SQM: An R Package for Signature Quality Metrics The Accuracy Component

John A. Ramey

April 30, 2012

1 Introduction

The SQM package contains the R code for the Signature Quality Metrics project under the Signature Discovery Initiative (SDI) at the Pacific Northwest National Laboratory (PNNL). The purpose of the SQM project is to provide subject-matter experts (SMEs) with a set of tools to assess the quality of a set of signatures in terms of accuracy, cost, and utility. For each of these components, we have provided easy-to-use, accessible functions to measure the quality of inputted signatures. In this document, we discuss each of the SQM components and demonstrate their usage with realistic examples[1].

The Accuracy component of SQM enables an SME to explore the efficacy of a set of candidate signatures to determine which of these signatures are nearly optimal as defined by a variety of available accuracy measures. Paired with other SQM components, such as Cost, the Accuracy component equips an SME to identify cost-effective signatures that attain excellent accuracy.

To motivate the Accuracy component, we use a widely-studied, public domain data set with six signatures that we construct below. For demonstration purposes, we use the UCI Machine Learning Repository's Satellite data set[2], which is available in the mlbench R package.

1.1 Signature Construction

The SQM package requires that candidate signatures are previously constructed. To demonstrate an approach this process, we first construct six candidate signatures from the Satellite data set using three machine learning algorithms, known as *classifiers*. The three classifiers are applied to the Satellite data set to obtain the first three signatures. The remaining signatures are constructed by preprocessing the Satellite data set and then applying the same three classifiers.

We begin the signature construction by loading the necessary R packages and the Satellite data set.

[1] Familarity with R is recommended to fully comprehend our provided examples.

[2] For more details about the Satellite data set, see http://archive.ics.uci.edu/ml/datasets/Statlog+(Landsat+Satellite).

```
library("caret")
library("SQM")
library("mlbench")
library("reshape2")
data("Satellite")
```

The Satellite data is provided in a `data.frame`. For clarity, we explicitly assign the variables (i.e covariates/feature vectors) into the usual `X` matrix. Also, we note that the true classes of each observation are given, and we assign these to the usual `Y` vector.

```
satelliteX <- subset(Satellite, select = -classes)
satelliteY <- Satellite$classes
```

As provided, the class labels in the Satellite data set have white space, which can lead to annoying warnings. To stifle these warnings, we replace the whitespace in the class labels with periods.

```
levels(satelliteY) <- gsub(" ", ".", levels(satelliteY))
```

Next, we randomly partition the Satellite data set into a training and a test data set, where the training data set has 1/4 of the original observations and the test data set contains the remaining observations. To partition the data set, we use the `createDataPartition` function from the `caret` package after setting a fixed seed for the random number generator to achieve reproducible results.

```
set.seed(42)
inTrain <- createDataPartition(satelliteY, p = 1/4, list = FALSE,
    times = 1)
trainX <- satelliteX[inTrain, ]
testX <- satelliteX[-inTrain, ]
trainY <- satelliteY[inTrain]
testY <- satelliteY[-inTrain]
```

The class labels for the observations in the data set are considered unknown from the perspective of the signatures. Thus, the class labels in `testY` are considered as the unknown, ground truth. Below, we train three classifiers on the training data set and classify the test observations in hopes that we correctly identify their true classes, which are given in testY. For clarity, we store the test classes as `truthClass`. Each classifier should be viewed as a different signature, and their test classifications/predictions are the `predictedClass` in the `SQM` package.

```
truthClass <- testY
```

We train each of the given classifiers with two training data sets. The first data set is the unaltered training data set from above. To construct the second

data set, we keep only the features (variables) with a pairwise correlation above 0.95 with another feature. The choice of 0.95 is arbitrary. This feature selection approach is useful for demonstration but should be viewed as naive in practice.

```
keptFeatures <- findCorrelation(cor(trainX), cutoff = 0.95)
trainX2 <- trainX[, keptFeatures]
testX2 <- testX[, keptFeatures]
```

To construct each signature, we apply 10-fold cross-validation with the help of the `trainControl` function given by the `caret` package.

```
fitControl <- trainControl(method = "cv", number = 10)
```

As we have discussed above, we utilize three classifiers in order to construct the six signatures. These classifiers that we use are:

1. Linear Discriminant Analysis
2. k-Nearest Neighbors
3. Support Vector Machine with Radial Basis Functions

We train each classifier twice: first on the unaltered training data set and then on the preprocessed data set. We use the `train` function from the `caret` package to train the classifiers.

```
ldaFit1 <- train(trainX, trainY, method = "lda", trControl =
fitControl)
knnFit1 <- train(trainX, trainY, method = "knn", trControl =
fitControl)
svmFit1 <- train(trainX, trainY, method = "svmRadial", trControl
= fitControl)

ldaFit2 <- train(trainX2, trainY, method = "lda", trControl =
fitControl)
knnFit2 <- train(trainX2, trainY, method = "knn", trControl =
fitControl)
svmFit2 <- train(trainX2, trainY, method = "svmRadial", trControl
= fitControl)
```

Next, we formulate the fitted models as lists to easily classify the test data.

```
fittedModels1 <- list(ldaFit1 = ldaFit1, knnFit1 = knnFit1,
svmFit1 = svmFit1)
fittedModels2 <- list(ldaFit2 = ldaFit2, knnFit2 = knnFit2,
svmFit2 = svmFit2)
```

Now, we classify the test observations with each of the six trained models.

```
modelPredictions1 <- lapply(predict(fittedModels1, testX),
as.character)
modelPredictions2 <- lapply(predict(fittedModels2, testX2),
as.character)
modelPredictions <- cbind.data.frame(do.call(cbind,
modelPredictions1),
    do.call(cbind, modelPredictions2))
```

At this point, we have each signatures's classifications for the test observations. We combine the signatures into a `data.frame` to easily export for usage with the `SQM` package. Each row of `signatures` corresponds to a single observation in the test data set. Furthermore, notice that each row of `signatures` has a `signatureID`, `truthClass`, and `predictedClass`: the `signatureID` identifies the corresponding signature, the `truthClass` is the true value of the test observation, and the `predictedClass` is the predicted classification for this test observation by the current signature specified in `signatureID`. We display the first few signatures.

```
signatures <- cbind(obsNum = seq_along(testY), truthClass =
testY,
    modelPredictions)
signatures <- melt(signatures, id = c("obsNum", "truthClass"),
measure = c("ldaFit1",
    "knnFit1", "svmFit1", "ldaFit2", "knnFit2", "svmFit2"))
signatures <- mutate(signatures, obsNum = NULL)
signatures <- with(signatures, cbind.data.frame(signatureID =
variable,
    truthClass, predictedClass = value))
head(signatures)

##   signatureID truthClass predictedClass
## 1     ldaFit1  grey.soil      grey.soil
## 2     ldaFit1  grey.soil      grey.soil
## 3     ldaFit1  grey.soil      grey.soil
## 4     ldaFit1  grey.soil      grey.soil
## 5     ldaFit1  grey.soil      grey.soil
## 6     ldaFit1  grey.soil damp.grey.soil
```

At this point, we have the signatures as they would be provided to the `SQM` package. To continue with our example, we write the signatures to a CSV file. This CSV file would need to be provided in an actual application of the `SQM` package.

```
inputFile <- "demo_signatures.csv"
write.table(signatures, file = inputFile, sep = ",", row.names =
FALSE,
    col.names = FALSE, quote = FALSE)
```

2 Signature Accuracy

The CSV file `demo_signatures.csv` contains the classifications for each signature. The Accuracy component of the `SQM` package can be employed with relative ease after the signatures have been constructed. In fact, with our CSV file in hand, we are able to compute the accuracy measures in one line of R code. We do that here.

```
outAfAccuracy <- afAccuracy(inputFile = inputFile,
outputFilePrefix = "demo")
```

Notice the `outputFilePrefix` argument having the value of `demo`. Two output files are created that contain the accuracy estimates for the signatures. The first file generated is `demo_aggregate.csv`, which is a CSV file that contains the aggregate accuracy estimates for each signature. The second file generated is `demo_byClass.csv`, which is a CSV file that contains the accuracy measures for each class within each signature. The `aggregate` measure is useful for quick comparison among signatures, while the class-by-class breakdown in `byClass` is useful to measure the ability of a signature to correctly classify an observation for a given class.

The object `outAfAccuracy` contains these same accuracy results in a named list with two elements: `aggregate` and `byClass`.

First, let's see the first few `aggregate` results:

```
##   signatureID accuracy precision recall sensitivity specificity    TPR
## 1     knnFit1   0.9605    0.8814 0.8814      0.8814      0.9763 0.8814
## 2     knnFit2   0.9549    0.8648 0.8648      0.8648      0.9730 0.8648
## 3     ldaFit1   0.9433    0.8298 0.8298      0.8298      0.9660 0.8298
## 4     ldaFit2   0.9381    0.8143 0.8143      0.8143      0.9629 0.8143
## 5     svmFit1   0.9599    0.8798 0.8798      0.8798      0.9760 0.8798
## 6     svmFit2   0.9563    0.8688 0.8688      0.8688      0.9738 0.8688
##      TNR     FPR    FNR Fscore
## 1 0.9763 0.02371 0.1186 0.8814
## 2 0.9730 0.02703 0.1352 0.8648
## 3 0.9660 0.03404 0.1702 0.8298
## 4 0.9629 0.03715 0.1857 0.8143
## 5 0.9760 0.02405 0.1202 0.8798
## 6 0.9738 0.02624 0.1312 0.8688
```

And now, the first few `byClass` results:

```
##   signatureID              class accuracy precision recall sensitivity
## 1     knnFit1         cotton.crop   0.9925    0.9589 0.9734      0.9734
## 2     knnFit1      damp.grey.soil   0.9279    0.6290 0.6290      0.6290
## 3     knnFit1           grey.soil   0.9583    0.8828 0.9253      0.9253
## 4     knnFit1            red.soil   0.9863    0.9609 0.9826      0.9826
## 5     knnFit1  vegetation.stubble   0.9666    0.8951 0.7887      0.7887
## 6     knnFit1 very.damp.grey.soil   0.9312    0.8596 0.8444      0.8444
##   specificity    TPR    TNR     FPR     FNR Fscore
## 1      0.9949 0.9734 0.9949 0.00512 0.02657 0.9661
## 2      0.9600 0.6290 0.9600 0.03995 0.37100 0.6290
## 3      0.9672 0.9253 0.9672 0.03284 0.07466 0.9036
## 4      0.9875 0.9826 0.9875 0.01252 0.01741 0.9716
## 5      0.9886 0.7887 0.9886 0.01141 0.21132 0.8385
## 6      0.9578 0.8444 0.9578 0.04224 0.15561 0.8519
```