Logistic Regression with Penalites

D2K Course Staff

2/28/2019

Introduction

One of the most common classification models for predictive modeling is Logistic Regression. It is a generalized linear model for a binomial y distribution. By calculating the logit of the binomial probability, the logistic regression model gives us a linear decision boundary that we can use for classifying binary data. Note that logistic regression is a probabilistic model that computes the likelihood of a "success" given the data, or $P(y = 1|X = x_i)$.

Logistic Regression

We define the logistic regression model as,

$$P(y_i = 1|X = x_i) = \frac{1}{1 + e^{-(\beta_0 + \beta^T x_i)}}$$

or we can express it in terms of the logit,

$$log(\frac{P(y_i = 1|X = x_i)}{P(y_i = 0|X = x_i)}) = \beta_0 + \beta^T x_i$$

where X is the $n \times p$ data matrix, and β is the $p \times 1$ vector of unknown parameters we learn, and β_0 is the intercept terms. We calculate the unknown parameters by solving the following optimization problem,

$$\min_{\beta_0, \beta} - \Sigma_{i=1}^{N} [y_i(\beta_0 + \beta^T X_i) - log(1 + e^{\beta_0 + \beta^T x_i})]$$

using the gradient descent algorithm.

Just like with linear regression, the quality of a predictive model is defined by its ability to predict on new "unseen" data rather than its ability to fit to the training data. However, we cannot measure our predictive ability using MSE like we did for linear regression since we are working with discrete data. There are many different loss functions that you can use to assess binary classification methods, but the Cross-Entropy loss function is preferred for logistic regression. We show the form of the loss function below,

$$L(y, \hat{y}) = \Sigma_{i=1}^{n} - (y_i log(\hat{y}_i) + (1 - y) log(1 - \hat{y}_i))$$

where $\hat{y}_i$ is the outputted probability from the logistic regression model and y_i is the true value of the binary outcome variable.

We use an example dataset from https://www.kaggle.com/ronitf/heart-disease-uci and split the data into our training/validation, query, and test sets so that we can train, compare and evaluate our final model with unseen data.

```r
# Set seed for reproducibility
set.seed(123)

### Create 60-20-20 split for training/validation, query, test
### sets ###

# Read in data, load as matrix 'x' and vector 'y'
heart_data <- read.csv("data/heart.csv")

# Set mpg as outcome variable
y <- heart_data$target

# Set all other variables as predictors
x <- heart_data %>% select(age, sex, cp, trestbps, chol, fbs,
    restecg, thalach, exang, oldpeak, slope, ca, thal)

# Create training/test index
tr_idx <- sample(1:nrow(x), nrow(x) * 0.6, replace = FALSE)

# Create training/validation set
x_train <- x[tr_idx, ]
y_train <- y[tr_idx]

x_test <- x[-tr_idx, ]
y_test <- y[-tr_idx]

# Create query set from test set
qr_idx <- sample(1:nrow(x_test), nrow(x_test) * 0.5, replace = FALSE)

x_query <- x_test[qr_idx, ]
y_query <- y_test[qr_idx]

x_test <- x_test[-qr_idx, ]
y_test <- y_test[-qr_idx]

# Combine x and y for flexibility later on
train <- data.frame(target = y_train, x_train)
query <- data.frame(target = y_query, x_query)
test <- data.frame(target = y_test, x_test)
```

It is important to not only create a test set, but also a query set since we will be comparing the performance of the logistic regression model to the penalized logistic regression models we will fit later on. Now that we have chunked our data to keep the query and test sets "hidden" for now, we can fit our logistic regression model on the training data before evaluating it on the query set and reporting the misclassification error

```r
# Load library to calculate cross-entropy
library(MLmetrics)

# Fit logistic regression model on training data
log_fit <- glm(as.factor(target) ~ ., data = train, family = binomial(link = "logit"))

# Predict new probabilities using the X values in the query
# data
y_query_log <- predict(log_fit, query, type = "response")
```

```
# Calculate cross-entropy on query set
log_query_loss <- LogLoss(y_query_log, y_query)
log_query_loss
```

```
[1] 0.2965096
```

We can see the cross-entropy of the logistic regression model on the query set above. It is hard to determine how "good" of a model this is without comparing it to other models. Next, we will examine different versions of logistic regression with penalty terms to see if we are able to improve our predictive ability.

Penalized Logistic Regression

Since logistic regression is a generalized linear model, it is based upon a transformation of a **linear** model. This means that the same penalties we applied to the linear regression model in the **penalized_linear_regression.Rmd** file can be applied to logistic regression as well. We simply add the L1 and L2 norms with the mixing coefficient α. We solve the same optimization problem to formulate our penalized logistic regression model as the normal logistic regression model, however, we add the penalty terms to the end.

$$\min_{\beta_0,\beta} - \Sigma_{i=1}^{N}[y_i(\beta_0 + \beta^T X_i) - log(1 + e^{\beta_0+\beta^T x_i})] + \lambda[\alpha||\beta||_1 + (1-\alpha)||\beta||_2]$$

When $\alpha = 1$ we perform lasso logistic regression, when $\alpha = 0$ we perform ridge logistic regression, and when *alpha* is any value between we consider it elastic net logistic regression.

Lasso Logistic Regression

We first assess how lasso logistic regression performs relative to normal logistic regression.

```
# Set seed for reproducibility
set.seed(123)

# Load glmnet package
library(glmnet)

# Set lambda values
lambdas <- 10^seq(-3, 2, by = 0.1)

# Find optimal lambda using built-in cross validation in
# glmnet package
lasso_cv <- cv.glmnet(as.matrix(x_train), y_train, type.measure = "deviance",
    alpha = 1, lambda = lambdas, family = "binomial")

# Predict on query set
y_query_lasso <- predict(lasso_cv, newx = as.matrix(x_query),
    s = "lambda.min", type = "response")

# Calculate cross-entropy on query set
lasso_query_loss <- LogLoss(y_query_lasso, y_query)
lasso_query_loss
```

```
[1] 0.3223555
```

We can see that at the optimal λ, lasso logistic regression performs better than logistic regression without penalization. This due to the reduction in significance of variables that we determine are not predictive during the cross-validation process.

Ridge Logistic Regression

We next assess how ridge logistic regression performs relative to normal logistic regression and the lasso.

```
# Set seed for reproducibility
set.seed(123)

# Find optimal lambda using built-in cross validation in
# glmnet package
ridge_cv <- cv.glmnet(as.matrix(x_train), y_train, type.measure = "deviance",
    alpha = 0, lambda = lambdas, family = "binomial")

# Predict on query set
y_query_ridge <- predict(ridge_cv, newx = as.matrix(x_query),
    s = "lambda.min", type = "response")

# Calculate cross-entropy on query set
ridge_query_loss <- LogLoss(y_query_ridge, y_query)
ridge_query_loss
```

```
[1] 0.3219586
```

We can see that at the optimal λ, ridge logistic regression performs better than logistic regression without penalization, but still not as well as the lasso.

Elastic Net

Now that we have attempted both the Lasso and Ridge regression, we try a combination of the two and set $\alpha = 0.5$.

```
# Set seed for reproducibility
set.seed(123)

# Find optimal lambda using built-in cross validation in
# glmnet package
elnet_cv <- cv.glmnet(as.matrix(x_train), y_train, type.measure = "deviance",
    alpha = 0.5, lambda = lambdas, family = "binomial")

# Predict on query set
y_query_elnet <- predict(elnet_cv, newx = as.matrix(x_query),
    s = "lambda.min", type = "response")

# Calculate cross-entropy on query set
elnet_query_loss <- LogLoss(y_query_elnet, y_query)
elnet_query_loss
```

```
[1] 0.3229246
```

We can see that the elastic net performs better than the ridge and logistic regression without penalization. However, the lasso still performs slightly better in comparison.

It is important to note that α is a hyper-parameter that can and should be tuned to optimize predictive results.

Final Model

After comparing all of our models at their optimal configuration using the cross-validation and a query set, we can now retrain our best model on all of the already seen data (i.e. traing **and** query sets) and report our final accuracy on the test set. Since our best model was the Lasso, we will use that to report our final cross-entropy loss.

```
# Set seed for reproducibility
set.seed(123)

# Combine training and query sets
x_train_query <- data.frame(rbind(x_train, x_query))
y_train_query <- c(y_train, y_query)

# Find optimal lambda on combined sets using built-in cross
# validation in glmnet package
final_lasso_cv <- cv.glmnet(as.matrix(x_train_query), y_train_query,
    type.measure = "deviance", alpha = 1, lambda = lambdas, family = "binomial")

# Predict on test set
y_test_pred <- predict(final_lasso_cv, newx = as.matrix(x_test),
    s = "lambda.min", type = "response")

# Calculate cross-entropy on the test set
test_loss <- LogLoss(y_test_pred, y_test)
test_loss
```

```
[1] 0.3707949
```

We can see that we actually performed better on the final test set than we did on any of the query sets. This indicates that we did a good job avoiding overfitting and that we were able to select a model that performs well on future data. The variation in cross-entropy values between the query and the test set is also likely due to the small sample size of our data. Such differences are way less frequent when working on larger data sets.

References

Hastie, Trevor, et al. The Elements of Statistical Learning Data Mining, Inference, and Prediction. Springer, 2009.

Hastie, Trevor, and Junyang Qian. "Glmnet Vignette." Web.stanford.edu, Stanford University, 26 June 2014, web.stanford.edu/~hastie/glmnet/glmnet_alpha.html.