Week 6 Practical

Model Evaluation

Overview

In this Practical we will extend the discussion on model evaluation. Task 1 represents an overview of commonly used evaluation metrics that we need to understand. Through Tasks 2-4, we will show how those metrics can be calculated

Objectives

- Understand model evaluation metrics: accuracy, precision, recall, F1 score, Kappa, ROC, sensitivity, and specificity.
- Calculate accuracy, precision, recall, and F1 score using the confusion matrix.
- Calculate and plot ROC, accuracy and Kappa.
- Run cross validation.

Sharing results

While tasks should be self-explanatory, some of the challenges might be more complex. Feel free to share your results or questions in the course discussion forum. Results to all the challenges will be available UPON REQUEST (discussion forum or email).

Task 1. Evaluation metrics overview

Before going into practical use of some of the evaluation metrics, we will first introduce the most commonly used approaches to evaluate predictive models.

As you are aware of, when performing classification predictions, there are four types of outcomes that could occur:

- **True positives (TP)** – when our model predicts that an observation belongs to a class and it actually belongs to that class.
- **True negatives (TN)** – when our model predicts that an observation does not belong to a class and it actually does not belong to a class.
- **False positives (FP)** – when our model predicts that an observation DOES belong to a class when it DOES NOT.
- **False negatives (FN)** – when our model predicts that an observation DOES NOT belong to a class when it DOES.

Those four outcomes are usually plotted in a form of a confusion matrix. For example, Figure 1 shows a confusion matrix for the model we introduced in Practical 4. Please note that your results (if you decide to try this code) might be somewhat different. Also, *positives* in our case are labeled with "0", whereas *negatives* are labeled with "1"

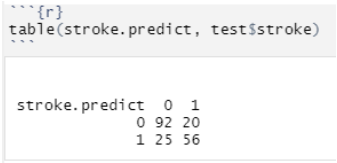


Figure 1. Confusion matrix for a decision tree model we introduced in Practical 4.

The four most commonly used metrics to evaluate a classification model are accuracy, precision, recall, *F1* score, sensitivity, and specificity.

Accuracy is defined as the percentage of correct predictions for the test data.

$$accuracy = \frac{correct\ predictions}{all\ predictions} = \frac{TP + TN}{TP + FP + TN + FN}$$

Precision is defined as the fraction of relevant examples (true positives) among all the examples which were predicted to belong to a certain class.

$$precision = \frac{true\ positives}{true\ positives + false\ positives} = \frac{TP}{TP + FP}$$

Recall is defined as the fraction of examples that were predicted to belong to a class with respect to all the examples that truly belong to the given class.

$$recall = \frac{true\ positives}{true\ positives + false\ negatives} = \frac{TP}{TP + FN}$$

F1 score is defined as a harmonic mean of precision and recall.

$$F1\ score = 2 * \frac{precision * recall}{precision + recall} = \frac{TP}{TP + \frac{1}{2}\ (FP + FN)}$$

As we discussed in Week 6 lecture, those metrics have certain limitations. Therefore, you will be up Kappa, Root Mean Squared Error (RMSE), R squared (R^2) and Area under ROC Curve to evaluate your models.

Kappa or Cohen's Kappa (as a most commonly used metric) is similar to accuracy, except it is normalized at the baseline of random chance on your dataset. It is more useful metric (than accuracy) when we have imbalanced classes.

If you use `caret` R package, default evaluation metrics used to evaluate algorithms will be RMSE and R^2. **RMSE** is the average deviation of the predictions from the observations, whereas **R^2** provides a "goodness of fit" measure for the predictions to the observations.

ROC is the area under the ROC curve or AUC and represents model's ability to discriminate between positive and negative classes. **It is suitable for binary classification problems only**. ROC can be broken down into sensitivity and specificity.

Sensitivity is defined as the fraction of positive predictions that are correctly identified.

$$sensitivity = \frac{true\ positives}{all\ positives} = \frac{TP}{TP + FN}$$

Specificity is defined as the fraction of negative predictions that are correctly identified.

$$specificity = \frac{true\ negatives}{all\ negatives} = \frac{TN}{TN + FP}$$

Task 2. Building decision tree and calculating accuracy, precision, recall, and F1 score

To explore various evaluation metrics in practice, we will use the code we had in Practical 4 to build a relatively straightforward decision tree and print confusion matrix.

Please make sure you run the code below.

```
# Load data
data <- read.csv(url("http://bit.ly/infs5100-stroke-data"))

nrow(data)
ncol(data)
head(data)
```

```
# Set variable types
data$stroke <- as.factor(data$stroke)
data$gender <- as.factor(data$gender)
data$hypertension <- as.factor(data$hypertension)
data$heart_disease <- as.factor(data$heart_disease)
data$ever_married <- as.factor(data$ever_married)
data$work_type <- as.factor(data$work_type)
data$residence_type <- as.factor(data$residence_type)
data$smoking_status <- as.factor(data$smoking_status)
data$bmi <- as.numeric(data$bmi)

# Impute missing BMI values
data.imputed <- mice(data, m=3, maxit = 50, method = 'pmm', seed = 500,
printFlag = FALSE)
data.complete <- complete(data.imputed, 1)

# Create training and test sets
set.seed(1000)
data.class <- data.complete[, c(-1)]

train_index <- sample(1:nrow(data.class), 0.8 * nrow(data.class))
test_index <- setdiff(1:nrow(data.class), train_index)
train <- data.class[train_index,]
test <- data.class[test_index,]
list( train = summary(train), test = summary(test) )

# Build full decision tree
c.tree.full <- rpart(stroke ~ ., train, method = "class", cp=0)

# Prune decision tree
p.tree.prune <- prune(c.tree.full, cp=
c.tree.full$cptable[which.min(c.tree.full$cptable[,"xerror"]),"CP"])

# Make a prediction
stroke.predict <- predict(p.tree.prune, test, type = "class")

# Print confusion matrix
table(stroke.predict, test$stroke)
```

When completed, the code above should provide a confusion matrix similar to the example in Figure 1.

Using the confusion matrix provided above, calculate:

a) **Accuracy,**

b) **Precision,**

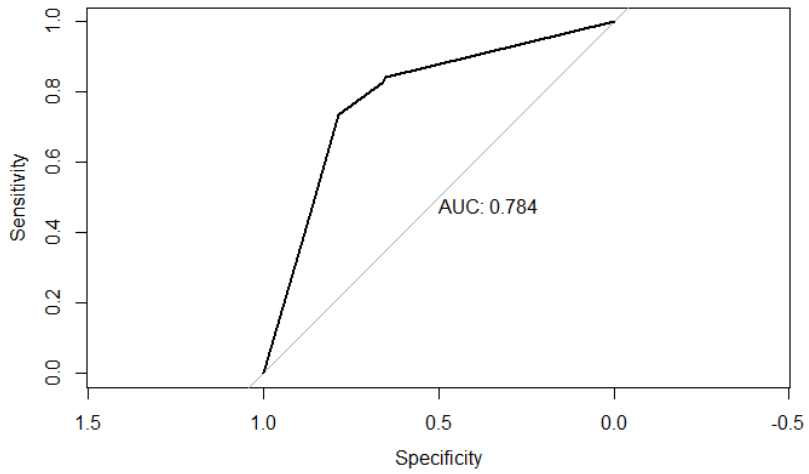
c) **Recall, and**

d) **F1 score for the classification model built in the previous sample.**

Task 3. The ROC curve

To plot the ROC curve, we will need some pre-processing.

```
# Calculating and plotting ROC
prob.stroke = predict(p.tree.prune, newdata = test, type = "prob")[,2]

res.roc <- roc(test$stroke, prob.stroke)
plot.roc(res.roc, print.auc = TRUE)
```



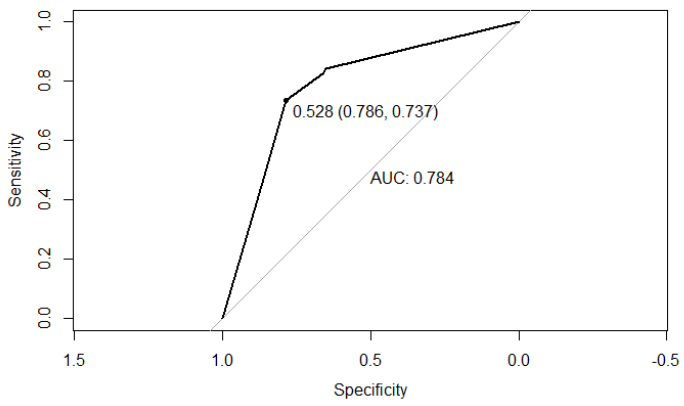
Please note that here we use `type = "prob"` to obtain a matrix of class probabilities, instead of predicted values as we did previously.

We can notice here that ROC is expressed as the ratio between **sensitivity** and **specificity**. The gray diagonal line represents a classifier no better than random chance. A highly performant classifier will have an ROC that rises steeply to the top-left corner, that is it will correctly identify lots of positives without misclassifying lots of negatives as positives.

In our case, the AUC is 0.784, which is much better than random chance, suggesting that we managed to build a very good classifier.

We can also add the best threshold with the highest sum between sensitivity and specificity. Please note that there might be more than one such some in general.

```
# Adding the best threshold value
plot.roc(res.roc, print.auc = TRUE, print.thres = "best")
```



Task 4. Cross validation

As we discussed in Week 6 lecture, ***cross validation*** is a resampling approach that enables us to obtain a generalizable and more honest error rate estimate. There are many ways to run cross validation in R. We will be using a rather straightforward method that utilizes `caret` package to build the same decision tree as we did above.

We will start by loading required packages:

```
# Loading required packages
library(caret)
library(rpart.plot)
```

Next, we will configure `caret` to run 10-fold cross validation. You can explore different configurations for `number` parameter. However, the results (in this case) should be similar for different configurations.

```
# Set up caret to perform 10-fold cross validation
cv.control <- trainControl(method = "cv",
                           number = 10)
```

Finally, we will build a decision tree using those parameters.

```
# Use caret to train the rpart decision tree using 10-fold cross
# validation and use 15 values for tuning the cp parameter for rpart.
# This code returns the best model.
rpart.cv <- train(stroke ~ .,
                  data = train,
                  method = "rpart",
                  trControl = cv.control,
                  tuneLength = 15)
```

What this actually means? Compared to the example above, here we are using 15 values for `cp` parameter tunning (compared to one above), in which case we should obtain the best performing tree for this dataset.

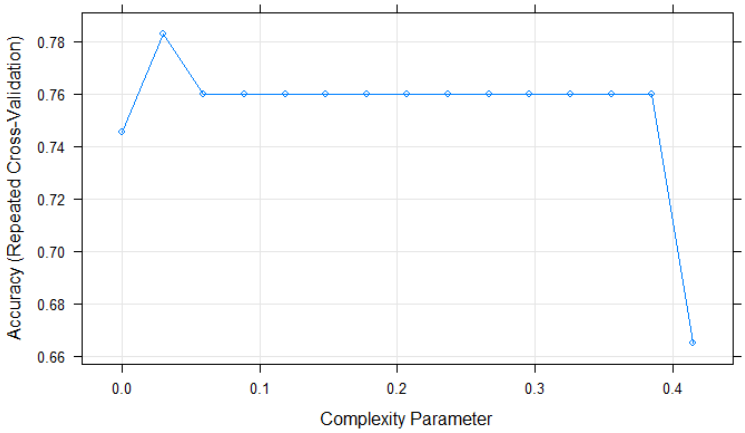
```
rpart.cv
```

```
CART

771 samples
 10 predictor
  2 classes: '0', '1'

No pre-processing
Resampling: Cross-validated (5 fold)
Summary of sample sizes: 617, 617, 617, 617, 616
Resampling results across tuning parameters:

  cp          Accuracy   Kappa
  0.00000000  0.7497109  0.4836127
  0.02961121  0.7638877  0.5211579
  0.05922242  0.7600335  0.5316379
  0.08883363  0.7600335  0.5316379
  0.11844485  0.7600335  0.5316379
  0.14805606  0.7600335  0.5316379
  0.17766727  0.7600335  0.5316379
  0.20727848  0.7600335  0.5316379
  0.23688969  0.7600335  0.5316379
  0.26650090  0.7600335  0.5316379
  0.29611212  0.7600335  0.5316379
  0.32572333  0.7600335  0.5316379
  0.35533454  0.7600335  0.5316379
  0.38494575  0.7600335  0.5316379
  0.41455696  0.6485882  0.1960888

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.02961121.
```

The output shows that the best model was obtained for `cp = 0.02961121`.

We can also plot those values.

```
# Plot model selection
plot(rpart.cv)
```



Compared to the accuracy you calculated in Task 2, does model obtained using cross validation and parameter tunning performs better?

Challenge 1. Let's take another look at Task 2. Particularly, the step that says:

```
# Create training and test sets
```

Please go ahead and use two additional configurations: i) 70/30 training and test split; and ii) 60/40 training and test split.

How this changes the Accuracy, Precision, Recall, and F1 scores that you previously calculated?

Challenge 2. Calculate confidence intervals for accuracy, after doing cross-validation, for:

- 95% confidence, and
- 98% confidence.