`lrc`: Logistic regression classification (LRC) with an arbitrary loss function

Alex Venzin, Landon Sego

January 13, 2016

1 Introduction

The `lrc` package extends the `glmnet` package by making it possible to train elastic net logistic regression classifiers (LRC's) using a customized, discrete loss function. This allows users to assign unique loss values to false positive and false negative errors. This approach was originally implemented to automate the process of determining the curation quality of mass spectrometry samples. Some of the data documented in [1] will be used here to demonstrate how to train your own classifier. The elastic net parameter estimates are obtained by maximizing a penalized likelihood function. The penalty is essentially a weighted average of the ridge penalty (ℓ_2 norm) and the lasso penalty (ℓ_1 norm) of the regression parameters. This approach balances feature selection and model simplicity.

2 Training

Let's begin by loading the package and the training data:

```
> # Load the package
> library(lrc)
> # Load the VOrbitrap Shewanella QC data
> data(traindata)
> # A view of first five rows and first 12 columns
> traindata[1:5, 1:12]
```

```
      Instrument_Category Instrument Dataset_ID Acq_Time_Start Acq_Length
pt701           VOrbitrap VOrbiETD03     251690     12/31/2011         98
pt702           VOrbitrap VOrbiETD03     251706       1/1/2012         98
pt703           VOrbitrap VOrbiETD03     251887       1/4/2012         98
pt704           VOrbitrap VOrbiETD03     252361      1/10/2012         98
pt705           VOrbitrap VOrbiETD04     255284       2/2/2012         99
                                          Dataset Dataset_Type Curated_Quality
pt701 QC_Shew_11_06_col2A_30Dec11_Cougar_11-10-11      HMS-MSn            good
pt702 QC_Shew_11_06_col2C_30Dec11_Cougar_11-10-11      HMS-MSn            good
pt703  QC_Shew_11_06_Col2B_4Jan12_Cougar_11-10-11      HMS-MSn            good
pt704   QC_Shew_11_06_col1_9Jan12_Cougar_11-10-09      HMS-MSn            good
pt705  QC_Shew_11_06_Col1B_2Feb12_Cougar_11-10-09      HMS-MSn            good
```

```
      XIC_WideFrac XIC_FWHM_Q1 XIC_FWHM_Q2 XIC_FWHM_Q3
pt701     0.297090     19.3820     21.1900     24.3149
pt702     0.305519     19.3785     21.1812     24.3262
pt703     0.327858     19.7357     21.5033     24.5288
pt704     0.305112     20.2610     22.4647     26.2807
pt705     0.314518     23.1260     25.0415     28.3565
```

```
> # Here we select the predictor variables
> predictors <- as.matrix(traindata[,9:96])
```

We fit the LRC model by calling the `LRCglmnet()` function, which requires a binary response variable, coded as a `factor`. The order in which the response variable is coded is important. Specifically, the class we want to predict with the greatest sensitivity should be encoded as the second level. To illustrate how this is done, consider the Shewanella QC data, where the objective is to be sensitive to predicting poor datasets. Hence we code "poor" last, as follows:

```
> response <- factor(traindata$Curated_Quality,
+                    levels = c("good", "poor"),
+                    labels = c("good", "poor"))
> levels(response)
```

```
[1] "good" "poor"
```

Now we must define a discrete loss matrix. For the curation of dataset quality, predicting "good" when the dataset is "poor" is considerably worse (Loss = 5) than predicting "poor" when the dataset is "good" (Loss = 1). Correct predictions receive a penalty of zero loss:

```
> lM <- lossMatrix(c("good","good","poor","poor"),
+                  c("good","poor","good","poor"),
+                  c(     0,     1,     5,     0))
> # Observe the structure of the loss matrix
> lM
```

```
           Predicted.good Predicted.poor
Truth.good              0              1
Truth.poor              5              0
```

To train an elastic net model, the user needs to supply a handful of arguments to the `LRCglmnet` function. The mandatory arguments are the true class labels, `truthLabels` (which, in this case, is, is the `response` object we created above), the matrix of predictor variables, `predictors`, and the loss matrix `lossMat`. Noteworthy additional arguments include `tauVec`, a vector of potential thresholds $\tau \in (0, 1)$ that are used to dichotomize the predicted probabilities from the logistic regression into two class labels; `alphaVec`, a vector of potential values of the elastic net mixing parameter $\alpha \in [0, 1]$; `cvFolds`, the number of cross validation folds; and `masterSeed`, which controls the partitioning the data into the cross validation folds. Keep in mind that α governs the tradeoff between the two regularization penalties. When $\alpha = 0$, the objective is ℓ_2 regularization (ridge regression) and when $\alpha = 1$, the objective is ℓ_1 regularization (lasso regression).

Be advised that heavier sampling of `tauVec` or `alphaVec` (i.e., sequences of greater length) leads to increased computation time, but more of the parameter space will be sampled, potentially leading to a

better classifier. A step by step description of the algorithm that generates the classifier is provided in the appendix. We now train the elastic net logistic regression using the default settings for τ and the number of cross validation folds (which is 5) and restricting $\alpha = (0.5, 1)$:

```
> # Set the number of cores to be one less than the total available
> ncores <- max(1, parallel::detectCores() - 1)
> lrc_fit <- LRCglmnet(response, predictors, lM, nJobs = ncores,
+                      alphaVec = c(1, 0.5), tauVec = c(0.3, 0.5, 0.7),
+                      cvReps = ncores)
>
```

The call to `LRCglmnet` uses cross validation to solve for the optimal parameter settings (α, λ, τ) that minimize the expected loss for the elastic net logistic regression classifier. Printing the resulting object shows the median value for the parameters over the cross validation replicates:

```
> print(lrc_fit)
```

```
The optimal parameter values for the elastic net logistic regression fit:
     Df      %Dev alpha      lambda tau
[1,] 29 0.8174827     1 0.004205331 0.3
```

3 Prediction

Now that the classifier has been properly trained and the optimal parameters have been identified, we are interested in making predictions for new data observations. This requires the elastic net regression model (the output from `LRCglmnet`) and the set of new observations to be predicted, `newdata`. If true labels are available in `newdata`, the column containing these true class labels can be specified via the `truthCol` argument. Additionally, one may wish to carry through a subset of the explanatory variables in `newdata`. These columns are indicated using `keepCols`. True labels are not required to make predictions—but they are required to compute performance metrics (sensitivity, specificity, etc.) for the elastic net logistic regression model. We begin by testing the classifer on the training data:

```
> predictTrain <- predict(lrc_fit, traindata, truthCol = "Curated_Quality", keepCols = 1:2)
> # Look at beginning of the predicted data.  Note the extra columns that were kept.
> head(predictTrain)
```

```
      PredictClass Curated_Quality Instrument_Category Instrument
pt701         good            good           VOrbitrap VOrbiETD03
pt702         good            good           VOrbitrap VOrbiETD03
pt703         good            good           VOrbitrap VOrbiETD03
pt704         good            good           VOrbitrap VOrbiETD03
pt705         <NA>            good           VOrbitrap VOrbiETD04
pt706         poor            poor           VOrbitrap VOrbiETD02
```

```
> # Summarize the peformance of the new classifier in terms of a
> # variety of metrics:
> summary(predictTrain)
```

```
                         poor
sensitivity         0.96363636
specificity         0.88118812
false negative rate 0.03636364
false positive rate 0.11881188
accuracy            0.90314770
```

Note how the sensitivity for detecting poor datsets is considerably better than the specificity. These results reflect a loss function that penalizes false negative errors associated with classifying a "poor" curation quality as "good" more than false positive errors (classifying a "good" curation quality as "poor").
Now let's bring in some new data and examine the performance of the classifier:

```
> # load the data for testing
> data(testdata)
> # Create table observing the true number of good/poor items
> with(testdata, table(Curated_Quality))
```

```
Curated_Quality
good poor
  38   62
```

```
> # Predict new data
> predictTest <- predict(lrc_fit,testdata,
+                        truthCol = "Curated_Quality")
> # Look at the first few rows
> head(predictTest)
```

```
     PredictClass Curated_Quality
931          poor            good
1449         good            good
1467         good            good
1468         good            good
1470         good            good
1501         good            good
```

```
> # Summarize the output of predicting the data we trained on
> summary(predictTest)
```

```
                         poor
sensitivity         0.78688525
specificity         0.94736842
false negative rate 0.21311475
false positive rate 0.05263158
accuracy            0.84848485
```

4 Diagnostics

Finally, we would like to get a sense of the distribution of the parameters that were chosen during the cross validation phase. The `plot` method produces a 3 x 3 scatterplot matrix of the optimal triples (α, λ, τ)

associated with the selected regression model from each cross validation replicate. The univariate distribution of each parameter is plotted on the diagonal of the scatterplot matrix. Ideally, the distributions of the parameters will be tight over the cross validation replicates, indicating that the choice of (α, λ, τ) is stable regardless of the particular random partition used for cross-validation.

```
> plot(lrc_fit)
```

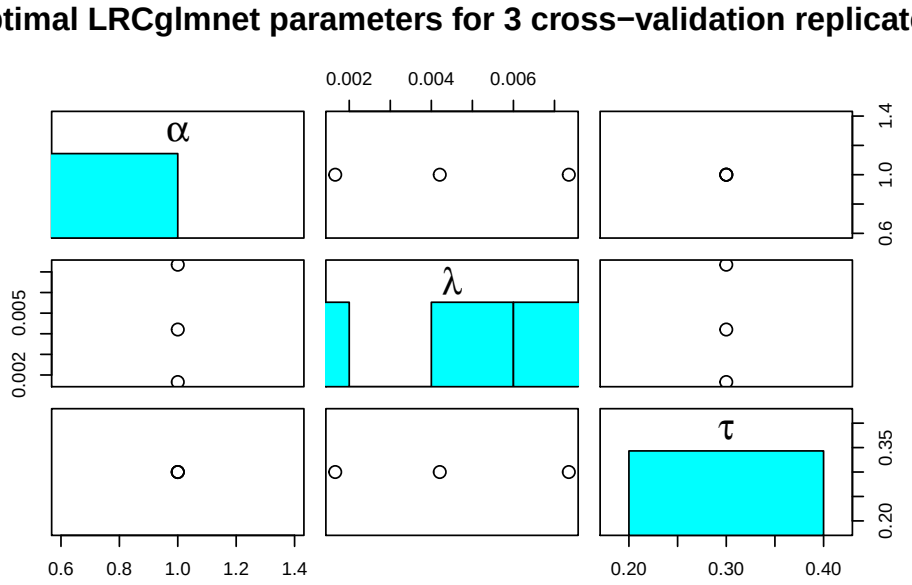

Figure 1: Scatterplot matrix of optimization parameters

References

[1] Brett G. Amidan, Danny J. Orton, Brian L. LarMarche, Matthew E. Monroe, Ronald J. Moore, Alexander M. Venzin, Richard D. Smith, Landon H. Sego, Mark F. Tardiff, and Samuel H. Payne. Signatures for mass spectrometry data quality. *Journal of Proteome Research*, 13(4):2215–2222, 2014.

Appendix

We provide below a high-level explanation of the parameter selection algorithm:

Algorithm 1: LRCGLMNET searches the parameter space of (α, λ, τ) for the combination that minimizes expected loss with respect to the user specified loss matrix.

Input: $truthLabels$ = Labels associated with binary outcome,
$predictors$ = matrix of explanatory variables,
$lossMat$ = user defined discrete loss matrix, $weight$ = vector of weights to fit elastic net model, $alpha$ = vector of elastic net parameter values that balance between lasso and ridge regression, $tauVec$ = vector of probability threshold values, $cvFolds$ = cross-validation folds, $seed$ = random number seed

Output: $parameters = (\alpha, \lambda, \tau)$ triple that minimizes expected loss, $(\tau - 0.5)^2$, and λ for a given cross validation replicate

// For each unique combination of α and `cvFolds`, generate a λ // sequence using `glmnet`, fit an elastic net logisitc regression model, predict // on validation set, and calculate the loss
Loss $\leftarrow array()$;
params $\leftarrow array()$;
for $a \in \alpha$ **do**
    for $v \in cvFolds$ **do**
        $\lambda \leftarrow glmnet(predictors, truthLabels, weight, a)\$\lambda$;
        model $\leftarrow glmnet(predictors, truthLabels, weight, a, \lambda)$;
        LossCalc $\leftarrow predict.loss(model, lossMat, \tau, a, \lambda, predictors, truthLabels, weight)$;
        params $\leftarrow array(a, \lambda, \tau, model, LossCalc)$;
    $Loss[\alpha] \leftarrow params$;
// Calculate expected loss over all training folds ;
out $\leftarrow array()$;
for $v \in cvFolds$ **do**
    $Eloss \leftarrow \frac{params[LossCalc[v]]}{\sum weight[v]}$;
    $out[v] \leftarrow array(Eloss, params[\alpha[v]], params[\lambda[v]], params[\tau[v]], seed)$;
// sort the output based on objective min$Eloss + (\tau - 0.5)^2 - \lambda$
$sorted.out \leftarrow sort(out, by = Eloss + (\tau - 0.5)^2 - \lambda)$;
// optimal parameters will be first row entry
return $parameters = sorted.out[1,]$