure/10, 10.2 * total.exposure/10)
```

```
dt.holdout$bin.B <- cut(cum.expo, breaks = bks, labels = 1:10)
dt.holdout <- dt.holdout[order(dt.holdout$obs),]

rm(ord, cum.expo, total.exposure, bks)

# Check that summing exposure by bin number gives about
# equal exposure in each bin
tapply(dt.holdout$exposure, dt.holdout$bin.B, sum)

#
# Now let's calculate the lift that the new model provides
# compared to the current base model using deciles
#
avg.mod.B <- tapply(dt.holdout$mod.B,
                    dt.holdout$bin.B, mean)/mean(dt.holdout$mod.B)
avg.actual <- tapply(dt.holdout$claimcst0,
                     dt.holdout$bin.B, mean)/mean(dt.holdout$mod.B)

par(mar = c(4,4,1,1)+0.1)
lim = c(0.5, 1.65)
plot(x = 1:10, y = avg.actual, type = "b", ylim = lim,
     xlab = "Deciles", ylab = "")
points(x = 1:10, y = avg.mod.B, pch = 16, col = "red")
lines(x = 1:10, y = avg.mod.B, col = "red")
text(x = 7, y = avg.actual[7], labels = "Actual", pos = 1)
text(x = 9, y = avg.mod.B[9], labels = "Model B", pos = 2, col = "red")
abline(h = avg.mod.B[c(1,10)], lty = 3, col = "red")
arrows(x0 = 1.5, y0 = avg.mod.B[1], x1 = 1.5, y1 = avg.mod.B[10],
       code = 3, col = "red", angle = 15, length = 0.1)
text(x = 1.5, y = 1.4,
     labels = paste(c("Lift = ",
                      round(avg.mod.B[10],2), " - ",
                      round(avg.mod.B[1],2), " = ",
                      round(diff(avg.mod.B[c(1,10)]),2)),
                    collapse = ""),
     pos = 4, cex = 0.8, col = "red")

rm(avg.mod.B, avg.actual, lim)
```

Calculation of double lift charts

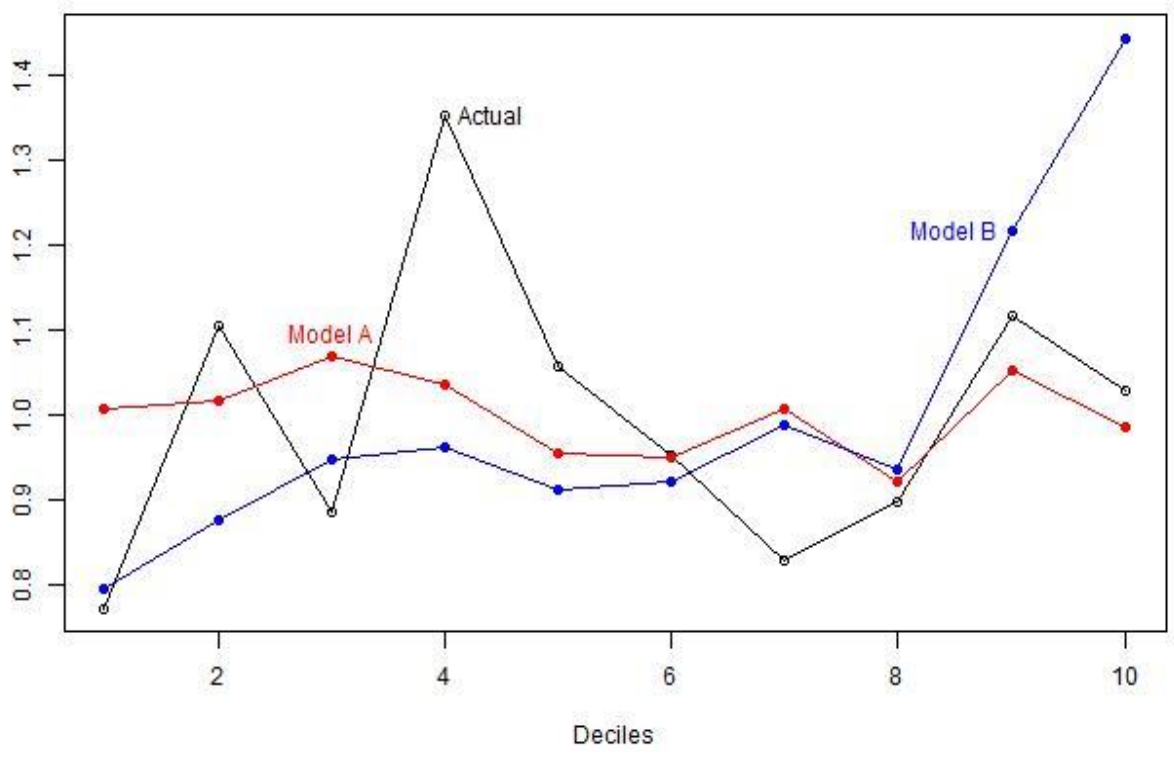
A common concern that arises during a predictive modeling engagement is where does a new model differ the most in its predictions relative to a current model and how do these models perform against actual experience. Double lift charts are helpful in this situation as the chart directly compares two models.

Suppose we have models “A” and “B.” To create a double lift chart do the following:

1. Using the predicted values from model “A” and model “B” calculate their ratio. We will use these values to sort the data from lowest ratio to largest ratio.
2. Create bins of equal volume exposure based on the ratio calculated in step #1.

3. For each bin, calculate the average predicted value based on model "A", the average predicted value based on model "B," and the average actual experience.
4. Plot the points calculated in the previous step.

You can normalize the values by dividing them by the overall predicted average. The following graph shows an example of a double lift chart:



In the above graph we calculated the ratio as model "B" divided by model "A". In the first bin, we have the records where this ratio is the smallest; that is, the records where model "A" predicts much larger relative to model "B." Similarly, the records that belong to bin #10 are those where model "B" predicts large values relative to the predictions of model "A." These two bins contain the records where these models disagree the most.

Notice that in the above graph, model "A" is mostly flat and 'far away' from the actual experience (except for bins 6 and 8). Model "B" seems to track actuals a bit better. The volatility of actual experience makes it difficult to pick a clear better performing model. On the lower bins (1, 2, and 3) model "B" seems better. On the higher bins (8, 9, and 10) model "A" seems better.

The R code to calculate the double lift chart is:

```
#
# Double lift chart comparing models A and B
#
s.ratio <- dt.holdout$mod.B / dt.holdout$mod.A
summary(s.ratio)
bks <- c(0.1,
         quantile(s.ratio, probs = seq(0, 1, length = 11))[-c(1,11)], 2.5)
ct <- cut(s.ratio, breaks = bks, labels = 1:10)
avg.mod.A <- tapply(dt.holdout$mod.A, ct, mean)/mean(dt.holdout$mod.A)
avg.mod.B <- tapply(dt.holdout$mod.B, ct, mean)/mean(dt.holdout$mod.B)
avg.actual <- tapply(dt.holdout$claimcst0, ct, mean)/mean(dt.holdout$claimcst0)

par(mar = c(4,4,1,1)+0.1)
lim <- range(c(avg.actual, avg.mod.A, avg.mod.B))
plot(x = 1:10, y = 1:10,
     type = "n", ylim = lim, xlab = "Deciles", ylab = "")

points(x = 1:10, y = avg.actual, pch = 1)
lines(x = 1:10, y = avg.actual)
text(x = 4, y = avg.actual[4], labels = "Actual", pos = 4)

points(x = 1:10, y = avg.mod.A, col = "red", pch = 16)
lines(x = 1:10, y = avg.mod.A, col = "red")
text(x = 3, y = avg.mod.A[3], labels = "Model A", pos = 3, col = "red")

points(x = 1:10, y = avg.mod.B, col = "blue", pch = 16)
lines(x = 1:10, y = avg.mod.B, col = "blue")
text(x = 9, y = avg.mod.B[9], labels = "Model B", pos = 2, col = "blue")

rm(s.ratio, bks, ct, avg.mod.A, avg.mod.B, avg.actual, lim)
```

The Receiver Operating Characteristic Curve

For a logistic model the prediction is a probability that the event under study occurs. For example, if we are looking at policy quotations we may want to study the conversion to a sold policy. In this situation, we may build a model that would predict the probability that a quote converts to a sold policy. With this information, we may want to take specific actions and in order to do this we typically select a threshold value. Having selected a threshold value for our model we can classify each record in our data into four categories:

		Model Prediction	
		POSITIVE	**NEGATIVE**
Actual Observed Value	**TRUE**	True Positive	False Negative
	FALSE	False Positive	True Negative

The above table is named the confusion matrix. In two of the four entries, our model's predictions are accurate; namely, the cells labeled true positive and true negative. For the other two entries, namely, false positive and false negative, our model gets confused and provides erroneous predictions.

There are a few simple calculations that we can do with the entries in the confusion matrix that are important. The ratio of true positive to the total actual true records is called the sensitivity or the true positive rate. The ratio of true negative to total actual negative records is called the specificity. Note that one minus specificity is called the false positive rate.

These ratios clearly depend on the choice of threshold. As we vary the threshold from zero to one, the specificity and sensitivity of our classifier change and the receiver operating characteristic curve is a graphical summary of these measures across all possible thresholds.

To illustrate the construction of a receiver operating characteristic curve we have fitted a logistic model to some data. Using a threshold equal to 10% we get the following confusion matrix:

		Predicted	
		POSITIVE	**NEGATIVE**
Actual	**TRUE**	244	1
	FALSE	226	73

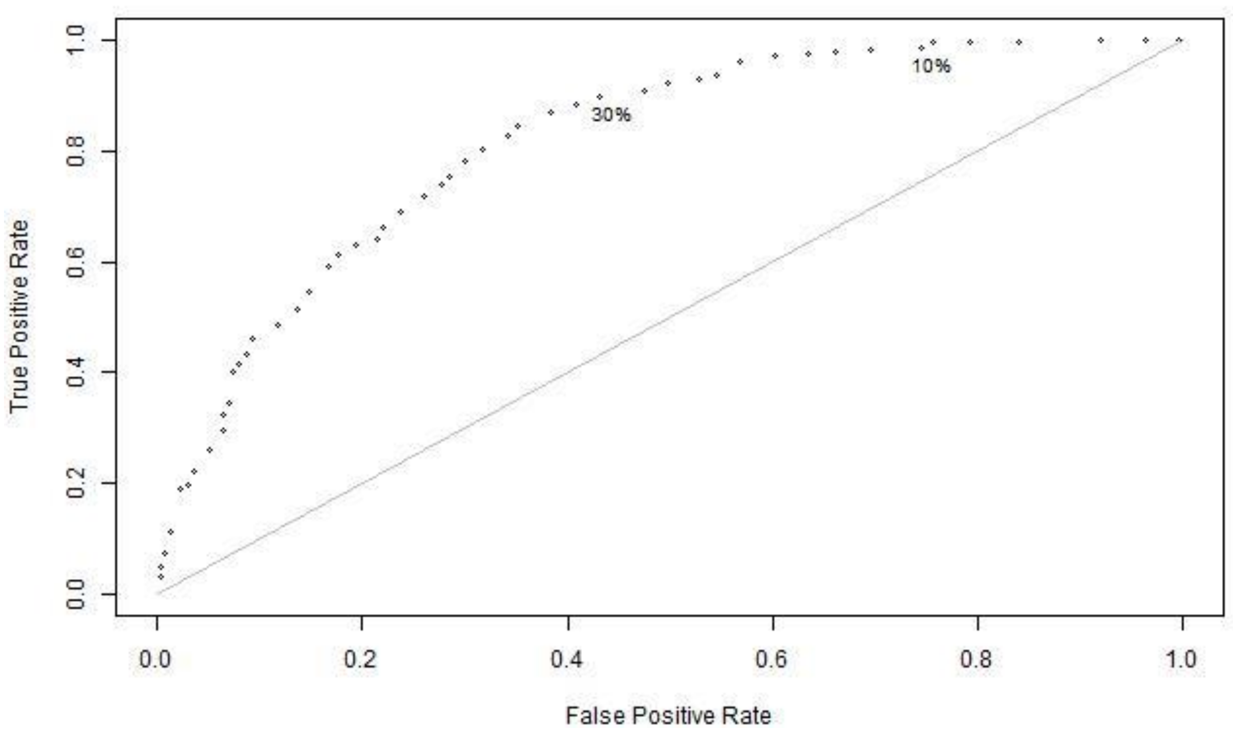
The true positive rate (sensitivity) is equal to 244 / (244 + 1) = 0.996 and the false positive rate (1 – specificity) = 226 / (226 + 73) = 0.756.

Using a threshold of 30% we get the following confusion matrix:

		Predicted	
		POSITIVE	**NEGATIVE**
Actual	**TRUE**	222	23
	FALSE	133	166

The true positive rate (sensitivity) is equal to 222 / (222 + 23) = 0.906 and the false positive rate (1 – specificity) = 133 / (133 + 166) = 0.445.

These computations give us two points (see graph below) to plot in our ROC curve. Doing 50 thresholds between 0 and 1 gives the following graph:



The best possible model would have a sensitivity of 1 and a specificity of 1 also. The plotted point in the ROC curve would be located in the upper left-hand corner. Better models try to reach that point. Models that are no better than random guessing would produce a ROC curve that stays close to the diagonal line (see gray line) from (0,0) to (1,1).

Typically, one also computes the area under the ROC curve; usually denoted as AUC. In the above example, the area under the curve is equal to about 0.818. A perfect model would have an area under the curve of 1 and a random guessing model (area under the diagonal line) would have an AUC equal to 0.5. The area under the curve gives you a summary of the model across all thresholds.

Note that this ROC graph has been calculated on the training data used to fit the model. Therefore, this ROC graph is overly optimistic. Candidates are urged to compare this ROC graph with one using holdout data. Within the dataset, the variable `STUDY` has four unique levels (`development, indiana, n172, and eortc/mrc`). We used the `development` subset to train our model. Candidates can use the `indiana` subset as a holdout sample.

The R code to generate the above graph is:

```
library(foreign)
library(rms)

# Set to TRUE to generate the graph as a JPEG file
generate.file <- FALSE

#
# Using the testicular cancer (t821.sav) data from the book
# Clinical Prediction Models
# by Ewout W. Steyerberg
# http://www.clinicalpredictionmodels.org/
#
tmp <- read.spss("t821.sav", to.data.frame = TRUE)
# The data t821.sav is in SPSS format and reading it will
# throw a warning message about undeclared levels in the
# HOSP variable.  Please ignore it as we are not using
# this variable for this example.
n544 <- tmp[tmp$STUDY == "development",]

#
# Set the vector Y to Yes or No depending
# on whether the patient has benign tissue
#
Y <- rep("No", 544)
Y[n544$NEC == 1] <- "Yes"
Y <- as.factor(Y)

full <- lrm(NEC ~ TER + PREAFP + PREHCG + SQPOST + REDUC10,
            data = n544,
            x = TRUE,
            y = TRUE)

probs <- predict(full, type = "fitted.ind")

thresholds <- seq(0.005, 0.93, length = 50)
N <- length(thresholds)
TPR <- numeric(N) # true positive rates
FPR <- numeric(N) # false positive rates
for(i in 1:N){
  mod.pred <- rep("N", 544)
  mod.pred[probs > thresholds[i]] <- "Y"
  tb <- table(mod.pred, Y)[2:1,2:1] #confusion matrix
  TPR[i] <- tb[1,1] / (tb[1,1]+tb[2,1])
  FPR[i] <- tb[1,2] / (tb[1,2]+tb[2,2])
}

if(generate.file)
  jpeg(filename = "roc-example.jpeg", width = 640, height = 380)
op <- par(mar = c(4,4,1,1))
plot(x = FPR, y = TPR,
     type = "p", pch = 1, cex = 0.5,
     xlim = c(0,1), ylim = c(0,1),
     ylab = "True Positive Rate", xlab = "False Positive Rate")
lines(x = c(0,1), y = c(0,1), col = "gray")
text(y = 0.996, x = 0.756, label = "10%", cex = 0.8, pos = 1)
text(y = 0.906, x = 0.445, label = "30%", cex = 0.8, pos = 1)
if(generate.file) dev.off()

#
# Calculate an approximation to the
# area under the curve (AUC) using
# mid-point as the height of the
# approximating rectangles
#
FPR <- rev(FPR)
TPR <- rev(TPR)
rect.base <- (FPR[-1] - FPR[-length(FPR)])
AUC <- (sum(rect.base * TPR[-1]) + sum(rect.base * TPR[-length(TPR)]))/2
```