Evaluation of classifiers

STAT30270 – Statistical Machine Learning

Contents

1 Cross-validation and classifier selection

In the lectures we worked through assessing the performance of single and multiple classifiers. In particular, we introduced how to split the data into training, validation and test sets in order to select the best classifier among a collection of competitors and evaluate its accuracy at predicting the classification of new data points.

Cross-validation could also be used to select the best classifier. Indeed, k-fold cross-validation can be used to check which model proves better at predicting the validation data points in the dropped folds. In this lab we implement a general k-fold cross-validation procedure that we will use to select the best one among a collection of competing classifiers.

We consider the problem of identifying highly popular subjects in a social network. The dataset `data_influencers.RData` provides information on influential subjects in a social network. Each data point describes two individuals, `A` and `B`. For each individual, numerical features based on Twitter activity (such as volume of interactions, number of followers, retweets, etc.) and on network topology are provided. The variable `choice` represents a human judgement about which one of the two individuals is more influential, either `A` or `B`. The goal is to train a machine learning model which, for a pair of individuals, predicts the human judgement on who is more influential with high accuracy. The main individual of interest is `B`, hence it will be assigned the "positive" class.

```
# load the data
load("data_influencer.RData")
```

We consider 4 competing logistic regression classifiers:

1. A logistic regression using all the features.
2. A logistic regression using only the number of followers of `A` and those of `B`.
3. A logistic regression using only the network-related features.
4. A logistic regression using only the Twitter activity features.

For all models, we consider a threshold of 0.5 to determine the final estimated classification, hence applying the MAP rule (note that this may not be an optimal choice).

First, we set aside a portion of the data (20%) that will be used to test the final selected model. This will "simulate" the real life scenario where we would need to label new/future observations after we selected our best model using the available labeled data. This portion of the data will not be used anywhere in the training and selection of the model, it will be used only at the end for testing.

The remaining portion of the data is used as the base data for training and validation within th k-fold cross-validation procedure.

```
set.seed(202502)   # to reproduce this example

# set aside test data
n <- nrow(data_influencer)
test <- sample(1:n, n*0.2)
```

```
data_test <- data_influencer[test,]

# select data for training and validation
train <- setdiff(1:n, test)
data <- data_influencer[train,]
n_train <- nrow(data)
```

There is a quite relevant amount of variability across the ranges of variation of the features, hence we decide to standardize the input features (to have mean 0 and variance 1). Note that the features are related to different aspects of the social network, some record counts, other are continuous features and are derived quantities, therefore tend to take values in different ranges and on different scales.

The standardization statistics (sample means and standard deviations) estimated on the training + validation data are used to standardize the test data. The reason why we take this approach is because we want to minimize or avoid entirely **test information leakage** or **data leakage**, a situation in which information from outside the training data is employed to train/validate the model. This situation can occur when information extracted/related to the test data "leaks" into the training data, causing an overly optimistic or unfair evaluation of a model. Note that more in general, data leakage occurs because the model benefits from knowing the distribution of the test data, which violates the principle of keeping test data completely unseen.

By not using the whole data `data_influencer` (train + validation + test) to compute the standardization statistics, we make sure that information from the test data is not leaked into the data used for training and validation. In addition, we standardize the test data by using the statistics computed on the training data, as in this way the model does not get indirectly access to information about the test data distribution.

```
# check ranges and standard deviations
ranges <- apply( data_influencer[,-1], 2, range )
matplot(t(ranges), type = "l", col = "black")
#
sds <- apply(data_influencer[,-1], 2, sd)
plot(sort(sds), pch = 19)

# standardize -- see ?scale
x_scale <- scale(data[,-1])
data[,-1] <- x_scale
#
# standardize separately the test data
data_test[,-1] <- scale(data_test[,-1], center = attr(x_scale, "scaled:center"),
                        scale = attr(x_scale, "scaled:scale"))

# NOTE:
colMeans(data[,-1])        # all zero
colMeans(data_test[,-1])   # different from zero!
```

To measure the predictive performance, we use accuracy, sensitivity, and specificity, as we consider equally important to identify correctly when `A` is more influential than `B`, and when `B` is more influential than `A`. The different models are defined in terms of subset of features given in input during the training process.

```
# we use this function to compute metrics
class_metrics <- function(y, yhat) {
  yhat <- factor( yhat, levels = c(0, 1) )  # ensure 2 classes
  tab <- table(y, yhat)
  acc <- sum(diag(tab))/sum(tab)
  sens <- tab[2,2]/sum(tab[2,])
  spec <- tab[1,1]/sum(tab[1,])
  out <- c(acc = acc, sens = sens, spec = spec)
  return(out)
}
```

```r
# used to define the four different models
set2 <- c("choice", "A_follower_count", "B_follower_count")
set3 <- c("choice", "A_network_feature_1", "A_network_feature_2", "A_network_feature_3",
          "B_network_feature_1", "B_network_feature_2", "B_network_feature_3")
set4 <- c(1, 2:9, 13:20)
```

To compare the models, we consider a 5-fold cross-validation. Note that setting $K = 5$ corresponds to a 80%/20% split between training and validation set at each step of the cross-validation procedure. Because at each instance we randomly split the data into the folds, we replicate the cross-validation procedure $R = 50$ times to obtain a better estimate of the predictive performance of the competing models.

```r
K <- 5      # set number of folds
R <- 50     # set number of replicates --- NOTE : could be slow

out <- vector("list", R)    # store accuracy output
# out is a list, each slot of this list will contain an array
# containing the metrics for each classifier in the K folds

for ( r in 1:R ) {

  metrics <- array(NA, c(K, 4, 3))            # accuracy of the classifiers in the K folds
  folds <- rep( 1:K, ceiling(n_train/K) )
  folds <- sample(folds)             # random permute
  folds <- folds[1:n_train]          # ensure we got N_train data points

  for ( k in 1:K ) {
    train_fold <- which(folds != k)
    validation <- setdiff(1:n_train, train_fold)

    # fit classifiers on the training data
    #
    fit_log_1 <- glm(choice ~ ., data = data, family = "binomial", subset = train_fold)
    #
    fit_log_2 <- glm(choice ~ ., data = data[,set2], family = "binomial", subset = train_fold)
    #
    fit_log_3 <- glm(choice ~ ., data = data[,set3], family = "binomial", subset = train_fold)
    #
    fit_log_4 <- glm(choice ~ ., data = data[,set4], family = "binomial", subset = train_fold)

    # predict the classification of the validation data observations in the dropped fold
    #
    pred_log_1 <- predict(fit_log_1, type = "response", newdata = data[validation,])
    pred_log_1 <- ifelse(pred_log_1 > 0.5, 1, 0)
    metrics[k,1,] <- class_metrics(y = data$choice[validation], yhat = pred_log_1)
    #
    pred_log_2 <- predict(fit_log_2, type = "response", newdata = data[validation,])
    pred_log_2 <- ifelse(pred_log_2 > 0.5, 1, 0)
    metrics[k,2,] <- class_metrics(y = data$choice[validation], yhat = pred_log_2)
    #
    pred_log_3 <- predict(fit_log_3, type = "response", newdata = data[validation,])
    pred_log_3 <- ifelse(pred_log_3 > 0.5, 1, 0)
    metrics[k,3,] <- class_metrics(y = data$choice[validation], yhat = pred_log_3)
    #
    pred_log_4 <- predict(fit_log_4, type = "response", newdata = data[validation,])
    pred_log_4 <- ifelse(pred_log_4 > 0.5, 1, 0)
    metrics[k,4,] <- class_metrics(y = data$choice[validation], yhat = pred_log_4)
```

```
  }

  out[[r]] <- metrics
  # print(r)   # print iteration number
}
```

The object `out` is a list where the slots are the replications of the cross-validation procedure. Each slot contains the metrics of the four classifiers across the folds, arranged in an array. For example, `out[[19]]` contains the output of the 19th replication of the procedure, the first dimension denotes the fold (1 to 5), the second the classifier (there are 4 in total, so 4 columns), the third the metric (`1` accuracy, `2` sensitivity, `3` specificity).

```
out[[19]]
```

We can calculate the average fold metrics for each one of the considered models in all replications. The resulting array `avg` will be an array of R matrices, where each matrix contains the average metric across the 5 folds for each model. The rows of each matrix denote the classifier, the columns the metric.

```
# because we want to average over the folds (which is dimension 1 of a slot of out)
# we "apply" the mean keeping dimensions 2 and 3 fixed
avg <- sapply( out, function(x) apply(x, c(2,3), mean) )
avg <- array(avg, c(4, 3, R))

# for example, iteration 7
avg[,,7]
```

We can calculate the mean metrics on all the replications and produce a boxplot to visually compare the estimated accuracy of the two classifiers.

```
mean_acc <- rowMeans(avg[,1,])       # estimated mean accuracy
mean_sens <- rowMeans(avg[,2,])      # estimated mean sensitivity
mean_spec <- rowMeans(avg[,3,])      # estimated mean specificity

# just to identify the classifiers
classifiers <- c("log_reg_1", "log_reg_2", "log_reg_3", "log_reg_4")

# average predictive performance
res <- data.frame(acc = mean_acc, sens = mean_sens, spec = mean_spec,
                  row.names = classifiers)
res

# plot
par(mfrow = c(1, 3))
#
# accuracy
mat_acc <- data.frame( avg = c(t(avg[,1,])), classifiers = rep(classifiers, each = R) )
boxplot(avg ~ classifiers, mat_acc, main = "accuracy")
#
# sensitivity
mat_sens <- data.frame( avg = c(t(avg[,2,])), classifiers = rep(classifiers, each = R) )
boxplot(avg ~ classifiers, mat_sens, main = "sensitivity")
#
# specificity
mat_spec <- data.frame( avg = c(t(avg[,3,])), classifiers = rep(classifiers, each = R) )
boxplot(avg ~ classifiers, mat_spec, main = "specificity")
```

1.1 Task

- Which is the best model according to accuracy? Which one is the best according to sensitivity + specificity?
- Comment on how well the different models perform at predicting when the influential subject is `A`. Do you see significant differences?
- Use the selected model to predict the classification of the observations in the test data. In doing so, re-train the selected model on all the data employed for training and validation.
- Investigate the predictive performance of the selected model, and comment on its performance also supplementing the metrics considered with other metrics.
- Re-implement the cross-validation procedure considering different classification thresholds. Investigate how the threshold impacts the selection of the best model (if there are any changes).