Lab Week 6

STAT5003

Dr. Justin Wishart

Semester 2, 2022

Contents

Preparation and assumed knowledge

- Cross validation content in Module 6.
- Listen to the Week 6 lecture pre-recording.
- Required `R` packages
  - `mlbench` for the `Sonar` dataset.
  - `caret` package to simulate the data indices for each fold

Aims

- Construct your own repeated and/or nested cross validation that builds a model that uses k-NN

This week, we will be exploring cross validation and its effect on the estimated error.

We will be using the same data set as lecture example - the `Sonar` data set and it is included as part of the `mlbench` package. To load the data, do the following:

```
data(Sonar, package = "mlbench")
```

1 Construct a 10-fold repeated cross validation

Design a 10-fold cross-validation procedure to evaluate a kNN classification model (with $k = 5$) accuracy. Use `class::knn()` and NOT the `caret` package (but you can use the caret package to create validation cross fold partitions.)

1.1 Create a 10 fold split

Create a 10-fold split and partition up the data into the appropriate training, test splits to be used in the `knn` call

```
library(caret)
library(tidyverse)
library(MASS)
library(class)
library(mlbench)
data(Sonar, package = "mlbench")
set.seed(5003)
fold <- createFolds(Sonar$Class, k = 10)
test.fold.sizes <- vapply(fold, length, numeric(1))
# For convenience, put the 10 fold CV sets into a list
trainingTestSplits <- function(x, dat) {
    ind <- rep("Training", nrow(dat))
    ind[x] <- "Test"
    which.group <- factor(ind)
    split(dat, which.group)
}
training.test.splits <- lapply(fold, trainingTestSplits, dat = Sonar)
```

1.2 Compute the predicted values on each test fold

Using your training test split over the 10 folds. Fit the `knn` models and extract the observed and predicted class labels for each of the 10 folds. (*Hint*: Write a function that does this prediction step for later use, ideally it should have arguments for the training and test data and perhaps the number of)

```
predictKnn <- function(x, k = 5, label.name = "Class") {
    training <- x[["Training"]]
    test <- x[["Test"]]
    feature.vars <- !names(training) %in% label.name
    predicted <- knn(train = training[feature.vars],
                     test = test[feature.vars],
                     cl = training[[label.name]], k = k)
    observed <- test[[label.name]]
    list(predicted = predicted, observed = observed)
}
knn.output <- lapply(training.test.splits, predictKnn)
```

1.3 Compute the performance metrics

Using the predicted and observed values in each case, calculate the sensitivity, specificity, accuracy and F_1 score for the kNN classifier. You may use the `caret::confusionMatrix` or an equivalent helper function from another package if you wish. Look at the performance and identify if it has good/bad performance in the metrics.

```
# Computing the confusion matrices and metrics
confusion.matrix <- lapply(knn.output, function(x) confusionMatrix(x$predicted, x$observed,
                                                                   positive = "M"))
accuracy.estimated <- vapply(confusion.matrix, function(c) c[["overall"]][["Accuracy"]], numeric(1))
metrics.by.class <- vapply(confusion.matrix, function(c) c[["byClass"]], numeric(11))
all.metrics <- rbind(Accuracy = accuracy.estimated, metrics.by.class)
relevant.metrics <- all.metrics[row.names(all.metrics) %in% c("Accuracy", "Sensitivity", "Specificity",
average.metrics <- apply(relevant.metrics, 1, mean)
average.metrics
```

```
##    Accuracy Sensitivity Specificity          F1
##   0.8371429   0.8916667   0.7766667   0.8536988
```

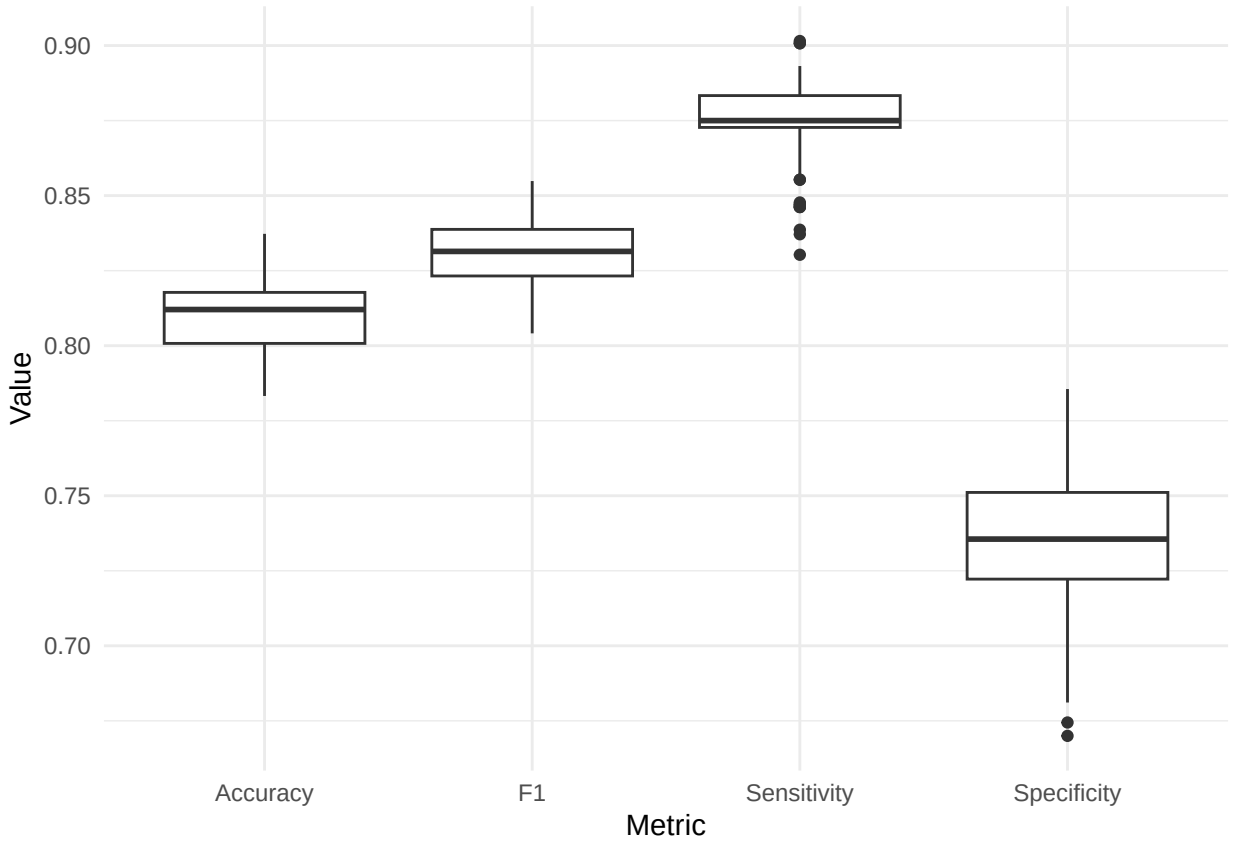
Inspecting the output metrics, the kNN model with $k = 5$ in this case (when `seed = 5003`) had the fairly high sensitivity of 0.8916667 with a lower Specificity and Accuracy.

2 Extend to write your own repeated cross validation

Use your code above to conduct a repeated cross validation of say `m = 100` runs (or however many you wish given the speed of your code or computation power of your hardware) to construct a sample of CV performance estimates for Accuracy, Sensitivity, Specificity and the F1 scores. Visualize your estimates. You may use the `caret::createMultiFolds` or create your own if you wish. The `createMultiFolds` function returns the data in a slightly different format (see the documentation at `? createFolds`)

Solution

```
m <- 100
multi.folds <- lapply(seq(m), function(i) createFolds(Sonar[["Class"]], k = 10))
computeCV <- function(fold, k = 5, dat = Sonar)
{
    training.test <- lapply(fold, trainingTestSplits, dat = dat)
    knn.output <- lapply(training.test, predictKnn)
    confusion.matrix <- lapply(knn.output, function(x) confusionMatrix(x$predicted,
                                                                       x$observed))
    accuracy.estimated <- vapply(confusion.matrix, function(c) c[["overall"]][["Accuracy"]], numeric(1))
    metrics.by.class <- vapply(confusion.matrix, function(c) c[["byClass"]], numeric(11))
    all.metrics <- rbind(Accuracy = accuracy.estimated, metrics.by.class)
    relevant.metrics <- all.metrics[row.names(all.metrics) %in% c("Accuracy", "Sensitivity", "Specificit
    average.metrics <- apply(relevant.metrics, 1, mean)
    average.metrics
}
repeated.cv.metrics <- vapply(multi.folds, computeCV, numeric(4))
dat <- as.data.frame(t(repeated.cv.metrics))
plot.dat <- dat %>% pivot_longer(everything(), names_to = "Metric", values_to = "Value")
ggplot(plot.dat) + geom_boxplot(aes(x = Metric, y = Value)) + theme_minimal()
```



3 Extend to write your own nested cross validation

Consider that in reality the `knn` model will be tuned before being used for classification. Instead of picking a single choice of `k`. Conduct a nested 10 fold cross validation which does a grid search over `k = 2:10` and selects the `k` that produces the best F_1 scores. Say $K = 10$ outer folds and $K_{inner} = 5$ folds within each outer training fold. (*Hint* : use `caret::F_meas` to compute the F_1 score).

Recall, the nested Cross validation will require the following to occur

1. Within each training set in the outer 10-fold split. the training data is split again into another inner training and test set (here the test set is usually called the validation set)
2. For each hyperparameter setting (`k = 2:10`) fit the kNN model on the inner training data and assess on the validation data.
3. Average the performance metric across all inner validation folds for each hyperparameter choice (should get a measure of each value of `k = 2:10` here)
4. Choose the value of `k` that has the best performing metric value.
5. Use that value of `k` as the in the hyper-parameter in outer training, test split.
6. Compute the performance metrics across all test folds using the tuned value of k each time.

Solution

```
set.seed(5003)
n.sonar <- nrow(Sonar)
sonar.seq <- seq(n.sonar)
#
folds <- createFolds(Sonar[["Class"]], k = 10, returnTrain = FALSE)
# Returns the value of k that maximizes the F1 score over k.seq
nestedTuning <- function(test.fold, k.seq, valid.fold.size = 10, dat = Sonar, label.name = "Class")
```

```
{
    training.dat <- dat[-test.fold, ]
    train.valid.folds <- createFolds(training.dat[[label.name]], k = valid.fold.size)
    training.validation <- lapply(train.valid.folds, trainingTestSplits, dat = training.dat)
    metric <- vapply(k.seq, function(k) {
        # Compute the metrics for each choice of k
        knn.output <- lapply(training.validation, predictKnn, label.name = label.name, k = k)
        vapply(knn.output,
               function(x) F_meas(data = x$predicted,
                                  reference = x$observed,
                                  relevant = "M"),
               numeric(1))

    }, numeric(length(training.validation)))
    k.seq[which.max(apply(metric, 2, mean))]
}
tuned.ks <- lapply(folds, nestedTuning, k.seq = 2:10, valid.fold.size = 10)
training.test <- lapply(folds, trainingTestSplits, dat = Sonar)
tuned.output <- mapply(predictKnn, training.test, k = tuned.ks, SIMPLIFY = FALSE)
confusion.matrix <- lapply(tuned.output, function(x) confusionMatrix(x$predicted, x$observed))
accuracy.estimated <- vapply(confusion.matrix, function(cM) cM[["overall"]][["Accuracy"]], numeric(1))
metrics.by.class <- vapply(confusion.matrix, function(cM) cM[["byClass"]], numeric(11))
all.metrics <- rbind(Accuracy = accuracy.estimated, metrics.by.class)
relevant.metrics <- all.metrics[row.names(all.metrics) %in% c("Accuracy", "Sensitivity", "Specificity",
average.metrics <- apply(relevant.metrics, 1, mean)
average.metrics
```

```
##    Accuracy Sensitivity Specificity          F1
##   0.8135714   0.8840909   0.7355556   0.8331466
```