**Data Analytics (CMP330)**

# Practical 9 – Evaluating Model Performance

Review the lecture slides for week 9.

**Exercise 1. Implement the holdout example on slides 10 and 11 of the lecture slides and make sure you understand what is going on. You should use the non-stratified sampling approach on slide 11 to make predictions and generate a confusion matrix and then compare with the stratified approach on slide 10.**

**Solution**

To implement the approach on slide 11 replace:

**trainIndex <- createDataPartition(iris$Species, p=split, list=FALSE)**

on slide 10 with

**trainIndex <- sample(1:nrow(iris),size=nrow(iris)\*0.8,replace = FALSE)**

and then proceed as on slide 10.

Before going further, you should complete the week 8 lab. The material below follows on from the end of the week 8 lab, so you will need to have run the code to create the **credit\_model** using C5.0 and then used the predict function to generate **credit\_pred**.

The Classification and Regression Training package **caret** by Max Kuhn includes functions for computing many such performance measures. This package provides a large number of tools for preparing, training, evaluating, and visualizing machine learning models and data. You will need to install the package using the install.packages("caret") command.

**library(caret)**

**confusionMatrix(credit\_pred, credit\_test$default)**

At the top of the output is a confusion matrix. The output also includes a set of performance measures.

The ROCR package provides an easy-to-use set of functions for creating ROC curves and computing AUC. The ROCR website

<https://cran.r-project.org/web/packages/ROCR/index.html>

includes a list of the full set of features, as well as several examples of the visualization capabilities. Before continuing, be sure that you have installed the package using the

**install.packages("ROCR").**

**library(ROCR)**

To create visualizations with ROCR, two vectors of data are needed. The first must contain the estimated probability of the positive class and the second must contain the predicted class values. We will use **credit\_model** to generate estimated probabilities (probability of “yes”) which are then used along with the actual class labels to produce the ROC curve as follows:

**credit\_pred\_prob <- predict(credit\_model, credit\_test, type="prob")**

**credit\_pred\_prob**

**credit\_pred\_prob <- as.data.frame(credit\_pred\_prob)**

**credit\_roc <- data.frame(credit\_pred\_prob$yes,credit\_test$default=='yes')**

**colnames(credit\_roc) <- c("predict", "label")**

**credit\_roc**

**pred\_roc <- prediction(credit\_roc$predict, credit\_roc$label)**

**perf <- performance(pred\_roc, measure = "tpr", x.measure = "fpr")**

Using the perf object, we can visualise the ROC curve with R’s plot() function.

**plot(perf, main = "ROC curve for credit classification", col = "blue", lwd = 2)**

**# add a reference line to the graph**

**abline(a = 0, b = 1, lwd = 2, lty = 2)**

To calculate AUC, we use performance function().

**perf.auc <- performance(pred\_roc, measure = "auc")**

**str(perf.auc)**

**as.numeric(perf.auc@y.values)**

This result is AUC=0.7464. According to the guide for AUC values, the performance is acceptable (AUC: 0.7-0.8).

**Exercise 2. Implement the cross validation (CV) approach on slide 16 of the lectures. Then use the same approach to apply CV to the credit problem in Part B of practical 8. Step 1 of part B would be the same as before, but then you should apply CV by adapting the code on slide 16. (Note: you will see results for different tuning parameters – more on this below). How does the accuracy of the selected model compare with the result obtained earlier without CV? (See the metrics displayed with the confusion matrix.) See if you can display the accuracy for each of the 10 folds and then calculate the mean and standard deviation of the accuracy over the 10 folds.**

**Solution**

# define training control

**train\_control <- trainControl(method="cv", number=10)**

# train the model

**model <- train(default~.,data=credit, trControl=train\_control, method="C5.0")**

# summarize results

**print(model)**

**print(model$resample)**

**mean(model$resample$Accuracy)**

**sd(model$resample$Accuracy)**

**Cross Validation for Parameter Tuning**

The caret package, provides tools to assist with automated parameter tuning.

Use the modelLookup() function to find the tuning parameters. For example, check the tuning parameters for C5.0 model.

**modelLookup(“C5.0”)**

Customizing the tuning process

The default settings allow optimized models to be created easily. Each step in the model selection process can be customized.

The trainControl() function is used to create a set of configuration options known as a control object. This object guides the train() function and allows for the selection of model evaluation criteria.

To create a control object named ctrl that uses 10-fold CV and the oneSE selection function (it selects the simplest model whose mean falls within one standard error of the best result), use the following command (note that number =10 is included only for clarity; since this is the default value for method="cv", it could have been omitted):

**ctrl <- trainControl(method="cv",number=10,selectionFunction = 'oneSE')**

In the meantime, the next step in defining our experiment is to create the grid of parameters to optimize. The grid must include a column named for each tuning parameter in the desired model. It must also include a row for each desired combination of parameter values. Since we are using a C5.0 decision tree, this means we will need columns named model, trials, and winnow.

**grid <- expand.grid(model="tree",trials=c(1,5,10,15,20,25,30,35),winnow=FALSE)**

The train() function will build a candidate model for evaluation using each row’s combination of model parameters.

**m <- train(default ~ ., data = credit, method = "C5.0",metric = "Kappa", trControl = ctrl,tuneGrid = grid)**

**m**

visualisation of the model:

# plot the effect of parameters on Kappa

**plot(m)**