Estimating Confidences of Individual Regression Predictions with confReg

Sebastian Briesemeister

July 5, 2011

1 Introduction

In contrast to classification approaches, where the uncertainty between classes is an indicator of reliability, most regression methods do not offer an opportunity to estimate the confidence of individual predictions.

In this package, we wrap different methods for confidence estimation for regression prediction. In addition, the package implements a linear regression, a ridge regression, a support vector regression, and a simple feature selection approach. Every confidence estimator measures the confidence in a prediction using a confidence score between zero and one, where a confidence score of zero corresponds to a low-confidence prediction and a score close to one corresponds to a highly confident prediction. In addition, confidence scores are translated into confidence intervals, in which the prediced response lies with a probability of 80%. Given a good confidence estimation, the confidence intervals will be tighter for confident predictions.

To load the package type:

```
> library("confReg")
```

2 An Artificial Dataset

We first create an artificial dataset for testing the described methods of this package. We create a dataset of 200 instances with 10 features and one response. The response is created by applying the Friedman function to the first five features and adding some noise.

```
> set.seed(100)
> dataMatrix <- matrix(runif(10 * 200), 200, 10)
> y <- (10 * sin(pi * dataMatrix[, 1] * dataMatrix[, 2]) + 20 *
+     (dataMatrix[, 3] - 0.5)^2 + 10 * dataMatrix[, 4] + 5 * dataMatrix[,
+     5]) + rnorm(200, 0, 0.1)
> dataMatrix <- cbind(dataMatrix, y)
```

After creating the data matrix, we save the data in a Dataset object of the package.

```
> data <- Dataset(dataMatrix)
```

Since we have to test our methods, we split the dataset in a training and test dataset. We calculate the fold assignment for every instance assuming a split in five parts and assign set five as our test set.

```
> folds <- calculateFolds(data, 5, random = TRUE)
> datasets <- splitByFolds(data, folds, 5)
```

Finally, we normalize the training dataset to zero mean and a standard deviation of one and apply the same normalization to the test dataset.

```
> trainingData <- normalize(datasets[[1]])
> testData <- normalizeBy(datasets[[2]], trainingData)
```

3 Model Creation

Lets create a model on our training data. For this, we will first select features since some features in the dataset are non-predictive. We first create a regression object and a model evaluation object, which is needed for evaluating the worth of a feature set in the feature selection. The regression object can be created with any regression method that uses a formula as input for training, e.g. "lm" and has a corresponding prediction function, e.g. "predict".

```
> me <- ModelEvaluation()
> rModel <- Regression(lm, predict)
```

Note, that we could also use the provided FastRegression object, which is a fast implementation of a ridge regression.

```
> rModel <- FastRegression()
```

We then create a feature selection object, namely a TwinScan object, which aims to minimizes the average error of predictions made with a feature set.

```
> fs <- TwinScan(averageError, min)
```

Now, we select a feature set from the training data using the above objects.

```
> featureList <- selectFeatures(fs, trainingData, me, randomize = TRUE,
+     rModel)
> featureList
```

```
[1] 1 2 3 4 5
```

Lets create a model using the selected feature set.

```
> featureSet(trainingData) <- featureList
> rModel <- learnFromDataset(rModel, trainingData)
```

4 Model Testing

Before we come to the actual confidence estimation, lets test our model on the test data. We assign the feature set of the training data to the test data and predict the responses of the test data.

```
> featureSet(testData) <- featureList
> predictions <- predictDataset(rModel, testData)
> predictions
```

```
             [,1]
 [1,] -0.12633007
 [2,]  0.05029106
 [3,]  0.96222107
 [4,]  0.88265502
 [5,] -0.01338626
 [6,] -0.16452070
 [7,] -0.31711196
 [8,]  0.64604206
 [9,]  1.30421336
[10,] -0.41184359
[11,]  1.74850039
[12,] -0.06082061
[13,] -1.51800586
[14,]  0.14934645
[15,] -0.08763298
[16,] -0.94604161
[17,] -0.04629694
[18,] -0.99593038
[19,]  1.65450202
[20,]  1.28570963
[21,]  0.97761312
[22,] -1.16113762
[23,] -0.75595752
[24,] -0.88318289
[25,] -1.97354000
[26,]  0.48561524
[27,]  0.57489831
[28,] -0.34784174
[29,]  0.16369287
[30,]  1.71971332
[31,]  0.09097077
[32,] -0.60592559
[33,]  0.07625206
[34,] -0.31804901
[35,]  0.48852976
[36,]  0.33610229
```

```
[37,]  1.96057032
[38,]  0.61331678
[39,] -0.83325330
[40,]  0.93586990
```

In contrast, the real response values are:

```
> getResponses(testData)

 [1] -0.095274343 -0.348808298  1.422912533  1.337572529  0.498226183
 [6]  0.802668849 -0.245927507  0.729828014  1.082307465  0.188889423
[11]  1.613101889  0.297076505 -1.451797119 -0.386881926 -0.033540275
[16] -1.146083146  0.160198548 -1.396756382  2.423148935  1.477409733
[21]  1.260573023 -0.661840301 -0.556130160 -0.934713379 -2.174491607
[26]  0.693936671  0.415853753  0.275802280 -0.091780822  2.343349362
[31] -0.301392953 -1.833809161  0.009856587 -0.619404877  0.163229116
[36]  0.521050458  2.704518761  0.623759866 -0.403960291  1.129643713
```

To evaluate the model, lets calculate the average squared error of the predicted responses.

```
> averageError(getResponses(testData), predictions)

[1] 0.1876864
```

Since we normalized the responses the average squared error, the real average squared error is calculated as follows.

```
> realResponses <- unscaleVector(testData, getResponses(testData))
> realPredictions <- unscaleVector(testData, predictions)
> averageError(realResponses, realPredictions)

[1] 4.168221
```

To get a more precise view on the performance of the model, we can also start a five-fold nested cross-validation.

```
> evaluateInCV(me, data, rModel, 5, random = FALSE, fs)

[1] 5.415507
```

The returned value is the average result of the quality function, assigned at the moment of creating the ModelEvaluation object, the average squared error in our case.

5 Confidence Estimation

To predict confidence scores and confidence intervals in addition to our plain predicted responses, we first create ConfidenceEstimator object. In our case, we use ConfidenceEstimatorNNErrors, which estimates the confidence of each prediction by calculating the average sqaured error the the m nearest neighbors.

```
> ce <- ConfidenceEstimatorNNErrors(rModel, trainingData)
```

To train our confidence estimator, we have to provide predictions on the training data. In addition, we have to decide whether we want to optimize our estimator on the training dataset or want to use a kernel estimate instead. We choose to optimize the estimator since it often leads to better results

```
> predictionsTrainingData <- predictDataset(rModel, trainingData)
> ce <- create(ce, optimize = TRUE, predictionsTrainingData)
```

Now, the estimator is trained and can be used to predict the confidences of our predictions.

```
> confidences <- estimate(ce, testData, predictions)
> confidences
```

```
         [,1]        [,2]        [,3]
 [1,] 0.92500 -0.47355372  0.14878158
 [2,] 0.27500 -0.67788918  0.73424720
 [3,] 0.98750  0.61499743  1.15931931
 [4,] 0.40000  0.32563771  1.49431389
 [5,] 0.16250 -0.86610124  0.70254853
 [6,] 0.16250 -1.01723569  0.55141409
 [7,] 0.88125 -0.66832359 -0.04020124
 [8,] 0.94375  0.29296567  0.92553921
 [9,] 0.03125  0.36932665  2.09110391
[10,] 0.27500 -1.14002383  0.27211255
[11,] 0.05625  0.83663238  2.53106517
[12,] 0.08125 -1.00721667  0.71190908
[13,] 0.98750 -1.86522950 -1.32090763
[14,] 0.93750 -0.20186517  0.42625718
[15,] 0.71250 -0.47325338  0.32123428
[16,] 0.91875 -1.29127126 -0.67182949
[17,] 0.21875 -0.82846430  0.67445787
[18,] 0.55000 -1.39969845 -0.53717820
[19,] 0.44375  1.19061621  2.21388330
[20,] 0.37500  0.72647815  1.94739927
[21,] 0.33125  0.37817166  1.64786663
[22,] 0.82500 -1.53172946 -0.87905405
[23,] 0.93125 -1.10517515 -0.47994633
```

```
[24,] 0.30000 -1.58518819 -0.20705683
[25,] 0.94375 -2.32661639 -1.69404285
[26,] 0.63750  0.06788423  0.92733649
[27,] 0.05000 -0.34272437  1.35854453
[28,] 0.25000 -1.09565318  0.34198696
[29,] 0.71250 -0.22192754  0.57256013
[30,] 0.05000  0.80209064  2.50335954
[31,] 0.49375 -0.34108898  0.63123631
[32,] 0.08125 -1.55232165  0.16680411
[33,] 0.29375 -0.63229697  0.75433565
[34,] 0.37500 -0.87728049  0.34364063
[35,] 0.30000 -0.21347554  1.16465583
[36,] 0.92500 -0.01112135  0.61121395
[37,] 0.05000  1.04294764  2.74421654
[38,] 0.48125  0.18125703  1.15358232
[39,] 0.98750 -1.18047694 -0.63615506
[40,] 0.75625  0.55932333  1.22446781
```

For each instance in the test set, the estimator returned a score and a lower and an upper bound of an 80% confidence interval.

To analyze the quality of our confidence estimation on the training data, we calculate the correlation of the prediction errors with the width of the corresponding confidence intervals

```
> evaluateConfidenceEstimates(getResponses(testData), predictions,
+     confidences)

[1] 0.4269676
```

Depending on the random artificial dataset, we see different results. On average the correlation is positive. If we would repeat everything with a dataset of say 1000 instances, the correlation would very likely increase since confidence estimation is usally much more stable on large datasets.

An alternative way of looking at the enrichment of predictions with a low error is by calculating the confidence associated prediction improvement. The prediction improvement shows by what percentage the average squared error is reduced if we consider only the 20% predictions with the smalles confidence interval. Therefore, we calculate the average squared error of the top 20% predictions and normalize it by the error on all predictions.

```
> interval_widths <- confidences[, 3] - confidences[, 2]
> top20 <- which(interval_widths <= sort(interval_widths)[length(interval_widths) *
+     0.2])
> top20Error <- averageError(getResponses(testData)[top20], predictions[top20])
> top20Error

[1] 0.08984874
```

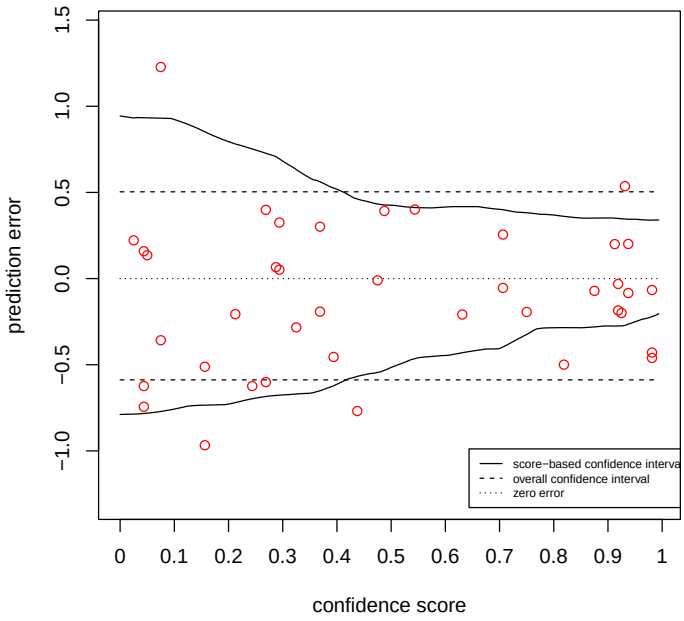

Figure 1: Predicted score-based confidence interval and predicted instances.

```
> overallError <- averageError(getResponses(testData), predictions)
> overallError

[1] 0.1876864

> (1 - top20Error/overallError) * 100

[1] 52.12826
```

Finally, we can evaluate the estimated confidence intervals by plotting the intervals and the predicted instances 1.

```
> plotConfidenceIntervals(ce, getResponses(testData), predictions,
+     confidences[, 1])
```

We expect the confidence interval to be smaller for large confidence scores. We also expect about 80% of the instances within the interval. However, especially for small datasets, often less than the expected number of instances is within the interval borders.