BigML Assignment 5: Logistic Regression Using Stochastic Gradient Descent

Due Wednesday, March 20, 2013 via Blackboard

Out March 6, 2013

1 Important Note

As usual, you are expected to use Java for this assignment. It could take hours to run your experiments. Start early.

Bin Zhao (binzhao@cs.cmu.edu) is the contact TA for this assignment. Please post clarification questions to the Google Group:
`machine-learning-with-large-datasets-10-605-in-spring-2013`

2 Background: SGD for Logistic Regression

One fairly simple way (and extremely scalable way) to implement logistic regression is stochastic gradient descent.

In the lecture we followed Charles Elkan's notes[1], which are for a binary classification task. We estimate the probability p that an example $\mathbf{x} = \langle x_1, \ldots, x_d \rangle$ is positive in the log-odds form:

$$\log \frac{p}{1-p} = \alpha + \sum_{j=1 \ldots d} \beta_j x_j \tag{1}$$

If we assume there is a "bias feature" x_0 that is true for every example, then you can simplify and drop the α, leaving just the β_j's to estimate. Therefore

$$p = \frac{\exp(\beta^T \mathbf{x})}{1 + \exp(\beta^T \mathbf{x}))}. \tag{2}$$

[1] `http://cseweb.ucsd.edu/~elkan/250B/logreg.pdf`

It's convenient to consider examples of the form $\mathbf{x}, y$ where $y = 0$ or $y = 1$. The log of the conditional likelihood for example is example will be $LCL(\mathbf{x}, y) = \log p$ if $y = 1$ and $LCL(\mathbf{x}, y) = \log(1 - p)$ if $y = 0$, where p is computed as in Eq. 1. With a little calculus you can show that for a positive example,

$$\frac{\partial}{\partial \beta_j} LCL(\mathbf{x}, y) = \frac{1}{p} \frac{\partial}{\partial \beta_j} p$$

and for a negative example,

$$\frac{\partial}{\partial \beta_j} LCL(\mathbf{x}, y) = \frac{1}{1-p}(-\frac{\partial}{\partial \beta_j} p)$$

and that

$$\frac{\partial}{\partial \beta_j} p = p(1-p)x_j$$

and putting this together we get that if $y = 1$

$$\frac{\partial}{\partial \beta_j} LCL(\mathbf{x}, y) = (1-p)x_j$$

and if $y = 0$ then

$$\frac{\partial}{\partial \beta_j} LCL(\mathbf{x}, y) = -px_j$$

so in either case

$$\frac{\partial}{\partial \beta_j} LCL(\mathbf{x}, y) = (y-p)x_j \tag{3}$$

So an update to the β's that would improve most LCL would be along the gradient—i.e., for some small step size λ, let

$$\beta_j = \beta_j + \lambda(y-p)x_i$$

Notice that if $x_i = 0$ then β_j is unchanged.

So this leads to this algorithm, which is very fast (assuming you have enough memory to hash all the parameter values).

1. Initialize a hashtable B
2. For $t = 1, \ldots, T$
    - For each example $\mathbf{x}_i, y_i$:

- For each non-zero feature of $\mathbf{x_i}$ with index j and value x_j:
  - If j is not in B, set $B[j] = 0$.
  - Set $B[j] = B[j] + \lambda(y - p)x_i$

3. Output the parameters $\beta_1, \ldots, \beta_d$.

The time to run this is $O(nT)$, where n is the total number of non-zero features for each example and T is the number of iterations.

3 Efficient regularized SGD

Logistic regression tends to overfit when there are many rare features. One fix is to penalize large values of β, by optimizing, instead of LCL, some function such as $LCL - \mu \sum_{j=1}^{d} \beta_j^2$. Here μ controls how much weight to give to the penalty term. The update for β_j becomes

$$\beta_j = \beta_j + \lambda((y - p)x_i - 2\mu\beta_j)$$

or equivalently

$$\beta_j = \beta_j + \lambda(y - p)x_i - \lambda 2\mu\beta_j$$

Experimentally this greatly improves overfitting - but unfortunately, this makes the computation much more expensive, because now *every* β_j needs to be updated, not only the ones that are non-zero.

The trick to making this efficient is to break the update into two parts. One is the usual update of adding $\lambda(y - p)x_i$. Let's call this the "LCL" part of the update. The second is the "regularization part" of the update, which is to replace β by

$$\beta_j = \beta_j - \lambda 2\mu\beta_j = \beta_j \cdot (1 - 2\lambda\mu)$$

So we could perform our update of β_j as follows:

- Set $\beta_j = \beta_j \cdot (1 - 2\lambda\mu)$
- If $x_j \neq 0$, set $\beta_j = \beta_j + \lambda(y - p)x_i$

Following this up, we note that we can perform m successive "regularization" updates by letting $B_j = B_j \cdot (1 - 2\lambda\mu)^m$. The basic idea of the new algorithm is to not perform regularization updates for zero-valued x_j's, but

instead to simply keep track of how many such updates would need to be performed to update β_j, and perform them only when we would normally perform "LCL" updates (or when we output the parameters at the end of the day).

Here's the final algorithm (for more detail, see "Lazy sparse stochastic gradient descent for regularized multinomial logistic regression", Bob Carpenter[2])

1. Let $k = 0$, and let A and B be empty hashtables. A will record the value of k last time $B[j]$ was updated.
2. For $t = 1, \ldots, T$
   - For each example $\mathbf{x}_i, y_i$:
   - Let $k = k + 1$
     - For each non-zero feature of $\mathbf{x_i}$ with index j and value x_j:
       * If j is not in B, set $B[j] = 0$.
       * If j is not in A, set $A[j] = 0$.
       * Simulate the "regularization" updates that would have been performed for the $k - A[j]$ examples since the last time a non-zero x_j was encountered by setting
       $$B[j] = B[j] \cdot (1 - 2\lambda\mu)^{k - A[j]}$$
       * Set $B[j] = B[j] + \lambda(y - p)x_i$
       * Set $A[j] = k$
3. For each parameter $\beta_1, \ldots, \beta_d$, set
$$B[j] = B[j] \cdot (1 - 2\lambda\mu)^{k - A[j]}$$
4. Output the parameters $\beta_1, \ldots, \beta_d$.

The learning rate λ is often decreased over time. On the t-th sweep through the data, set $\lambda = \frac{\eta}{t^2}$. I used $\eta = 0.5$. (Sometimes λ is also scaled by $1/n_e$, where n_e is the number of examples.) I also used a value of $2\mu = 0.1$.

[2] `http://lingpipe.files.wordpress.com/2008/04/lazysgdregression.pdf`

When running stochastic gradient descent, it is usual to randomize the order of examples, and scale the feature values so that they are comparable (if they are not already binary). However, randomization is not trivial to do for a large dataset. I recommend implementing SGD as a process that streams once through a data stream, with the number of examples n_e being passed in separately as a command-line argument so that the algorithm is aware of what the current value of t is. Then write a separate module that will input a file of examples and then stream the individual examples out in approximately random order. **Hint:** to randomly sort a file in linux you can do

```
cat -n text_file | sort -R | cut -f2-
```

4 Task

For this assignment, we will be classifying Wikipedia articles by the languages in which they are also available. The data appears at `/afs/cs.cmu.edu/project/bigML/dbpedia/`

The data format is the same as for Assignment 1. There is one instance per line. The first token of each line is the (comma separated) list of class names , then a tab, then the data. Please do multi-label learning and evaluation as described for Assignment 1.

To reduce the experimental load we will only use the abstract data sets for this assignment. Again, they are in increasing size so that you can debug your code on smaller data. The files that start with *abstract* include Wikipidia text.

```
abstract.small.test
abstract.small.train
abstract.test
abstract.train
abstract.tiny.test
abstract.tiny.train
```

As in the naive Bayes assignment, we are going to train 14 binary classifiers (one for each class nl,el,ru,sl,pl,ca,fr,tr,hu,de,hr,es,ga,pt).

In contrast to the previous assignments, we are going to evaluate each document as 14 independent binary classification problems. For each test file, report a single accuracy which is the average over all labels and all test samples.

To reduce the experimental load fix the number of training iterations (scans of data sets) to 20 for all data sets. To fit our model to the memory of a desktop we will use the hash trick discussed in class: map every word to a features id in the range $0 - N$, where N is the dictionary size.

Hint: to convert a string to an id between 0 and N you can do something like

```
int id = word.hashCode() % N;
if (id<0) id+= N;
```

5 Deliverables

Submit a compressed archive (zip, tar, etc) of your code. With it, include a makefile so that

- The command *make demo* will build and test a classifier using the training and test files for the "abstract.tiny" dataset.
- The command *make test TESTFILE="test.txt"* builds a classifier using the "abstract.tiny" training file, and then classifies the documents in the file provided (test.txt in this example).

Your classification code should print out the classification results, with format the same as Assignment 1.

In addition to your code and a makefile, please include a pdf document with the output of your "make demo" command, as well as the percent correct on the other five test sets (using classifiers trained with the corresponding train set).

Also respond to the following questions.

1. Show values of overall likelihood function for each iteration when training with the small data set having dictionary size 10000 and $\mu = 0.1$. The objective function is defined as the sum of all 14 classes $\sum_i \sum_c LCL_c(\mathbf{x}^i, y^{c,i})$. Here c is the label id and i is the document id.

Hint: in order to prevent overflow when calculating p as defined in equation (2) you can use a special version of sigmoid function as the following:

```
static double overflow=20;
protected double sigmoid(double score) {
   if (score > overflow) score =overflow;
   else if (score < -overflow) score = -overflow;
   double exp = Math.exp(score);
   return exp / (1 + exp);
}
```

2. Show accuracy curves for the full data set with varying regularization parameter $\mu = 0, 1e-6, 1e-5, 1e-4, 1e-3, 0.01, 0.1, 0.2, 0.3, 0.5, 1$ and fixed dictionary size $1e5$, and discuss.

3. Show accuracy curves for the full data set with varying dictionary sizes $D = 10, 100, 1e3, 1e4, 1e5, 1e6$ and the best μ values you found in the previous step, and discuss.

4. Compare SGD logistic regression with naive Bayes classifier and discuss. (You will need to retest the naive Bayes classifiers on all three datasets with the new evaluation metric in this assignment).

BONUS question: Is there anything you find particular interesting during this study?

6 Marking breakdown

- Code correctness and makefile functionality: [**60 points**].
- Question 1, 2, 3, 4: [**10+10+10+10 points**]
- BONUS question: [**10 points**]