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

Landon Sego, Alexander Venzin

March 14, 2016

1 Introduction

The glmnetLRC package makes it easy to construct a binary classifier from virtually any number of quantitative predictors that will assign an example, or observation, to one of two classes. It 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 (Amidan et al., 2014). 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 final section of the document.

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 values of the threshold parameter $\tau \in (0, 1)$ 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 α 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 penalty) 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 two less than the number available
> ncores <- max(1, parallel::detectCores() - 2)
> # Fit the LRC model with 2 cross validation replicates per core
> lrc_fit <- glmnetLRC(response, predictors, lM, nJobs = ncores,
+                      cvFolds = 5, cvReps = 2 * ncores,
+                      alphaVec = c(1, 0.5), tauVec = c(0.3, 0.5, 0.7),
+                      estimateLoss = TRUE)
>
```

The call to `glmnetLRC()` 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, as well as the average and standard deviation of the expected loss values calculated for each cross validation replicate.

```
> print(lrc_fit)

The optimal parameter values for the elastic-net logistic regression fit:
     Df      %Dev alpha     lambda tau mean.ExpectedLoss sd.ExpectedLoss
[1,] 11 0.7104052     1 0.02053741 0.3         0.1753846      0.01095085
```

We can also extract the non-zero coefficients of the elastic-net logistic regression model that was created using the optimal values of α and λ (which were shown by the call to the `print()` method above):

```
> coef(lrc_fit)

   (Intercept)   XIC_WideFrac  XIC_Height_Q3     MS1_TIC_Q3     MS1_TIC_Q4
  7.624509e+00  -2.233079e+01   1.572170e+00   6.362096e-03   1.294175e-01
     MS2_Count MS2_Density_Q1           C_4A          IS_1A         MS1_2A
 -4.960917e-05  -1.090757e-03   5.770737e-02   4.458346e-03   2.651479e-04
        MS2_4A           P_2B
 -3.788410e-01  -6.577216e-04
```

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 `glmnetLRC`) and the set of new observations to be predicted, `newdata`. Note that `newdata` must contain all the columns (with equivalent names) that were used to train the LRC. 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 classifier on the original training data:

```
> # Predict 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:  "Instrument_Category" and "Instrument"
> head(predictTrain)
```

```
      PredictClass Curated_Quality Instrument_Category Instrument
pt701         good            good           VOrbitrap VOrbiETD03
pt702         good            good           VOrbitrap VOrbiETD03
pt703         good            good           VOrbitrap VOrbiETD03
pt704         poor            good           VOrbitrap VOrbiETD03
pt706         poor            poor           VOrbitrap VOrbiETD02
pt707         poor            poor           VOrbitrap VOrbiETD02
```

We can summarize the performance of the classifier predictions with a call to the `summary()` method. The performance metrics are oriented in terms of being sensitive to predicting a "poor" dataset. Thus, a false positive is predicting a dataset to be "poor" when it is "good," and a false negative is predicting a dataset to be "good" when it is "poor." This orientation resulted from us setting "poor" as the second level in `response`.

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

```
                         poor
sensitivity         0.92929293
specificity         0.93805310
false negative rate 0.07070707
false positive rate 0.06194690
accuracy            0.93538462
```

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   61
```

```
> # 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 test data
> summary(predictTest)

                          poor
sensitivity         0.88524590
specificity         0.92105263
false negative rate 0.11475410
false positive rate 0.07894737
accuracy            0.89898990
```

If we don't include a truth column in the call to `predict()`, the `summary()` method counts the number of observations classified to each category:

```
> summary(predict(lrc_fit, testdata))

 PredictClass
 good:42
 poor:57
```

4 Diagnostics

Finally, we would like to get a sense of the distribution of the tuning parameters that were chosen during the cross validation phase. The `plot()` method produces a 3×3 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)
```

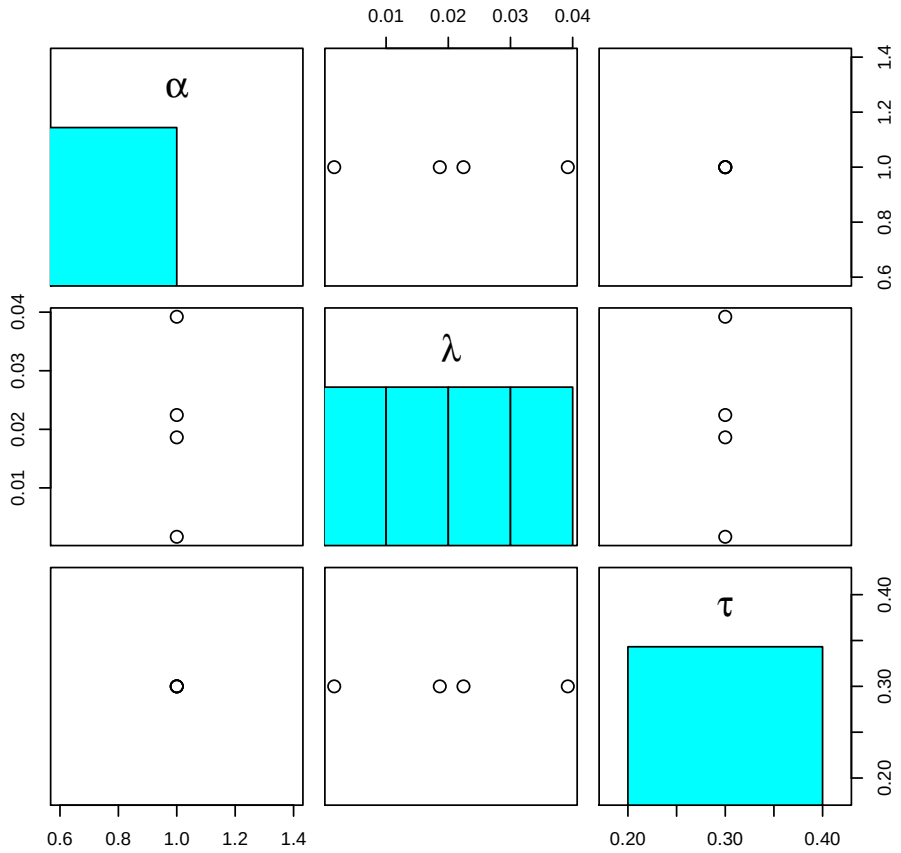

Figure 1: Scatterplot matrix of optimal tuning parameters. Each point represents the optimal estimate of (α, λ, τ) for a given cross validation replicate.

5 Mathematical Details

We present in detail the algorithm used by the `glmnetLRC` package to identify the optimal parameter estimates for a logistic regression classifier (LRC) with variables selection implemented by an elastic-net.

5.1 The model

We begin by defining a number of variables. Let $i = 1, \ldots, N$ index the observations in a training dataset. Let $y_i = 1$ indicate that observation i belongs to category "1" and $y_i = 0$ indicate that it belongs to category "0". Per the logistic regression model, let

$$P(y_i = 1) \equiv \pi_i = \frac{\exp(\beta_0 + \boldsymbol{\beta}_1^T \mathbf{x}_i)}{1 + \exp(\beta_0 + \boldsymbol{\beta}_1^T \mathbf{x}_i)} \tag{1}$$

where $\mathbf{x}_i = (x_1, \ldots x_p)^T$ is a vector of predictors, or covariates, that influence π_i, β_0 is an intercept, $\boldsymbol{\beta}_1 = (\beta_1, \ldots, \beta_p)^T$ is a vector of logistic regression coefficients, and for notational convenience, $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots \beta_p)^T$.

The estimate of the vector of regression parameters, $\boldsymbol{\beta}$, is influenced by two other tuning parameters, α and λ. For this reason, we will often write $\boldsymbol{\beta}$ as $\boldsymbol{\beta}(\alpha, \lambda)$. The value of $\lambda > 0$ controls the weight of the penalty of the log-likelihood function, while α controls the mixture of the ridge and lasso penalties. The relationship between $\boldsymbol{\beta}$, α, and λ will be clarified below. A final tuning parameter, $\tau \in (0, 1)$, provides a threshold for the LRC such that if $\pi_i > \tau$, observation i is predicted to belong to class "1".

5.2 Estimating the regression parameters

When we fit the elastic-net logistic regression model to the data, we obtain the estimator $\hat{\boldsymbol{\beta}}(\alpha, \lambda)$. Therefore, let $\hat{\pi}_i \equiv \pi\big(\mathbf{x}_i, \hat{\boldsymbol{\beta}}(\alpha, \lambda)\big)$ denote the predicted probability that $y_i = 1$. Then, if $\hat{\pi}_i > \tau$, the LRC predicts that $y_i = 1$, otherwise it predicts that $y_i = 0$. It will be useful to represent the predicted class of observation i as $\hat{y}_i \equiv f\big(\mathbf{x}_i, \hat{\boldsymbol{\beta}}(\alpha, \lambda), \tau\big) = I_{(\hat{\pi}_i > \tau)}$, where f can be thought of as the LRC. For the elastic-net, the estimate $\hat{\boldsymbol{\beta}}$ is the $\boldsymbol{\beta}$ that maximizes the penalized, binomial log-likelihood function:

$$\hat{\boldsymbol{\beta}}(\alpha, \lambda) = \underset{\boldsymbol{\beta} \in \mathbb{R}^{p+1}}{\arg\max} \left[\ell(\mathbf{x}_i, \boldsymbol{\beta}) - \lambda \left(\frac{1 - \alpha}{2} \sum_{k=1}^{p} \beta_k^2 + \alpha \sum_{k=1}^{p} |\beta_k| \right) \right] \tag{2}$$

where the unpenalized log-likelihood is given by

$$\ell(\mathbf{x}_i, \boldsymbol{\beta}) = \frac{1}{N} \sum_{i=1}^{N} \Big[y_i \big(\beta_0 + \boldsymbol{\beta}_1^T \mathbf{x}_i\big) - \log\big(1 + \exp(\beta_0 + \boldsymbol{\beta}_1^T \mathbf{x}_i)\big) \Big] \tag{3}$$

Parenthetically, $\alpha = 1$ is the lasso penalty, $\alpha = 0$ is the ridge regression penalty, and $0 < \alpha < 1$ is a mixture of the two. The penalty in (2) is the one specified in the documentation of the `glmnet` package (Hastie & Qian, 2014).

5.3 Estimating the tuning parameters

The optimal values of the tuning parameters, α, λ, and τ, are obtained by minimizing the risk, or expected loss, of the LRC, where the risk is calculated via cross validation. Calculating risk requires that we define a discrete loss function, $L(y, \hat{y})$ as follows:

	$\hat{y} = 0$	$\hat{y} = 1$
$y = 0$	0	κ_0
$y = 1$	κ_1	0

with $\kappa_0 > 0$ and $\kappa_1 > 0$ chosen to reflect the severity of false-positive and false-negative errors, respectively. Setting $\kappa_0 = \kappa_1 = 1$ results in the commonly used 0-1 loss function. In the `glmnetLRC` package, L is specified via a call to `lossMatrix()`, and the result is passed to the `lossMat` argument of `glmnetLRC()`.

Cross validation is accomplished by randomly partitioning the data into M folds (non-overlapping and exhaustive subsets), where each fold is tested using a model trained on the remaining folds. The value of M is controlled by the `cvFolds` argument in `glmnetLRC()`. Following the presentation of Hastie, et al. (2008), let

$$\delta : \{1, \ldots, N\} \to \{1, \ldots, M\} \tag{4}$$

map each observation in the training data to one of the folds. Let $\hat{\boldsymbol{\beta}}^{-k}(\alpha, \lambda)$ represent the estimate of $\boldsymbol{\beta}$ obtained by fitting the elastic-net logistic regression model to all the training data except the k^{th} fold. The cross validation estimate of the risk is given by

$$R(\alpha, \lambda, \tau) = \sum_{i=1}^{N} w_i L\Big(y_i, f\big(\mathbf{x}_i,\ \hat{\boldsymbol{\beta}}^{-\delta(i)}(\alpha, \lambda),\ \tau\big)\Big) \Big/ \sum_{i=1}^{N} w_i \tag{5}$$

where the weights w_i are specified in the `lossWeight` argument to `glmnetLRC()`. The optimal estimates of the tuning parameters are those that minimize the risk:

$$(\hat{\alpha}, \hat{\lambda}, \hat{\tau}) = \underset{\alpha,\lambda,\tau}{\arg\min}\, R(\alpha, \lambda, \tau) \tag{6}$$

In practice, we calculate $(\hat{\alpha},\ \hat{\lambda},\ \hat{\tau})$ by computing (5) over an irregular cube of discrete parameter values, defined by the combination of three vectors: $\boldsymbol{\alpha} \times \boldsymbol{\lambda} \times \boldsymbol{\tau}$. The point in the cube that minimizes the risk becomes the estimate for (α, λ, τ). In the event there are ties for the lowest risk for two or more points in the cube, points with τ nearer to 0.5 are preferred, and if that still doesn't break the tie, points with larger values of λ are preferred because they result in a more parsimonious model with fewer predictors. The values of $\boldsymbol{\alpha}$ and $\boldsymbol{\tau}$ are specified by the `alphaVec` and `tauVec` arguments of `glmnetLRC()`, respectively. The values of $\boldsymbol{\lambda}$ depend on each α and are chosen algorithmically by `glmnet()`, using the default values for the relevant arguments in `glmnet()` and using the entire training dataset.

So far in this discussion, we have made reference to a single random partition of the data into M folds. Naturally, the estimates of the tuning parameters depend on the partition. A different partition will yield different estimates of the tuning parameters. To ensure the final LRC is robust to the random partitioning process, we repeat the training process for multiple partitions, or cross validation replicates, indexed by $j = 1, \ldots, J$. The value of J is controlled by the `cvReps` argument in `glmnetLRC()`.

Calling the `plot()` method on the object returned by `glmnetLRC()` shows a pairs plot and univariate histogram of the various $(\hat{\alpha}_j, \hat{\lambda}_j, \hat{\tau}_j)$. This plot illustrates the consistency (or lack thereof) of the tuning parameter estimates across cross validation replicates.

5.4 The final LRC model

Once $\hat{\alpha}_j$, $\hat{\lambda}_j$, and $\hat{\tau}_j$ are identified for all the cross validation replicates, the final estimate of the tuning parameters is obtained by calculating the median of each one separately:

$$(\hat{\alpha}^\star, \hat{\lambda}^\star, \hat{\tau}^\star) = \big(\underset{j}{\text{median}}(\hat{\alpha}_j),\ \underset{j}{\text{median}}(\hat{\lambda}_j),\ \underset{j}{\text{median}}(\hat{\tau}_j)\big) \tag{7}$$

The final estimate of $\boldsymbol{\beta}$ is obtained by fitting all the training data (via (2)) using the final estimates of the tuning parameters (7), which gives rise to the final LRC:

$$f^\star \equiv f\big(\mathbf{x},\ \hat{\boldsymbol{\beta}}(\hat{\alpha}^\star, \hat{\lambda}^\star),\ \hat{\tau}^\star\big) \tag{8}$$

Calling the `predict()` method on the object returned by `glmnetLRC()` uses $f^\star$ to classify new observations. Likewise, calling the `coef()` method on the object returned by `glmnetLRC()` returns $\hat{\boldsymbol{\beta}}(\hat{\alpha}^\star, \hat{\lambda}^\star)$.

5.5 The cross validation estimate of the risk

A measure of overall performance for the LRC is provided by the cross validation estimate of the risk. Extending (4), let $\delta_1, \ldots, \delta_J$ represent the same partition mappings of the J cross validation replicates

that were used to train the LRC. Using the final tuning parameter estimates $(\hat{\alpha}^\star, \hat{\lambda}^\star)$ defined by (7), a corresponding set of regression parameter estimates, $\hat{\boldsymbol{\beta}}_j^{-k}(\hat{\alpha}^\star, \hat{\lambda}^\star)$, are obtained using (2) for each of the M folds in replicate j. The estimate of the risk for replicate j is given by applying (5) as follows:

$$R_j = \sum_{i=1}^{N} w_i L\Big(y_i,\ f\big(\mathbf{x}_i,\ \hat{\boldsymbol{\beta}}_j^{-\delta_j(i)}(\hat{\alpha}^\star, \hat{\lambda}^\star),\ \hat{\tau}^\star\big)\Big) \Big/ \sum_{i=1}^{N} w_i \tag{9}$$

The value of R_j is calculated for $j = 1, \ldots, J$ and summarized using the mean and standard deviation in the usual way:

$$\bar{R} = \frac{1}{J}\sum_{j=1}^{J} R_j, \qquad \sigma_R = \sqrt{\frac{\sum_{j=1}^{J}(R_j - \bar{R})^2}{J-1}} \tag{10}$$

The values of $\bar{R}$ and σ_R are obtained by calling `glmnetLRC()` with the argument `estimateLoss = TRUE` and then printing the resulting object.

References

Amidan, B. G., Orton, D. J., LarMarche, B. L., Monroe, M. E., Moore, R. J., Venzin, A. M., ... Payne, S. H. (2014). Signatures for mass spectrometry data quality. *Journal of Proteome Research*, *13*(4), 2215-2222.

Hastie, T., & Qian, J. (2014). *Glmnet vignette.* Retrieved 20 January 2016, from `http://cran.fhcrc.org/web/packages/glmnet/vignettes/glmnet_beta.html`

Hastie, T., Tibshirani, R., & Friedman, J. H. (2008). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). Springer-Verlag.