MODEL EVALUATION

E. Fersini

INTRODUCTION

- To measure the performance of a classification model, we can first generate a table based on our predicted label and actual label. Then, we can use a confusion matrix to obtain performance measures such as precision, recall, specificity and accuracy.
- During this lab recipe, we will demonstrate how to retrieve a confusion matrix using the caret package.
    - We will use the **churn dataset**

LOAD DATA AND PACKAGES

```
install.packages("C50")
library(C50)

install.packages(”caret")
library(caret)

data(churn)
churnTrain = churnTrain[,! names(churnTrain) %in% c("state”, "area_code",
"account_length") ]

set.seed(2)

ind = sample(2, nrow(churnTrain), replace = TRUE, prob=c(0.7, 0.3))
trainset = churnTrain[ind == 1,]
testset = churnTrain[ind == 2,]
```

COMPUTE PERFORMANCE

- Let's start training an SVM model

```
svm.model= train(churn ~ ., data = trainset, method = "svmRadial")
```

- We can then predict labels using the fitted model svm.model:

```
svm.pred = predict(svm.model, testset[,! names(testset) %in% c("churn")])
```

- We can now generate a classification table

```
table(svm.pred, testset[,c("churn")])
```

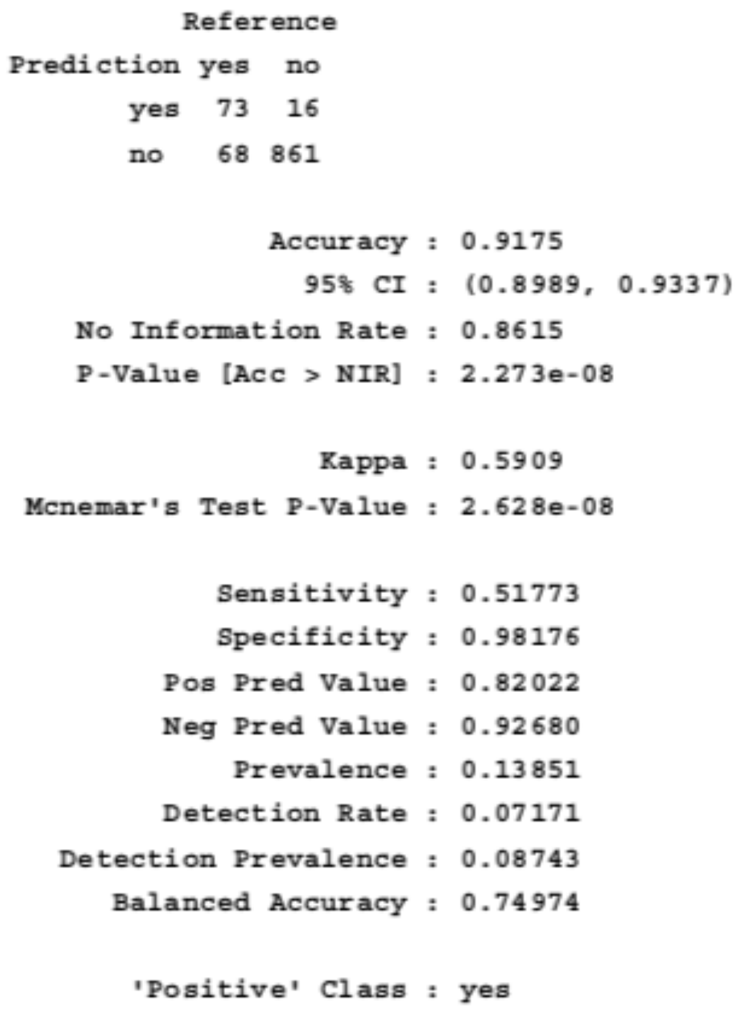
- Lastly, we can generate a confusion matrix using the prediction results and the actual labels from the testing dataset:

```
result = confusionMatrix(svm.pred, testset[,c("churn")])
```

- We can also require Precision, Recall and F1-Measure

```
result2 = confusionMatrix(svm.pred, testset[,c("churn")], mode = "prec_recall")
```

RESULTS



```
          Reference
Prediction yes  no
       yes  73  16
       no   68 861

               Accuracy : 0.9175
                 95% CI : (0.8989, 0.9337)
    No Information Rate : 0.8615
    P-Value [Acc > NIR] : 2.273e-08

                  Kappa : 0.5909
 Mcnemar's Test P-Value : 2.628e-08

            Sensitivity : 0.51773
            Specificity : 0.98176
         Pos Pred Value : 0.82022
         Neg Pred Value : 0.92680
             Prevalence : 0.13851
         Detection Rate : 0.07171
   Detection Prevalence : 0.08743
      Balanced Accuracy : 0.74974

       'Positive' Class : yes
```

For more than two classes, these results are calculated comparing each #' factor level to the remaining levels (i.e. a "one versus all" approach)

Global Measures (for the positive class = "yes")

If you want to look at the performance for the negative class, use the parameter

positive="no"

when calling the confusionMatrix function

ROC CURVE

- A **Receiver Operating Characteristic** (**ROC**) curve is a plot that illustrates the performance of a binary classifier system, and plots the true positive rate against the false positive rate for different cut points. We most commonly use this plot to calculate the **Area Under Curve** (**AUC**) AUC to measure the performance of a classification model. We will also demonstrate how to illustrate an ROC curve and calculate the corresponding AUC.

ROC CURVE

- Let's install and load the ROCR package

```
install.packages("ROCR")
library(ROCR)
```

- Train the svm model using the training dataset with a probability equal to TRUE:

```
svmfit=svm(churn~ ., data=trainset, prob=TRUE)
```

- Make predictions based on the trained model on the testing dataset with the probability set as TRUE:

```
pred=predict(svmfit,testset[, !names(testset) %in% c("churn")], probability=TRUE)
```

- Obtain the probability of labels with "yes":

```
pred.prob = attr(pred, "probabilities")
pred.to.roc = pred.prob[, 2]
```

ROC CURVE

- Use the prediction function to generate a prediction result:

```
pred.rocr = prediction(pred.to.roc, testset$churn)
```

- Use the performance function to obtain the performance measurement:

```
perf.rocr = performance(pred.rocr, measure = "auc", x.measure = "cutoff”)
perf.tpr.rocr = performance(pred.rocr, "tpr","fpr")
```

- Visualize the ROC curve using the plot function:

```
plot(perf.tpr.rocr, colorize=T,main=paste("AUC:",(perf.rocr@y.values)))
```

- Plot the random classifier

```
abline(a=0, b=1)
```

CUT-OFF

- In some applications of ROC curves, you want the point closest to the TPR of (1) and FPR of (0). This cut point is "optimal" in the sense it weighs both sensitivity and specificity equally. To deterimine the **optimal cutoff**:

```
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(perf.tpr.rocr, pred.rocr))
```

ACCURACY

- Another cost measure that is popular is **overall accuracy**. This measure optimizes the correct results, but may be skewed if there are many more negatives than positives, or vice versa. Let's get the overall accuracy for the simple predictions and plot it:

```
acc.perf = performance(pred.rocr, measure = "acc")
plot(acc.perf)
```

- What if we actually want to extract the **maximum accuracy** and the **cutoff** corresponding to that? In the performance object, we have the slot x.values, which corresponds to the cutoff in this case, and y.values, which corresponds to the accuracy of each cutoff. We'll grab the index for maximum accuracy and then grab the corresponding cutoff:

```
ind = which.max( slot(acc.perf, "y.values")[[1]] )
acc = slot(acc.perf, "y.values")[[1]][ind]
cutoff = slot(acc.perf, "x.values")[[1]][ind]
print(c(accuracy= acc, cutoff = cutoff))
```

COMPARING MODELS

- When it comes to the problem of how to choose the **best fitted model**, we need to compare all the performance measures we are interested in for all the classifiers we trained. To make the comparison easy, the caret package allows us to generate and compare the performance of models:

- Let’s install and load the ROCR package

```
install.packages("pROC ")
library(pROC)
```

- Set up the training control with a 10-fold cross-validation in 3 repetitions:

```
control = trainControl(method = "repeatedcv", number = 10,repeats = 3,
classProbs = TRUE, summaryFunction = twoClassSummary)
```

COMPARING MODELS

- Train a Support Vector Machine:

```
svm.model= train(churn ~ ., data = trainset, method = "svmRadial", metric =
"ROC", trControl = control)
```

- Train a Decision Tree:

```
rpart.model= train(churn ~ ., data = trainset, method = "rpart", metric =
"ROC", trControl = control)
```

- Make predictions for the trained models:

```
svm.probs = predict(svm.model, testset[,! names(testset) %in%  c("churn")],
type = "prob")
rpart.probs = predict(rpart.model, testset[,! names(testset) %in% c("churn")],
type = "prob")
```

COMPARING MODELS

- Generate the **ROC curve of each model**, and plot the curve on the same figure:

```
svm.ROC = roc(response = testset[,c("churn")], predictor =svm.probs$yes,
levels = levels(testset[,c("churn")]))
plot(svm.ROC,type="S", col="green")

rpart.ROC = roc(response = testset[,c("churn")], predictor =rpart.probs$yes,
levels = levels(testset[,c("churn")]))
plot(rpart.ROC, add=TRUE, col="blue")
```

- To compare the AUC:

```
svm.ROC
rpart.ROC
```

MEASURING DIFFERENCES BETWEEN MODELS

- We can also compare the **statistics** of the generated performance measure:

```
cv.values = resamples(list(svm=svm.model, rpart = rpart.model))
summary(cv.values)
```

- Use dotplot to plot the results in the ROC metric:

```
dotplot(cv.values, metric = "ROC")
```

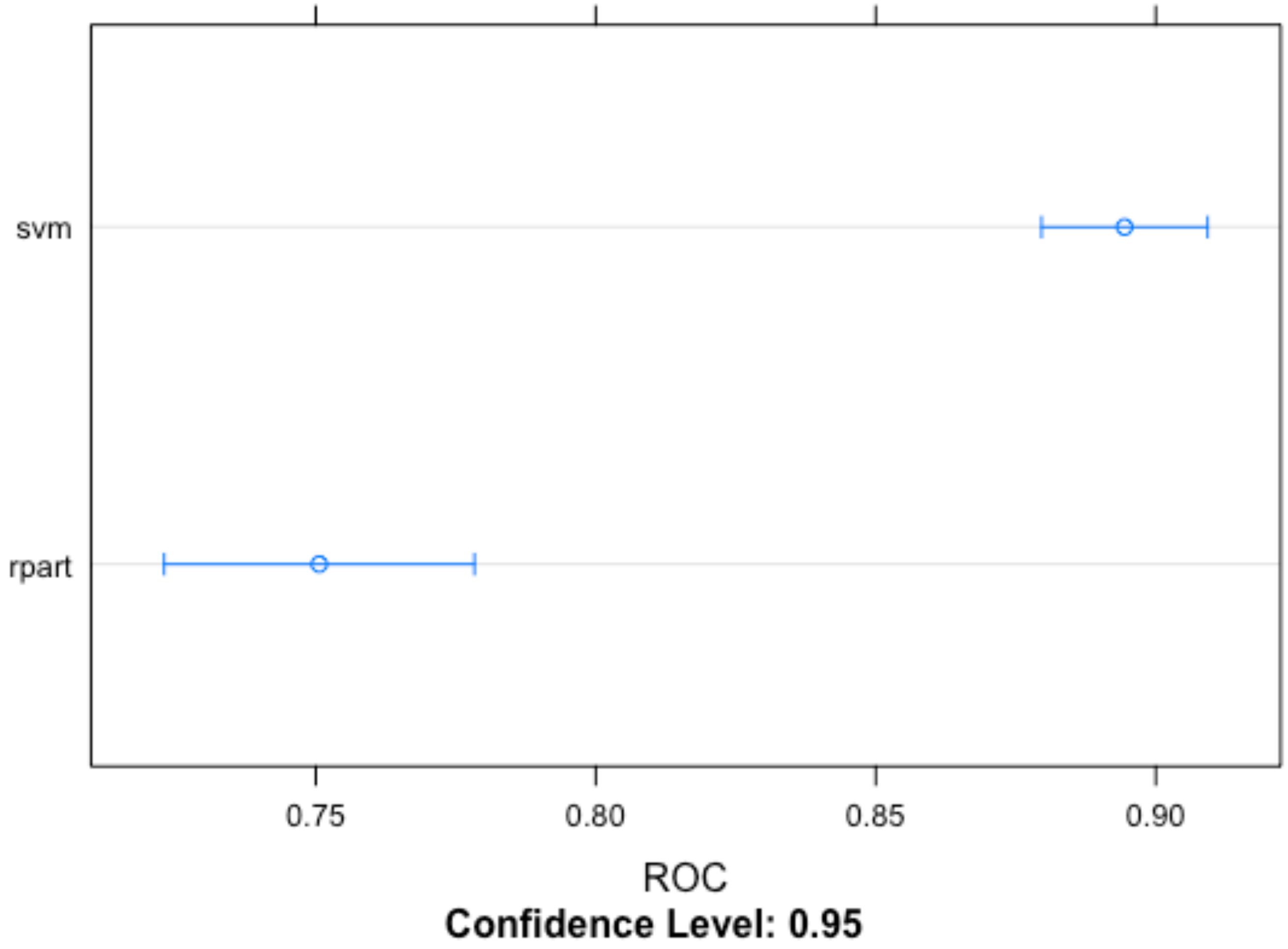
- Or the bwplot:

```
bwplot(cv.values, layout = c(3, 1))
```

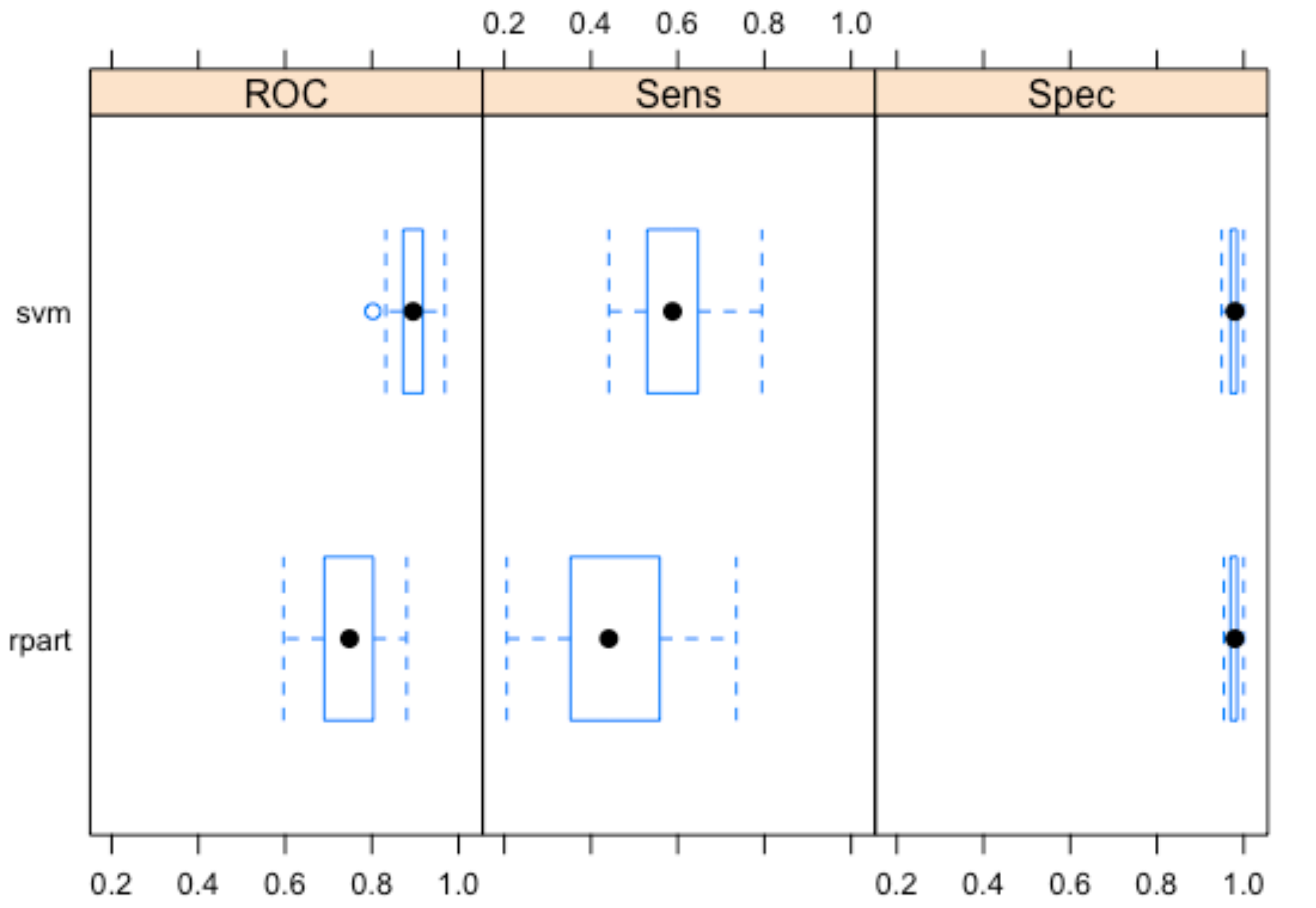
- Or the splom plot:

```
splom(cv.values,metric="ROC")
```

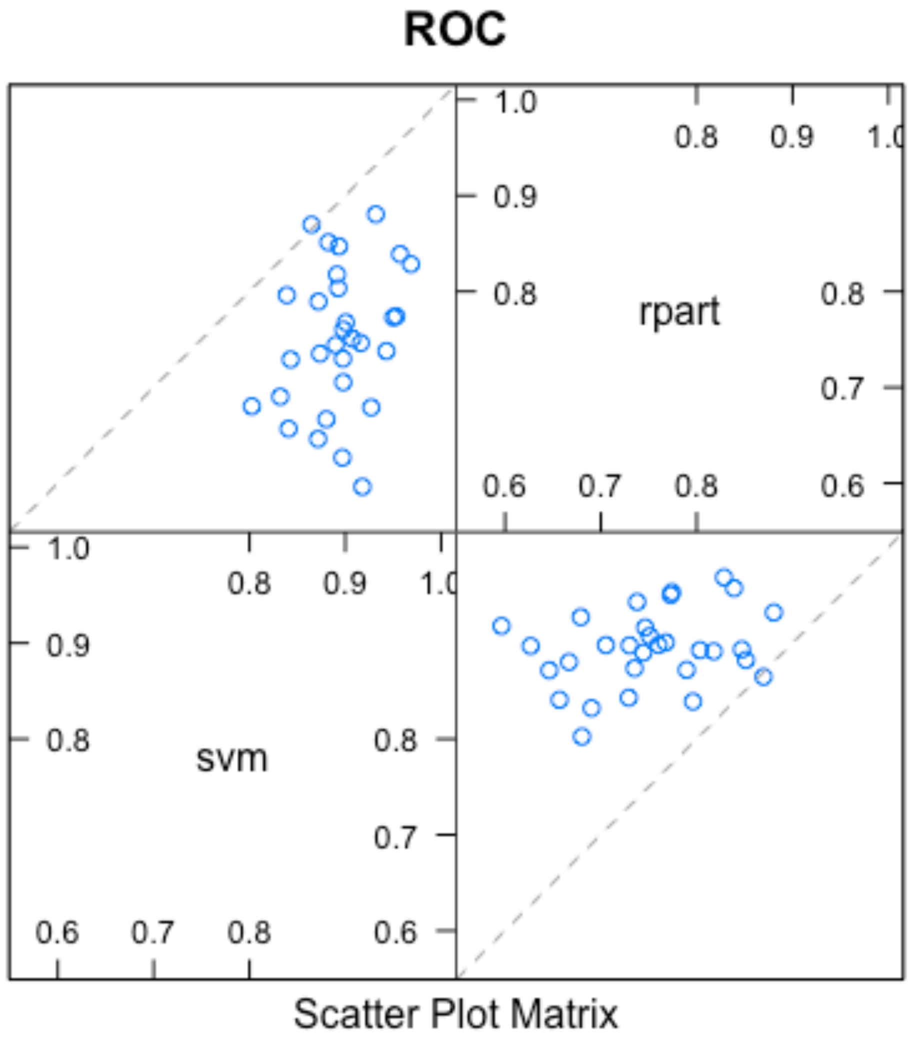
MEASURING DIFFERENCES BETWEEN MODELS



MEASURING DIFFERENCES BETWEEN MODELS



MEASURING DIFFERENCES BETWEEN MODELS



TIMINGS

- We can also take into account the **timings** required for training the models:

```
cv.values$timings
```

```
      Everything   FinalModel   Prediction
svm   38.427       0.420        NA
rpart  2.330       0.032        NA
```

ASSIGNMENT

1. Use the iris dataset for training two models (svm and decision tree)
2. Define your 10-fold cross validation function
3. Estimate the confusion matrix
4. Compute Precision, Recall, F1-Measure for each class label

For Multi-label problems, you can use the **multiRoc** package(*)

(*) https://mran.microsoft.com/snapshot/2018-02-12/web/packages/multiROC/vignettes/my-vignette.html