# glmnetLRC: Lasso and elastic-net logistic regression classification with an arbitrary loss function

Landon Sego, Alexander Venzin

January 25, 2016

## 1 Introduction

The glmnetLRC package extends the glmnet package by making it possible to train lasso or elastic-net logistic regression classifiers (LRC's) using a customized, discrete loss function to measure the classification error. This allows users to assign unique loss values to false positive and false negative errors. The logistic regression parameter estimates are obtained by maximizing the elastic-net penalized likelihood function that contains several tuning parameters. These tuning parameters are estimated by minimizing the expected loss, which is calculated using cross validation. This approach was originally implemented to automate the process of determining the curation quality of mass spectrometry samples (?, ?). Those same data will be used here to demonstrate how to train your own classifier.

In the sections that follow, we show how to use the glmnetLRC package to train LRC models, create diagnostic plots, extract coefficients, predict the binary class of new observations, and summarize the performance of those predictions. The details of the algorithms used by the package are provided in the Appendix.

## 2 Training

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

```
> # Load the package
> library(glmnetLRC)
> # Load the VOrbitrap Shewanella QC data
> data(traindata)
> # A view of first two rows and first 12 columns
> traindata[1:2, 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
                                        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
      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
```

```
> # Columns 9 to 96 contain various measures of dataset quality that
> # we will use to predict the "Curated_Quality"
> predictors <- as.matrix(traindata[,9:96])
```

We fit the LRC model by calling `glmnetLRC()`, 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:

```
> # Define the loss matrix
> 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 `glmnetLRC()`. 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; `cvReps`, the number of times the cross validation process is repeated with a different random partition of the data, and `masterSeed`, which controls the partitioning of the data into the cross validation folds. Keep in mind that $\alpha$ governs the tradeoff between the two regularization penalties. When $\alpha = 0$, $L_2$ regularization (the ridge penalty) is used, and when $\alpha = 1$, $L_1$ regularization (the lasso penality) is used.

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. We now train the elastic-net logistic regression using restricted values of `tauVec` and `alphaVec` and a small number of cross validation replicates, `cvReps`.

```
> # Set the number of cores to be one less than the total available
> ncores <- max(1, parallel::detectCores() - 1)
```