Business Data Mining (IDS 572)

Model Evaluation

Once we have a set of prediction models, various metrics can be used to evaluate their performance.

For regression models:

- R^2 is very popular. In many complex models, the notion of the model degrees of freedom is difficult. Unadjusted R^2 can be used, but does not penalize complexity. (caret:::RMSE, pls:::RMSEP)
- the root mean square error is a common metric for understanding the performance (caret:::Rsquared, pls:::R2)
- Spearman's correlation may be applicable for models that are used to rank samples (cor(, method = "spearman"))

Of course, honest estimates of these statistics cannot be obtained by predicting the same samples that were used to train the model.

A test set and/or resampling can provide good estimates.

For classification models:

- Overall accuracy can be used, but this may be problematic when the classes are not balanced.
- the Kappa statistic takes into account the expected error rate:

$$\kappa = \frac{O - E}{I - E}$$

  where O is the observed accuracy and E is the expected accuracy under chance agreement (psych:::cohen.kappa, vcd:::Kappa, ...)
- For 2–class models, Receiver Operating Characteristic (ROC) curves can be used to characterize model performance.

Estimating performance for classification

1. Confusion matrix
   A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.

- R functions for confusion matrices are in the e1071 package (the classAgreement function), the caret package (confusionMatrix), the mda (confusion) and others.
- Below we draw the confusion matrix for the rpart decision tree model on the iris dataset using the "caret" package.

>library(caret)
> # get table of results
> table(predict(iris.rpart, type = “class”, newdata = irisTest), irisTest$Species)

```
            setosa versicolor virginica
setosa          10          0         0
versicolor       0         12         2
virginica        0          0        14
```

> #get results from confusion matrix
> # the arguments in this functions are confusionMatrix(data, reference, positive = NULL, dnn = c("Prediction", Reference"), ...); data takes the predicted values; reference takes the actual values; If there are only two factor levels, the first level will be used as the "positive" result; and dnn gives name for rows and columns.
> result = confusionMatrix(predict(iris.rpart, type = “class”, newdata = irisTest), irisTest$Species)

```
Confusion Matrix and Statistics

            Reference
Prediction   setosa versicolor virginica
  setosa         10          0         0
  versicolor      0         12         2
  virginica       0          0        14

Overall Statistics

               Accuracy : 0.9474
                 95% CI : (0.8225, 0.9936)
    No Information Rate : 0.4211
    P-Value [Acc > NIR] : 7.335e-12

                  Kappa : 0.9202
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: setosa Class: versicolor Class: virginica
Sensitivity                 1.0000            1.0000           0.8750
Specificity                 1.0000            0.9231           1.0000
Pos Pred Value              1.0000            0.8571           1.0000
Neg Pred Value              1.0000            1.0000           0.9167
Prevalence                  0.2632            0.3158           0.4211
Detection Rate              0.2632            0.3158           0.3684
Detection Prevalence        0.2632            0.3684           0.3684
Balanced Accuracy           1.0000            0.9615           0.9375
```

Look at the link below to get the definition of each out-come
http://artax.karlin.mff.cuni.cz/r-help/library/caret/html/confusionMatrix.html

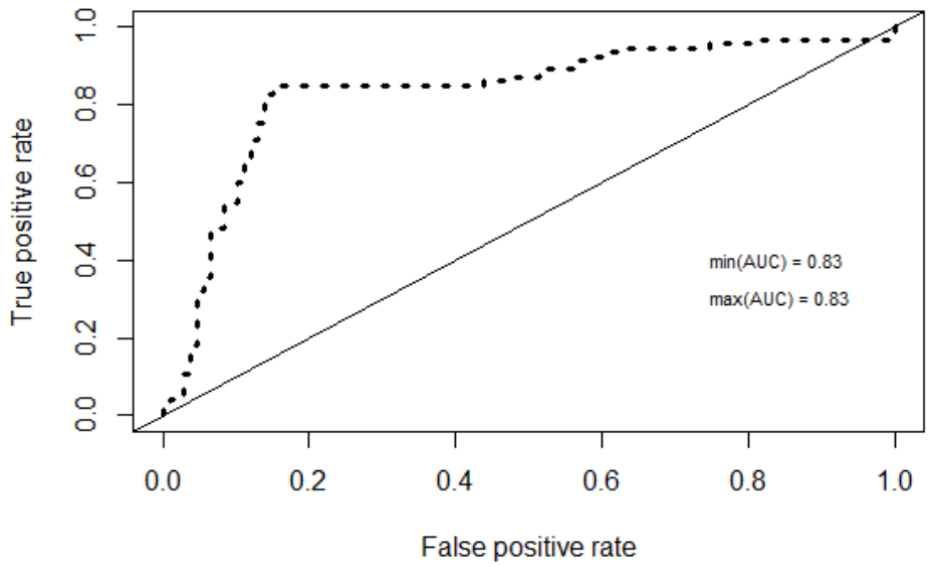
2. ROC Curve

A receiver operating characteristic (ROC), or ROC curve, is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. The curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

ROC curve functions are found in the ROCR package (performance), the verification package (roc.area), the pROC package (roc) and others.

```
> # Install package
> install.packages("ROCR")
> # Computing a simple ROC curve (x-axis: fpr, y-axis: tpr)
> library(ROCR)
> data(ROCR.simple)
> # Calculating the values for ROC curve
> pred = prediction( ROCR.simple$predictions, ROCR.simple$labels)
> perf = performance(pred,"tpr","fpr")
> # Plotting the ROC curve
> plot(perf, col = “black”, lty = 3, lwd = 3)

> # Calculating AUC
> auc = performance(pred, “auc”)
> # Now converting S4 class to a vector
> auc = unlist(slot(auc, "y.values"))
> # Adding min and max ROC AUC to the center of the plot
> minauc = min(round(auc, digits = 2))
> maxauc = max(round(auc, digits = 2))
> minauct = paste(c(“min(AUC) = “), minauc, sep = “”)
> maxauct = paste(c(“max(AUC) = “), maxauc, sep = “”)
> legend(0.7, 0.5, c(minauct, maxauct, “\n”), border = “white”, cex = 0.7, box.col = “white”)
>abline(a= 0, b=1)
```



At every cutoff, the TPR and FPR are calculated and plotted. The smoother the graph, the more cutoffs the predictions have. We also plotted a 45-degree line, which represents, on average, the performance of a Uniform(0, 1) random variable. The further away from the diagonal line, the better. Overall, we see that we see gains in sensitivity (true positive rate, > 80%), trading off a false positive rate (1- specificity), up until about 15% FPR. After an FPR of 15%, we don't see significant gains in TPR for a tradeoff of increased FPR.

Getting an "optimal" cut point

In some applications of ROC curves, you want the point closest to the TPR of one and FPR of zero. This cut point is "optimal" in the sense it weighs both sensitivity and specificity equally. To determine this cutoff, you can use the code below. The code takes in BOTH the performance object and prediction object and gives the optimal cutoff value of your predictions:

```
>opt.cut = function(perf, pred){
  cut.ind = mapply(FUN=function(x, y, p){
  d = (x - 0)^2 + (y-1)^2
  ind = which(d == min(d))
  c(sensitivity = y[[ind]], specificity = 1-x[[ind]],
  cutoff = p[[ind]])
  }, perf@x.values, perf@y.values, pred@cutoffs)}
>print(opt.cut(roc.perf, pred))
```

Now, there is a cost measure in the ROCR package that you can use to create a performance object. If you use it to find the minimum cost, then it will give you the same cutoff as opt.cut, but not give you the sensitivity and specificity.

```
>cost.perf = performance(pred, "cost")
>pred@cutoffs[[1]] [which.min(cost.perf@y.values[[1]])]
```

Different costs for FP and FN

The output from opt.cut and a performance object with measure cost are NOT equivalent if false positives and false negatives are not weighted equally. The cost.fn and cost.fp arguments can be passed to performance, corresponding to the cost of a false negative and false positive, respectively. Let's say false positives are twice as costly as false negatives, and let's get a cut point:

```
> cost.perf = performance(pred, "cost", cost.fp = 2, cost.fn = 1)
> pred@cutoffs[[1]] [which.min(cost.perf@y.values[[1]])]
```

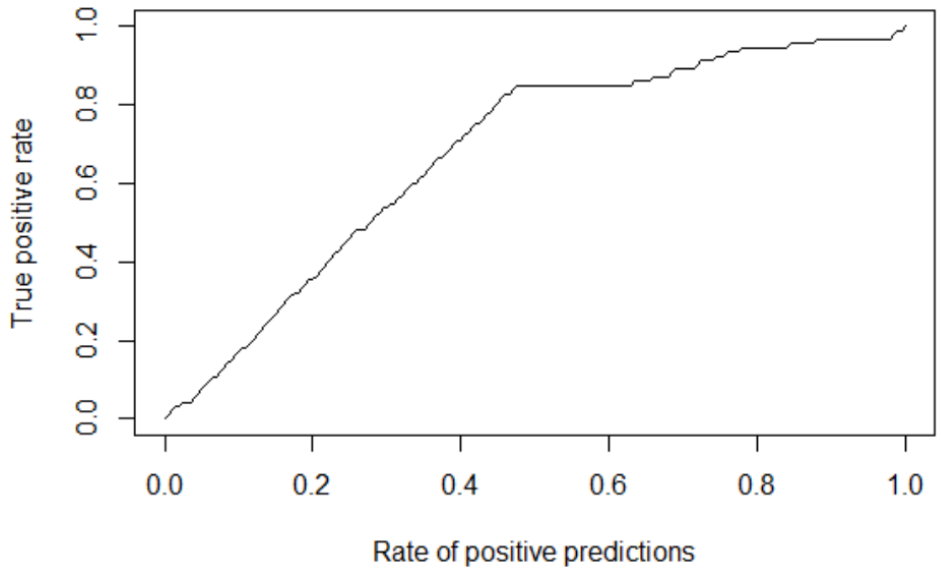
Note that R has many options within the performance function. I've listed only a few of them below. For more see: ?performance

- acc: Accuracy. Estimated as: (TP+TN)/(P+N).
- err: Error rate. Estimated as: (FP+FN)/(P+N).
- fpr: False positive rate. Estimated as: FP/N.
- tpr: True positive rate. Estimated as: TP/P.
- sens: Sensitivity. Same as tpr.
- fnr: False negative rate. Estimated as: FN/P.
- tnr: True negative rate. Estimated as: TN/N.
- spec: Specificity. Same as tnr.
- ppv: Positive predictive value. Estimated as: TP/(TP+FP).
- prec: Precision. Same as ppv.
- npv: Negative predictive value. Estimated as: TN/(TN+FN).
- rpp: Rate of positive predictions. Estimated as: (TP+FP)/(TP+FP+TN+FN).
- rnp: Rate of negative predictions. Estimated as: (TN+FN)/(TP+FP+TN+FN).
- odds: Odds ratio. (TP*TN)/(FN*FP). Note that odds ratio produces Inf or NA values for all cutoffs corresponding to FN=0 or FP=0. This can substantially decrease the plotted cutoff region.

- auc: Area under the ROC curve. This is equal to the value of the Wilcoxon-MannWhitney test statistic and also the probability that the classifier will score are randomly drawn positive sample higher than a randomly drawn negative sample. Since the output of auc is cutoff-independent, this measure cannot be combined with other measures into a parametric curve. The partial area under the ROC curve up to a given false positive rate can be calculated by passing the optional parameter fpr.stop=0.5 (or any other value between 0 and 1) to performance.

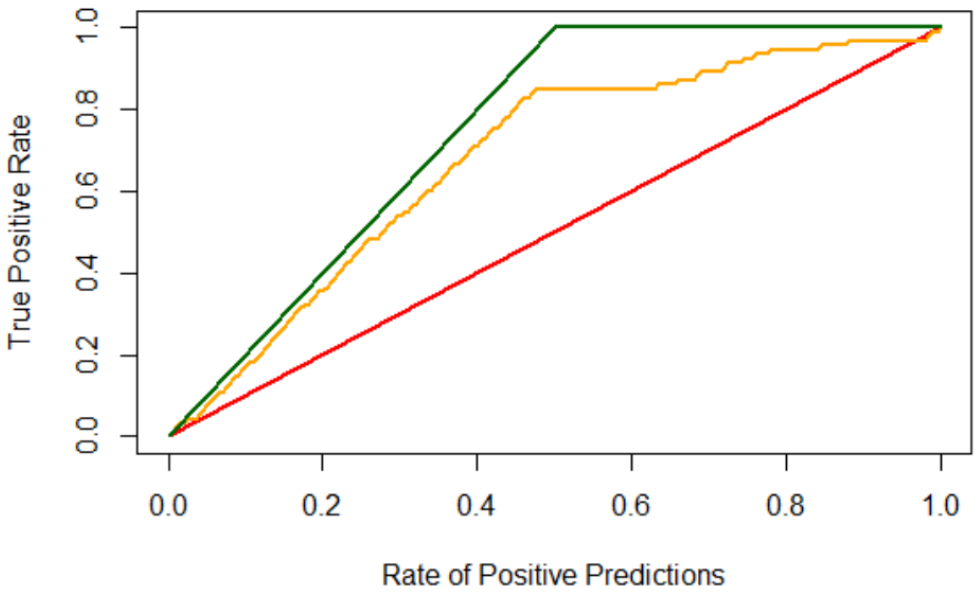
3. Gain chart
   Gain chart shows sensitivity versus rate of positive predictions.

```
> library(ROCR)
>data(ROCR.simple)
>pred = prediction( ROCR.simple$predictions, ROCR.simple$labels)
>perf = performance(pred,"tpr", "rpp")
>plot(perf)
```



We can also add the base line and idea line to the graph

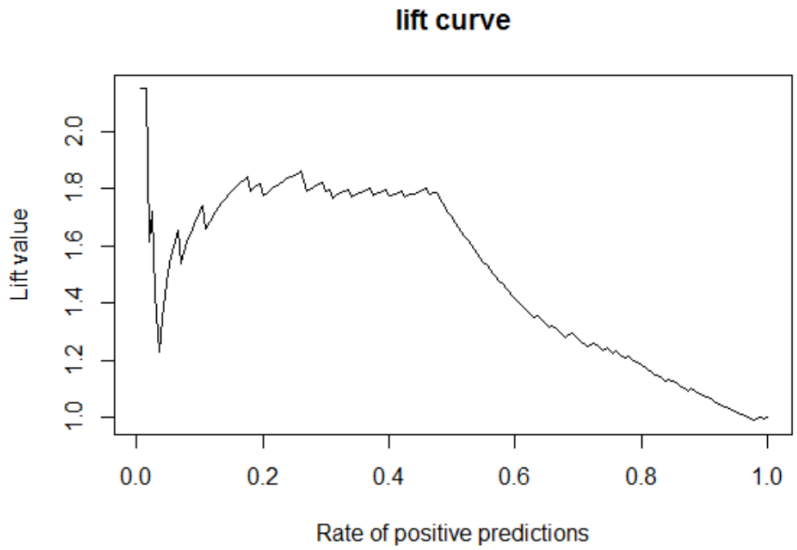
```
> plot(x=c(0, 1),y=c(0, 1),type="l",col="red",lwd=2,ylab="True Positive Rate", xlab="Rate of Positive Predictions")
> lines(x=c(0, 0.5, 1), y=c(0, 1, 1), col="darkgreen", lwd=2)
> gain.x = unlist(slot(perf, 'x.values'))
> gain.y = unlist(slot(perf, 'y.values'))
> lines(x=gain.x, y=gain.y, col="orange", lwd=2)
```



4. Lift chart

   Lift is a measure of the effectiveness of a predictive model calculated as the ratio between the results obtained with and without the predictive model.

```
> perf = performance(pred,"lift","rpp")
> plot(perf, main="lift curve")
```



You can also use “lift” package to draw the lift chart

```
> install.packages("lift")
> library(lift)
```

```
> plotLift(ROCR.simple$predictions, ROCR.simple$labels, n.buckets = 100)
```