Logistic Regression

Jim Harner

5/31/2020

6.4 Logistic Regression

In many situations the outcome variable is one of k levels or groups. In this section we explore the case in which $k = 2$.

6.4.1 Basics

Define the possible groups as $G = \{G_1,\, G_2\}$. We define a outcome variable as $Y = 1$ if G_1 occurs and 0 otherwise, i.e., outcomes y_i, $i = 1, 2, \ldots, n$ are binary values of 0 and 1. We are interested in $P(Y = 1 \,|\, \mathbf{x})$ where $\mathbf{x}$ are observed features.

Thus, we want a function that takes the data and transforms it into a single value bounded inside the closed interval [0, 1] . The inverse-logit function is commonly used:

$$P(x) = \text{logit}^{-1}(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x}, \quad -\infty < x < \infty.$$

The range is $[0, 1]$ and thus $P(x)$ can be interpreted as a probability. When x is large, e^{-x} is small, so the denominator is close to 1 and the overall value is close to 1. Similarly when x is small, e^{-x} is large so the denominator is large, which makes the function close to zero.

The logit function takes values in the $[0, 1]$ range and transforms them to the real line.

$$\text{logit}(p) = log(\frac{p}{1-p}) = log(p) - log(1-p).$$

In order to model the data, we look at a slightly more general form of the inverse-logit function:

$$P(Y_i = y_i \,|\, \mathbf{x}_i) = [\text{logit}^{-1}(\beta_0 + \beta^t \mathbf{x}_i)]^{y_i} [1 - \text{logit}^{-1}(\beta_0 + \beta^t \mathbf{x}_i)]^{1-y_i},$$

where y_i is the outcome and $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$ is the vector of p features for observation i.

If $y_i = 1$,

$$P(Y_i = 1 \,|\, \mathbf{x}_i) = \text{logit}^{-1}(\beta_0 + \beta^t \mathbf{x}_i) = \frac{1}{1 + e^{-(\beta_0 + \beta^t \mathbf{x}_i)}}.$$

Similarly, if $y_i = 0$,

$$P(Y_i = 0 \,|\, \mathbf{x}_i) = 1 - \text{logit}^{-1}(\beta_0 + \beta^t \mathbf{x}_i) = \frac{e^{-(\beta_0 + \beta^t \mathbf{x}_i)}}{1 + e^{-(\beta_0 + \beta^t \mathbf{x}_i)}}.$$

The log of the *odds ratio* is:

$$\text{logit}(P(Y_i = 1 \,|\, \mathbf{x}_i)) = log(\frac{P(y_i = 1 \,|\, \mathbf{x}_i)}{1 - P(y_i = 1 \,|\, \mathbf{x}_i)}) = \beta_0 + \beta^t \mathbf{x}_i,$$

the logit of the probability that outcome i is G_1 is being modeled as a linear function of the features. This model is called the *logistic regression model.*

The parameter β_0 is the *base rate*, or the unconditional probability of "1" knowing nothing about the feature vector $\mathbf{x}$.

If you had no information about your specific situation except the base rate, the average prediction would be given by just β_0:

$$P(y_i = 1) = \frac{1}{1 + e^{-\beta_0}}.$$

The parameter β defines the slope of the logit function. Note that in general it's a vector that is as long as the number of features you are using for each data point. The vector β determines the extent to which certain features are markers for increased or decreased likelihood of G_1.

Estimating β_0 and β

We use the training data to estimate β_0 and β. We use maximum likelihood estimation, which requires a convex optimization algorithm. We cannot use derivatives and vector calculus since the problem is not linear.

If $\theta = \{\beta_0, \beta\}$, the *likelihood function* L is:

$$L(\theta \,|\, \mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n) = P(\mathbf{X}_1 = \mathbf{x}_1 \,|\, \theta) \cdots P(\mathbf{X}_n = \mathbf{x}_n \,|\, \theta),$$

where the $\mathbf{X}_i$ are assumed to be independent.

You search for the parameters that maximize the likelihood, given the data:

$$\theta = argmax_\theta \, \Pi_{i=1}^n P(\mathbf{X}_i = \mathbf{x}_i \,|\, \theta).$$

Let $p_i = \frac{1}{1+e^{-(\beta_0 + \beta^t \mathbf{x}_i)}}$, the probability of a single observation. Then

$$P(\mathbf{X}_i = \mathbf{x}_i \,|\, \theta) = p_i^{y_i}(1 - p_i)^{1-y_i}.$$

Thus

$$\theta_{MLE} = argmax_\theta \, \Pi_i^n \, p_i^{y_i}(1 - p_i)^{1-y_i}.$$

How do we maximize the likelihood?

If you take derivatives with respect to β_0 and β and set the results to zero, the result is not in closed form. Maximizing the likelihood is equivalent to maximizing the log likelihood or equivalently minimizing the negative log likelihood.

Which optimization algorithm do we use? Newton's method and stochastic gradient descent both converge to a global maximum if they converge. This will occur if the variables are not linearly dependent, i.e., *Hessian matrix* is positive definite.

Newton's Method

Newton's method follows from calculus, i.e., a function can be approximated by the first few terms of its Taylor series.

Given a step size γ, we must compute the local gradient $\nabla\Theta$, which corresponds to the first derivative, and the Hessian matrix H, which corresponds to the second derivative. Each step of the algorithm looks like:

$$\theta_{m+1} = \theta_m - \gamma H^{-1} \cdot \nabla\theta.$$

Newton's method uses curvature of the log likelihood to choose a step direction. This involves inverting a $(p+1) \times (p+1)$ matrix, which is difficult if there are a lot of features. Actually you can solve a linear system of equation, but this is still difficult for large p.

Stochastic Gradient Descent

The *stochastic gradient* descent approximates a gradient using a single observation at a time. This algorithm updates the current best fit each time it sees a new data point. There's no big matrix inversion, and it works well with both huge data and sparse features.

Stochastic gradient descent is used by Spark for large scale machine learning algorithms.

A/B Testing

A/B testing is a methods of evaluation when we must decide on one of two possible actions. In statistics this is two-sample problem.

Actual vs. Predicted Table	Predicted = F	Predicted = T
Actual = F	TN	FP
Actual = T	FN	TP

Terms:

- Accuracy = $\frac{TP+TN}{TP+TN+FP+FN}$
- Precision = $\frac{TP}{TP+FP}$ (or positive predicted value)
- Recall = $\frac{TP}{TP+FN}$
- Sensitivity = Recall
- Specificity = $\frac{TN}{TN+FP}$

Performance metrics

We use different evaluation metrics for different kinds of models, and in different contexts. For logistic regression we typically:

1. estimate probabilities and then rank-order the items relative to their probabilities in decreasing order of likelihood. If you wanted to know how good your model was at discovering *relative rank*, you'd look to one of:

- **Area under the receiver operating curve (AUC)**: A *receiver operating characteristic curve*, or *ROC* curve, is defined as a plot of the true positive rate against the false positive rate for a binary classification problem as you change a threshold. The area under that curve, referred to as the *AUC*, is a way to measure the success of a classifier or to compare two classifiers.
- **Area under the cumulative lift curve**: The area under the *cumulative lift curve*, which is frequently used in direct marketing and captures how many times it is better to use a model versus not using the model (i.e., just selecting at random).

2. compute the predicted probability for any given unlabeled item to use this for classification. Then to minimize the misclassification rate, if the predicted probability is > 0.5 that the label is 1, you would label the item a 1, and otherwise 0. You have several options for how you'd then evaluate the quality of the model:

- **Lift**: How much more people are responding because of a model.
- **Accuracy**: How often the correct outcome is being predicted.

- **Precision**: This is the (number of true positives)/(number of true positives + number of false positives).

- **Recall**: This is the (number of true positives)/(number of true positives + number of false negatives).

- **F1-score**: This combines precision and recall into a single score. It's the harmonic mean of precision and recall, so:

  $$f1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}.$$

  Generalizations of $f1$ are based on changing weights for precision and recall.

It's hard to compare lift curves, but you can compare AUC (area under the receiver operator curve)—they are "base rate invariant." That is, if you bring the response rate from 1% to 2%, that's 100% lift; but if you bring it from 4% to 7%, that's less lift but more effect. AUC does a better job in such a situation when you want to make comparisons.

6.4.2 Binomial Wine Quality Example

We illustrate logistic regression modeling using the Wine Quality Data Set from the UCI Machine Learning Repository. The two datasets: the red and white variants of the Portuguese "Vinho Verde" wine, but we will restrict analysis to the red wine dataset.

The features (or predictors) are all numeric, whereas the labels (or the predictand) is a score ranging from 0 to 10 representing the wine quality. Technically, this is an ordered categorical model, but we will binarize the outcome variable at a threshold value of 5.0.

We can use `read.csv()` to read the `winequality-red.csv` file into local memory.

```r
wine_red_df <-read_csv2(
  "/home/rstudio/rspark-tutorial/data/wine/winequality-red.csv") %>%
  mutate(quality_bin = ifelse(quality > 5.0, 1, 0)) %>%
  resample_partition(p = c(training = 0.7, test = 0.3))
wine_red_train_df <- wine_red_df$training$data
```

The data is then partitioned into training and test data frames.

In order to determine which variables are good predictors of wine quality, variable selection can be done using the following code (not executed). However, to make comparisons between R and Spark models, we will use the Spark training data in R.

```
wine_red_train_df$quality_bin <- as.factor(wine_red_train_df$quality_bin)
wine_red_logistic_fit_0 <- glm(quality_bin ~ 1, data = wine_red_train_df,
                               family = binomial)
wine_red_logistic_fit_full <- glm(quality_bin ~ fixed.acidity + volatile.acidity + citric.acid + residua
wine_red_logistic_step <-step(wine_red_logistic_fit_full,
                              scope=formula(wine_red_logistic_fit_0),
                              direction="backward", k = 2)
wine_red_logistic_step
```

It would be possible to copy this local training data frame to Spark using `copy_to()`, but instead we read the `winequality-red.csv` file directory into a Spark DataFrame using `spark_read_sdf`.

```r
wine_red_sdf <- spark_read_csv(sc, "wine_red_sdf",
    path = "file:///home/rstudio/rspark-tutorial/data/wine/winequality-red.csv",
    delimiter = ";" )
wine_red_sdf
```

```
## # Source: spark<wine_red_sdf> [?? x 12]
##    fixed_acidity volatile_acidity citric_acid residual_sugar chlorides
##            <dbl>            <dbl>       <dbl>          <dbl>     <dbl>
##  1           7.4            0.7          0               1.9     0.076
##  2           7.8            0.88         0               2.6     0.098
##  3           7.8            0.76         0.04            2.3     0.092
##  4          11.2            0.28         0.56            1.9     0.075
##  5           7.4            0.7          0               1.9     0.076
##  6           7.4            0.66         0               1.8     0.075
##  7           7.9            0.6          0.06            1.6     0.069
##  8           7.3            0.65         0               1.2     0.065
##  9           7.8            0.580        0.02            2       0.073
## 10           7.5            0.5          0.36            6.1     0.071
## # ... with more rows, and 7 more variables: free_sulfur_dioxide <dbl>,
## #   total_sulfur_dioxide <dbl>, density <dbl>, pH <dbl>, sulphates <dbl>,
## #   alcohol <dbl>, quality <int>
```

```
# wine_red_tbl <- sdf_register(wine_red_sdf, name = "wine_red_tbl")
```

We register the Spark DataFrame so that the Scala Spark DataFrame API is used directly rather than the `dplyr` interface. Registering forces the SQL to completion without using `collect`, which is necessary for the pipeline in the next chunk.

We split `wine_red_sdf` into a training and a test Spark DataFrame. First, we need to cast `quality` as numeric in order to binarize it with a threshold.

```
wine_red_partition <- wine_red_sdf %>%
  mutate(quality = as.numeric(quality)) %>%
  ft_binarizer(input_col = "quality", output_col = "quality_bin",
               threshold = 5.0) %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 2)
# Create table references
wine_red_train_tbl <- wine_red_partition$training
wine_red_test_tbl <- wine_red_partition$test
```

Performing variable selection in Spark is difficult for logistic models and thus we will use the R `glm` function.

```
wine_red_train_df <- collect(wine_red_train_tbl)
wine_red_train_df$quality_bin <- as.factor(wine_red_train_df$quality_bin)
wine_red_logistic_fit_0 <- glm(quality_bin ~ 1, data = wine_red_train_df,
                               family = binomial)
wine_red_logistic_fit_full <- glm(quality_bin ~ fixed_acidity + volatile_acidity + citric_acid + residua
wine_red_logistic_step <- step(wine_red_logistic_fit_full,
                               scope=formula(wine_red_logistic_fit_0),
                               direction="backward", k = 2)
```

```
## Start:  AIC=1200.02
## quality_bin ~ fixed_acidity + volatile_acidity + citric_acid +
##     residual_sugar + chlorides + free_sulfur_dioxide + total_sulfur_dioxide +
##     density + pH + sulphates + alcohol
##
##                           Df Deviance    AIC
## - pH                       1   1176.0 1198.0
## - density                  1   1176.8 1198.8
## - residual_sugar           1   1177.2 1199.2
## <none>                         1176.0 1200.0
## - chlorides                1   1178.6 1200.6
## - fixed_acidity         1   1179.5 1201.5
## - citric_acid           1   1181.0 1203.0
## - free_sulfur_dioxide   1   1183.3 1205.3
## - total_sulfur_dioxide  1   1202.9 1224.9
## - sulphates             1   1205.1 1227.1
## - volatile_acidity      1   1212.8 1234.8
## - alcohol               1   1224.9 1246.9
##
## Step:  AIC=1198.02
## quality_bin ~ fixed_acidity + volatile_acidity + citric_acid +
##     residual_sugar + chlorides + free_sulfur_dioxide + total_sulfur_dioxide +
##     density + sulphates + alcohol
##
##                        Df Deviance    AIC
## - density               1   1177.2 1197.2
## - residual_sugar        1   1177.3 1197.3
## <none>                      1176.0 1198.0
## - chlorides             1   1178.8 1198.8
## - citric_acid           1   1181.0 1201.0
## - free_sulfur_dioxide   1   1183.5 1203.5
## - fixed_acidity         1   1183.8 1203.8
## - total_sulfur_dioxide  1   1204.1 1224.1
## - sulphates             1   1206.0 1226.0
## - volatile_acidity      1   1212.8 1232.8
## - alcohol               1   1243.2 1263.2
##
## Step:  AIC=1197.21
## quality_bin ~ fixed_acidity + volatile_acidity + citric_acid +
##     residual_sugar + chlorides + free_sulfur_dioxide + total_sulfur_dioxide +
##     sulphates + alcohol
##
##                        Df Deviance    AIC
## - residual_sugar        1   1177.7 1195.7
## <none>                      1177.2 1197.2
## - chlorides             1   1179.8 1197.8
## - citric_acid           1   1182.4 1200.4
## - fixed_acidity         1   1184.6 1202.6
## - free_sulfur_dioxide   1   1184.8 1202.8
## - total_sulfur_dioxide  1   1205.0 1223.0
## - sulphates             1   1206.0 1224.0
## - volatile_acidity      1   1218.1 1236.1
## - alcohol               1   1301.6 1319.6
##
## Step:  AIC=1195.67
## quality_bin ~ fixed_acidity + volatile_acidity + citric_acid +
##     chlorides + free_sulfur_dioxide + total_sulfur_dioxide +
##     sulphates + alcohol
##
##                        Df Deviance    AIC
## <none>                      1177.7 1195.7
## - chlorides             1   1180.2 1196.2
## - citric_acid           1   1182.8 1198.8
## - free_sulfur_dioxide   1   1185.3 1201.3
## - fixed_acidity         1   1185.4 1201.4
## - total_sulfur_dioxide  1   1205.1 1221.1
## - sulphates             1   1206.1 1222.1
## - volatile_acidity      1   1218.2 1234.2
## - alcohol               1   1306.0 1322.0
```

Using AIC as a criterion, `pH`, `density`, and `residual_sugars` are removed successively with AIC values of 1198.0, 1197.2, 1195.7 respectively. At this point, any attempts to remove further variables increase the AIC.

We now fit a logistic model using the `ml_logistic_regression`.

```
wine_red_logistic_fit <- wine_red_train_tbl %>%
  ml_logistic_regression(quality_bin ~ fixed_acidity + volatile_acidity + citric_acid + residual_sugar
```

Unfortunately, the AIC for `ml_logistic_regression` is not available. However, the AIC is available using the R function `ml_generalized_linear_regression` with the `family = binomial` argument. However, this latter function does not support regularization, whereas `ml_logistic_regression` does. Therefore, a thorough analysis involves running both modeling functions.

```
wine_red_logistic_predict <- wine_red_logistic_fit %>%
  ml_predict(wine_red_partition$training) %>%
  sdf_separate_column("probability", list("P[quality_bin=1]" = 2))
wine_red_logistic_predict
```

```
## # Source: spark<?> [?? x 21]
##    fixed_acidity volatile_acidity citric_acid residual_sugar chlorides
##            <dbl>            <dbl>       <dbl>          <dbl>     <dbl>
##  1           4.6            0.52         0.15           2.1     0.054
##  2           4.7            0.6          0.17           2.3     0.058
##  3           4.9            0.42         0              2.1     0.048
##  4           5              0.38         0.01           1.6     0.048
##  5           5              0.4          0.5            4.3     0.046
##  6           5              0.42         0.24           2       0.06
##  7           5              1.02         0.04           1.4     0.045
##  8           5              1.04         0.24           1.6     0.05
##  9           5.1            0.47         0.02           1.3     0.034
## 10           5.1            0.585        0              1.7     0.044
## # ... with more rows, and 16 more variables: free_sulfur_dioxide <dbl>,
## #   total_sulfur_dioxide <dbl>, density <dbl>, pH <dbl>, sulphates <dbl>,
## #   alcohol <dbl>, quality <dbl>, quality_bin <dbl>, features <list>,
## #   label <dbl>, rawPrediction <list>, probability <list>, prediction <dbl>,
## #   probability_0 <dbl>, probability_1 <dbl>, `P[quality_bin=1]` <dbl>
```

The area under the ROC (AUC) for this model is:

```
ml_binary_classification_evaluator(wine_red_logistic_predict, label_col = "quality_bin",
                                   raw_prediction_col = "rawPrediction",
                                   metric_name ="areaUnderROC")
```

```
## [1] 0.8187073
```

The area under the precision-recall (PR) curve is:

```
ml_binary_classification_evaluator(wine_red_logistic_predict, label_col = "quality_bin",
                                   raw_prediction_col = "rawPrediction",
                                   metric_name ="areaUnderPR")
```

```
## [1] 0.8425625
```

Now let's switch to the generalized linear model for examining feature importance.

```
wine_red_br_full_glmfit <- wine_red_train_tbl %>%
  ml_generalized_linear_regression(quality_bin ~ fixed_acidity +
                                   volatile_acidity + citric_acid +
                                   residual_sugar + chlorides +
                                   free_sulfur_dioxide + total_sulfur_dioxide +
                                   density + pH + sulphates + alcohol,
                                   family = binomial(link = "logit"))
tidy(wine_red_br_full_glmfit)
```

```
## # A tibble: 12 x 5
##    term                 estimate std.error statistic  p.value
##    <chr>                   <dbl>     <dbl>     <dbl>    <dbl>
##  1 (Intercept)           75.8      94.1       0.805   4.21e- 1
##  2 fixed_acidity          0.216     0.117     1.84    6.55e- 2
##  3 volatile_acidity      -3.36      0.584    -5.75    9.02e- 9
##  4 citric_acid           -1.47      0.664    -2.22    2.65e- 2
##  5 residual_sugar         0.0690    0.0643    1.07    2.83e- 1
##  6 chlorides             -2.89      1.81     -1.59    1.11e- 1
##  7 free_sulfur_dioxide    0.0266    0.00992   2.68    7.29e- 3
##  8 total_sulfur_dioxide  -0.0177    0.00354  -4.99    5.95e- 7
##  9 density              -85.4      96.1      -0.889   3.74e- 1
## 10 pH                     0.00392   0.847     0.00463 9.96e- 1
## 11 sulphates              2.65      0.516     5.14    2.76e- 7
## 12 alcohol                0.830     0.123     6.74    1.54e-11
```

To save time let's remove both `pH` and `density` (based on the stepwise results above). Notice that we don't really know which variables to remove without dropping all variables one at a time. Spark takes the approach that you will use regularization for variable selection—not a stepwise approach.

```
wine_red_br_full_glmfit <- wine_red_train_tbl %>%
  ml_generalized_linear_regression(quality_bin ~ fixed_acidity +
                                     volatile_acidity + citric_acid +
                                     residual_sugar + chlorides +
                                     free_sulfur_dioxide + total_sulfur_dioxide +
                                     sulphates + alcohol,
                                   family = binomial(link = "logit"))
tidy(wine_red_br_full_glmfit)
```

```
## # A tibble: 10 x 5
##    term                 estimate std.error statistic       p.value
##    <chr>                   <dbl>     <dbl>     <dbl>         <dbl>
##  1 (Intercept)           -9.40     1.13       -8.31  0
##  2 fixed_acidity          0.161    0.0598      2.69  0.00707
##  3 volatile_acidity      -3.49     0.575      -6.06  0.00000000136
##  4 citric_acid           -1.51     0.664      -2.27  0.0233
##  5 residual_sugar         0.0350   0.0517      0.677 0.499
##  6 chlorides             -2.80     1.75       -1.60  0.110
##  7 free_sulfur_dioxide    0.0269   0.00982     2.74  0.00610
##  8 total_sulfur_dioxide  -0.0176   0.00350    -5.05  0.000000449
##  9 sulphates              2.54     0.497       5.12  0.000000302
## 10 alcohol                0.897    0.0883     10.2   0
```

Next we remove `residual_sugar`.

```
wine_red_br_full_glmfit <- wine_red_train_tbl %>%
  ml_generalized_linear_regression(quality_bin ~ fixed_acidity +
                                   volatile_acidity + citric_acid + chlorides +
                                   free_sulfur_dioxide + total_sulfur_dioxide +
                                   sulphates + alcohol,
                                   family = binomial(link = "logit"))
tidy(wine_red_br_full_glmfit)
```

```
## # A tibble: 9 x 5
##   term                 estimate std.error statistic       p.value
##   <chr>                   <dbl>     <dbl>     <dbl>         <dbl>
## 1 (Intercept)           -9.44     1.13        -8.36 0
## 2 fixed_acidity          0.164    0.0596       2.75 0.00588
## 3 volatile_acidity      -3.46     0.573       -6.04 0.00000000154
## 4 citric_acid           -1.49     0.662       -2.24 0.0249
## 5 chlorides             -2.76     1.75        -1.58 0.115
## 6 free_sulfur_dioxide    0.0271   0.00984      2.76 0.00584
## 7 total_sulfur_dioxide  -0.0172   0.00344     -5.01 0.000000552
## 8 sulphates              2.53     0.496        5.09 0.000000363
## 9 alcohol                0.904    0.0878      10.3  0
```

Next we remove `chlorides`.

```
wine_red_br_full_glmfit <- wine_red_train_tbl %>%
  ml_generalized_linear_regression(quality_bin ~ fixed_acidity +
                                   volatile_acidity + citric_acid +
                                   free_sulfur_dioxide + total_sulfur_dioxide +
                                   sulphates + alcohol,
                                   family = binomial(link = "logit"))
tidy(wine_red_br_full_glmfit)
```

```
## # A tibble: 8 x 5
##   term                 estimate std.error statistic  p.value
##   <chr>                   <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)           -9.82     1.11        -8.87 0.
## 2 fixed_acidity          0.183    0.0586       3.12 1.82e- 3
## 3 volatile_acidity      -3.68     0.561       -6.56 5.25e-11
## 4 citric_acid           -1.79     0.636       -2.81 4.94e- 3
## 5 free_sulfur_dioxide    0.0267   0.00979      2.73 6.35e- 3
## 6 total_sulfur_dioxide  -0.0167   0.00341     -4.90 9.38e- 7
## 7 sulphates              2.19     0.440        4.97 6.61e- 7
## 8 alcohol                0.942    0.0851      11.1  0.
```

At this point all features are important in the sense that the AIC is increased if additional features are removed. Also, note that the residual deviance is not appreciably reduced relative to the null deviance.

```
spark_disconnect(sc)
```