Large-Scale Linear Regression Sentiment Model

Yan Yang (yxy128@berkeley.edu) &
Siamak Faridani (faridani@gmail.com)

Purpose:
For this project, we construct a regression models for predicting the sentiment of reviews from Amazon.com.

Data Provided:
Dictionary: of all words appearing in the reviews;
Line numbers of the words;
Space numbers of the words.

Designs Employed
1. L1 Error functions;
2. Lasso Regularization;

Steps and procedures:

Step 1 Feature extraction:
1. Read a block of matrix from the database using matfile() function
   - a. Extract a review between <review> </review>
   - b. Extract the rate of a particular review between <rate> </rate>
   - c. Extract the review words between <review_text> </review_text>
2. The stopwords (such as “this”, “as”, “a”) are removed from the reviewed words
3. Count unique reviews by mapping each review to the first 6 words as keys, so that only the unique reviews are used.
4. The feature matrix X is constructed by having the dictionary as the header of X and each row is a particular review. The
entry of each row is 1 if the word from the dictionary appears in the corresponding review, 0 otherwise.

5. Y is the ratings for the reviews. Each row is the unique review.

Step 2 Perform Lasso

1. Before performing lasso, we reduced the dimension space of the extracted feature X so that Lasso can be performed more efficiently:

![Graph showing the Number of Features vs AUC](image)

The number of features is selected by taking the first top features and using a subset of data and calculating the resulting AUC of lasso. The AUC increases continuously. However because of the computational limitations, we cut off when AUC is still increasing.

2. Lasso Tuning:
   To tune lambda in Lasso, we use 10-fold cross-validation and calculate the MSE for each lambda. The lowest MSE and the corresponding lambda is the lambda we use for Lasso.
The final lambda (the green circle) is 71.

Results

The ROC graph is below:
The lift plot is below:

The top relevant words:

<table>
<thead>
<tr>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Excellent</td>
<td>Money</td>
</tr>
<tr>
<td>2007</td>
<td>Bad</td>
</tr>
<tr>
<td>Wonderful</td>
<td>Didn’t</td>
</tr>
<tr>
<td>Highly</td>
<td>Reason</td>
</tr>
<tr>
<td>Great</td>
<td>Reviews</td>
</tr>
</tbody>
</table>

Notice that for positive sentiment, most of the words can correspond very well such as wonderful, excellent. For negative sentiment, one particular word that stands out is “bad”.

The Gflop
Extra Credit:

We also implemented multi-core featurizer which asks an input cell array in which each cell is a review or text and outputs a feature vector with a number of features (nminFeatures is the number of times that a feature should be presented in the corpus to be included in the feature vector.)

The approach is basically a bag-of-word but we also add bigrams to the feature vector too removeStopWords is a flag, if it is true it will remove the stop words.

Inputs:
inputcellarray: a cell array with texts as the content of each cell
nFeatures: the number of features that we like to see in the vector
removeStopWords: if ==1 it will remove all the stop words
doStem: a flag if true porter stemmer will be used
grams: 1 for 1-grams, 2 for bigrams (mixed with one grams)
cores: up to 4 on local machine

Outputs:
featureVector
selectedheaderskeys: text of the features

This can be found at
https://github.com/faridani/MatlabNLP/blob/master/nlp%20lib/funcs/featurize_parallel.m