Logistic Regression for Sentiment Classification

Stats 216 - Session 5
02/03/2016

Contents

0 - Data and Features ................................................................. 1
1 - Load the Data ........................................................................... 2
2 - Explore the Data ....................................................................... 2
3 - Compress the Data ................................................................. 3
4 - Split the Data .......................................................................... 3
5 - Logistic Regression ............................................................... 3
6 - Cross-Validation ..................................................................... 4
7 - Regularization ......................................................................... 4
8 - Conclusion ............................................................................... 5

In this session you will apply logistic regression to classify movie reviews into positive/negative. You will also observe the bias-variance tradeoff by implementing your own cross-validation code, as well as by using standard tools.

Note: you should use the code contained in session.sentiment.R and fill in the blanks. Before writing your own solution to each task, you should make sure that you understand the corresponding portions of code from session.sentiment.R.

\[
\Pr(Y = 1 | X = x) = \frac{e^{\beta_0 + \beta^T x}}{1 + e^{\beta_0 + \beta^T x}}, \quad (Y \in \{0, 1\})
\]

\[
\min_{(\beta_0, \beta) \in \mathbb{R}^{p+1}} \left[ \frac{1}{N} \sum_{i=1}^{N} y_i \cdot (\beta_0 + x_i^T \beta) - \log(1 + e^{\beta_0 + x_i^T \beta}) \right]
\]

0 - Data and Features

1000 positive and 1000 negative labeled movie reviews from: http://www.cs.cornell.edu/People/pabo/movie-review-data/ (polarity dataset v2.0).
The reviews were first obtained from the IMDb archive http://reviews.imdb.com/Reviews.
The text in each review was processed by performing a few preliminary steps:

- converting all characters to lower case,
- removing punctuation, special characters, extra spaces and stopwords (e.g. “a”, “are”, “the”),
- stemming (i.e. reducing inflected or derived words to their root).
A compressed dictionary, containing 373 of the most commonly occurring words, was created after all the reviews were pre-processed. The feature vector for each review consists in the number of occurrences of each word from the compressed dictionary.

For instance, the following positive movie review

"jaws" is a rare film that grabs your attention before it shows you a single image on screen. The movie opens with blackness, and only distant, alien-like underwater sounds. Then it comes, the first ominous bars of composer John Williams' now infamous score. Dah-dum. ...

but, as we can see now, Spielberg overcame all the obstacles, and delivered one of the finest primal thrillers ever to come out of Hollywood.

is represented in the feature matrix by a row vector whose 373 elements include:

<table>
<thead>
<tr>
<th>film</th>
<th>act</th>
<th>movi</th>
<th>one</th>
<th>perform</th>
<th>actual</th>
<th>first</th>
<th>right</th>
<th>take</th>
<th>time</th>
<th>come</th>
<th>great</th>
<th>let</th>
<th>like</th>
<th>make</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

The feature matrix was created using the \texttt{tm} text mining package available at: \url{https://cran.r-project.org/web/packages/tm/index.html}.

1 - Load the Data

\begin{verbatim}
setwd '~/Box Sync/statshomework/stats216')
rm(list=ls())
load("reviews.RData")

dim(reviews_df)
\end{verbatim}

\begin{verbatim}
[1] 2000 374
\end{verbatim}

\begin{verbatim}
reviews_df[1:5,1:10]
\end{verbatim}

## Sentiment abl act action actor actual almost along also although
## 1  1 0 2 0 0 1 1 0 0 0
## 2  1 0 0 0 0 0 0 1 0 0
## 3  1 0 1 0 0 1 0 0 0 0
## 4  1 0 6 0 1 4 1 0 0 1
## 5  1 0 0 0 1 0 1 0 1 0

2 - Explore the Data

\textbf{Task 2.1}: Study the distribution of word occurrences.

a. Compute the global (i.e. across all reviews) frequency of each unique word.

b. Draw the histogram of word frequencies.

c. Find the 10 most common words. Do you expect them to be good predictors of the review sentiment?

Show/Hide Solution 2.1
3 - Compress the Data

Task 3.1: Write a function that returns the column indices of Sentiment and the first q most occurring words.
We will later use this function to compress the data (with some loss) into a smaller frame on which we can perform a logistic regression with a smaller number of predictors.

- Function name: compress.row.indices
- Input arguments: data_df, q
- Output: a list consisting of q+1 integer numbers, in the form (1, pred_1, pred_2, ..., pred_q)

Hints 3.1:
- Use the code from Task 2.1.

Show/Hide Solution 3.1

4 - Split the Data

Task 4.1: Randomly partition the data for training and testing.

a. Create a new data frame train_data_df containing 85% of the original samples.

b. Create a new data frame test_data_df containing the remaining samples.

Show/Hide Solution 4.1

5 - Logistic Regression

Task 5.1: Perform a logistic regression of Sentiment on 100 predictors from the training data.

a. Use the list of column indices cols_keep obtained in Task 3.1 to train the model log_model_100 on the compressed dataset.

b. Recall that glm performs a z-test for each logistic regression coefficient and reports the corresponding p-value. Which predictors’ coefficients have the smallest associated p-values? Do they seem reasonable? Comment.

c. Use this model to predict the labels on the test data set. Store the predicted labels in log_pred_100.

d. Compute the classification error rate on the test set.

Hints 5.1: ?glm, ?predict.glm

Show/Hide Solution 5.1
6 - Cross-Validation

Task 6.1: We will now explore the effect of the number of predictors on the error rate, by implementing our own cross-validation code.

a. Create a list of 25 lists of column indices `cols_keep_ls` with total number of predictors ranging from 30 to 373. Use the function from Task 3.1.

b. Implement the function `prediction.error`. This function must withhold the rows of `cvdata` specified in `test_id` as a test set and treat the remaining rows as a training set. Then it must train a logistic classifier, using the predictors indicated in the list `cols_keep`, and return the prediction error on the test subset.

- Function name: `prediction.error`
- Input arguments: `cvdata`, `cols_keep`, `test_id`.
- Output: the prediction error

Once you have completed this task, you can perform 10-fold cross-validation on the number of predictors as follows:

```r
prediction.error = dget("functions/prediction.error.R")

one.fold <- function(folds, cvdata, cols_keep_ls, foldid) {
  print(paste("CV fold:",folds))
  sapply(cols_keep_ls, function(cols_keep) prediction.error(cvdata, cols_keep, foldid==folds) )
}

folds <- 1:10
foldid <- sample(rep(folds, length=length(train_data_df$Sentiment)))
cv.errors <- sapply(folds, one.fold, train_data_df, cols_keep_ls, foldid)
```

Show/Hide Solution 6.1

Task 6.2: Inspect the validation error curve and discuss whether any overfitting occurs.

a. How does the prediction error qualitatively behave as the number of predictors varies? How can you explain this?

b. What happens when the number of predictors is very large?

Show/Hide Solution 6.2

7 - Regularization

We can automatically select the best predictors by minimizing the penalized penalized logistic regression objective function.

$$
\min_{(\beta_0, \beta) \in \mathbb{R}^{p+1}} - \left[ \frac{1}{N} \sum_{i=1}^{N} y_i \cdot (\beta_0 + x_i^T \beta) - \log(1 + e^{(\beta_0 + x_i^T \beta)}) \right] + \lambda \| \beta \|_1.
$$

We will solve this optimization problem and select the best value of $\lambda$ by using the `glmnet` library.
library(glmnet)

Task 7.1: Regularized logistic regression and cross-validation.

a. Figure out how to use cv.glmnet to carry out 10-fold cross-validated logistic regression and select the value of $\lambda$ that produces the smallest CV error. cv.glmnet has an associated plot function to display the resulting CV error curve (see ?plot.cv.glmnet for more details); use this function to display CV error as a function of $\lambda$.

b. What do the error bars represent? Can you guess how the optimal value for $\lambda$ was selected?

c. In the model with lowest CV error, which predictors have non-zero coefficients? Can you compare them to those obtained in Task 6.1?

d. How does the validation error vary as a function of the number of predictors? How does this compare to the result obtained in Task 6.1?

e. Use the model obtained in Task 7.1 with lowest CV error to predict the labels from the test set. What is the prediction error now?

Hints 7.1: ?cv.glmnet, ?predict.cv.glmnet

Show/Hide Solution 7.1

8 - Conclusion

In this session we have discussed a sequence of steps that can turn a bunch of raw text files into an automatic predictive system for movie review sentiments.

a. How would you improve any of the steps above to try and decrease the test error?

b. What would you do if your data consisted instead of reviews with three possible labels (i.e. positive, neutral, negative)?