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

Landon Sego, Alexander Venzin

January 15, 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.

In the sections that follow, we show how to use the `lrc` 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:
We fit the LRC model by calling `LRCglmnet()`, 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: 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:

To train an elastic net model, the user needs to supply a handful of arguments to `LRCglmnet()`. 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 $\alpha = (0.5, 1)$:
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, as well as the average and standard deviation of the expected loss that is calculated for each cross validation replicate.
We can also extract the coefficients of the logistic regression model that was created using the optimal values of α and λ (which were shown by the call to the
tt print method above):

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: 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:

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.

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.

[2] Trevor Hastie, Robert Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer-Verlag, New York, 2 edition, 2008.

Appendix

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

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

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

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 $\boldsymbol{\beta}$ is influenced by two other parameters, α and λ. The value of $\lambda > 0$ controls the weight of the penalty of the log-likelihoood function, while α controls the mixture of the ridge and lasso penalties. The relationship between $\boldsymbol{\beta}$, α, and λ will be clarified below.

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 $\hat{\pi}_i > \tau$, where $0 < \tau < 1$, the LRC predicts that $y_i = 1$. To streamline the notation, let the predicted class of observation i be written as $\hat{y}_i \equiv f\big(\mathbf{x}_i, \hat{\boldsymbol{\beta}}(\alpha, \lambda), \tau\big) = I_{(\hat{\pi}_i > \tau)}$, and f can be though of as the LRC.

For the elastic net, the estimate $\hat{\boldsymbol{\beta}}$ is the $\boldsymbol{\beta}$ that maximizes the penalized log likelihood function:

$$\hat{\boldsymbol{\beta}}(\alpha, \lambda) = \arg\max_{\boldsymbol{\beta}} \left[\ell(\mathbf{x}_i, \boldsymbol{\beta}) - \lambda \left(\frac{1 - \alpha}{2} \sum_{i=1}^{p} \beta_i^2 + \alpha \sum_{i=1}^{p} |\beta_i| \right) \right] \tag{2}$$

where the unpenalized likelihood is given by

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

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 `glmnet()` in the `glmnet` package.

The optimal values of 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.

Cross-validation is accomplished by randomly partitioning the training data into m folds (non-overlapping and exhaustive subsets), of the training data. The value of m is controlled by the `cvFolds` argument in `glmnet()`. Following the presentation of Hastie et al. [2], let

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

map each observation in the training data to one of the folds. Let $f^{-k}\big(\mathbf{x}, \hat{\boldsymbol{\beta}}(\alpha, \lambda), \tau\big)$ represent the LRC obtained by estimating $\boldsymbol{\beta}$ using all the training data except the k^{th} fold. The cross-validation estimate of

the risk is given by

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

Note that (5) is defined such that each observation in the training data receives equal weight. The optimal estimates of α, λ, and τ 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). The values of $\boldsymbol{\alpha}$ and $\boldsymbol{\tau}$ are specified by the `alphaVec` and `tauVec` arguments of `LRCglmnet()`, 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()`.

Once $\hat{\alpha}_j$, $\hat{\lambda}_j$, and $\hat{\tau}_j$ are identified for a given partition j of the training data, an overall estimate of $\boldsymbol{\beta}_j$ is obtained by solving (2) for the entire training data set using $\alpha = \hat{\alpha}_j$ and $\lambda = \hat{\lambda}_j$ to obtain $\hat{\boldsymbol{\beta}}(\hat{\lambda}_j, \hat{\alpha}_j)$. Thus, the LRC obtained using partition j of the training data is given by $f\big(\mathbf{x}, \hat{\boldsymbol{\beta}}(\hat{\alpha}_j, \hat{\lambda}_j), \hat{\tau}_j\big)$. We can also estimate the risk using the overall parameter estimates by inserting $\hat{\alpha}_j$, $\hat{\lambda}_j$, and $\hat{\tau}_j$ into (5) as follows:

$$R_j = \frac{1}{N} \sum_{i=1}^{N} L\Big(y_i, f\big(\mathbf{x}_i, \hat{\boldsymbol{\beta}}_j(\hat{\alpha}_j, \hat{\lambda}_j), \hat{\tau}_j\big)\Big) \tag{7}$$

If we repeat the entire parameter estimation process for a different random partition, j', of the training data, we obtain obtain slightly different results. 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 `LRCglmnet()`. For each replication, we obtain $(\hat{\alpha}_j, \hat{\lambda}_j, \hat{\tau}_j)$, which has a corresponding risk, R_j, defined by (7). Calling the `plot()` method on the object returned by `LRCglmnet()` shows a pairs plot and univariate histogram of the various $(\hat{\alpha}_j, \hat{\lambda}_j, \hat{\tau}_j)$. This plot illustrates the consistency of the tuning parameter estimates across cross validation replicates.

The final estimate of the tuning parameters is obtained by calculating the median of each one separately:

$$(\hat{\alpha}^\star, \hat{\lambda}^\star, \hat{\tau}^\star) = \big(\text{median}_j(\hat{\alpha}_j), \text{median}_j(\hat{\lambda}_j), \text{median}_j(\hat{\tau}_j)\big) \tag{8}$$

and then creating the final LRC by fitting all the training data (via (2)) with those estimates:

$$f^\star \equiv f\big(\mathbf{x}, \hat{\boldsymbol{\beta}}_j(\hat{\alpha}^\star, \hat{\lambda}^\star), \hat{\tau}^\star\big) \tag{9}$$

Calling the `predict()` method on the object returned by `LRCglmnet()` uses $f^\star$ to the classify new observations.

The overall estimate of the risk (and its standard error) is obtained by calulating the mean and standard deviation of the R_j's in the usual way:

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