10601 Machine Learning Assignment 3: Logistic Regression

Due: Sept. 25th, 23:59 EST, via AutoLab
Late submission due: Sept. 27th, 23:59 EST with 50% discount of credits

TAs-in-charge: William Wang, Pengtao Xie

Policy on Collaboration among Students

These policies are the same as were used in Dr. Rosenfeld's previous version of 2013. The purpose of student collaboration is to facilitate learning, not to circumvent it. Studying the material in groups is strongly encouraged. It is also allowed to seek help from other students in understanding the material needed to solve a particular homework problem, provided no written notes are shared, or are taken at that time, and provided learning is facilitated, not circumvented. The actual solution must be done by each student alone, and the student should be ready to reproduce their solution upon request. The presence or absence of any form of help or collaboration, whether given or received, must be explicitly stated and disclosed in full by all involved, on the first page of their assignment. Specifically, each assignment solution must start by answering the following questions in the report:

- Did you receive any help whatsoever from anyone in solving this assignment? Yes / No. If you answered 'yes', give full details: _______________ (e.g. "Jane explained to me what is asked in Question 3.4")
- Did you give any help whatsoever to anyone in solving this assignment? Yes / No. If you answered 'yes', give full details: _______________ (e.g. "I pointed Joe to section 2.3 to help him with Question 2".

Collaboration without full disclosure will be handled severely, in compliance with CMU's Policy on Cheating and Plagiarism. As a related point, some of the homework assignments used in this class may have been used in prior versions of this class, or in classes at other institutions. Avoiding the use of heavily tested assignments will detract from the main purpose of these assignments, which is to reinforce the material and stimulate thinking. Because some of these assignments may have been used before, solutions to them may be (or may have been) available online, or from other people. It is explicitly forbidden to use any such sources, or to consult people who have solved these problems before. You must solve the homework assignments completely on your own. I will mostly rely on your wisdom and honor to follow this rule, but if a violation is detected it will be dealt with harshly. Collaboration with other students who are currently taking the class is allowed, but only under the conditions stated below.

1 Logistic Regression with L_2 Regularization

Logistic regression (LR) classifier is one of the most powerful discriminative classifiers. It allows one to arbitrarily incorporate any features into the model without pain. It is also easy to train LR classifiers using standard constrained or unconstrained optimization techniques, and by incorporating the regularization component, LR is robust to noise. With such nice property, LR is widely used in numerous academic and industrial applications, such as sentiment analysis, question answering, natural language understanding, spoken language processing, and click-through rate prediction.

1.1 Parameter Estimation

In the standard logistic regression model, we define an example $(\mathbf{x}, y)$, where $y = 0$ or $y = 1$, and $\mathbf{x}$ is a feature vector. We will let the predicted value of y be

$$P(Y = y|X = \mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{x}\cdot\mathbf{w}}}$$

So then the probability of a single example $(\mathbf{x}, y)$ is

$$P(Y = y|X = \mathbf{x}, \mathbf{w}) = \begin{cases} \frac{1}{1+e^{-\mathbf{x}\cdot\mathbf{w}}} & \text{if } y = 1 \\ 1 - \frac{1}{1+e^{-\mathbf{x}\cdot\mathbf{w}}} & \text{if } y = 0 \end{cases}$$

To motivate this formula, note that this similar in form to the other linear classifiers we've looked at, like Naive Bayes and the perceptron—it's just that instead of using an argmax or a sign function on the inner product $\mathbf{x} \cdot \mathbf{w}$, we're using the easy-to-differentiate logistic function $f(z) = \frac{1}{1+e^{-z}}$, which approximates a step function between 0 and 1.

What you will need to do is look at the gradient of $\log P_{\mathbf{w}}(y|\mathbf{x})$ for a single example $(\mathbf{x}, y)$. The algorithm we will use will be to pick a random example, compute this gradient, and adjust the parameters to take a small step in that direction - so we're using a randomized approximation to the true gradient over all the data. This turns out to be fast and surprisingly effective—and algorithmically a lot like the perceptron method.

As notation, let p be

$$p \equiv \frac{1}{1 + e^{-\mathbf{x}\cdot\mathbf{w}}} = \frac{1}{1 + \exp(-\sum_j x^j w^j)}$$

and consider the log probability (which, since it's monotonic with the probability, will have the same maximal values):

$$\log P(Y = y|X = \mathbf{x}, \mathbf{w}) = \begin{cases} \log p & \text{if } y = 1 \\ \log(1-p) & \text{if } y = 0 \end{cases}$$

Let's take the gradient with respect to some parameter value w^j. You will consider the two cases separately for now: using $(\log f)' = \frac{1}{f} f'$, we have

$$\frac{\partial}{\partial w^j} \log P(Y = y|X = \mathbf{x}, \mathbf{w}) = \begin{cases} \frac{1}{p}\frac{\partial}{\partial w^j} p & \text{if } y = 1 \\ \frac{1}{1-p}(-\frac{\partial}{\partial w^j} p) & \text{if } y = 0 \end{cases}$$

Now you can get the gradient for p, using $(e^f)' = e^f f'$ and the chain rule:

$$\begin{aligned} \frac{\partial}{\partial w^j} p &= \frac{\partial}{\partial w^j}(1 + \exp(-\sum_j x^j w^j))^{-1} \\ &= (-1)(1 + \exp(-\sum_j x^j w^j))^{-2} \frac{\partial}{\partial w^j} \exp(-\sum_j x^j w^j) \\ &= (-1)(1 + \exp(-\sum_j x^j w^j))^{-2} \exp(-\sum_j x^j w^j)(-x^j) \\ &= \frac{1}{1 + \exp(-\sum_j x^j w^j)} \frac{\exp(-\sum_j x^j w^j)}{1 + \exp(-\sum_j x^j w^j)} x^j \end{aligned}$$

Note that

$$1 - p = \frac{1 + \exp(-\sum_j x^j w^j)}{1 + \exp(-\sum_j x^j w^j)} - \frac{1}{1 + \exp(-\sum_j x^j w^j)} = \frac{\exp(-\sum_j x^j w^j)}{1 + \exp(-\sum_j x^j w^j)}$$

so the final line can be rewritten to give

$$\frac{\partial}{\partial w^j} p = p(1-p)x^j$$

which we can plug into the cases above and get

$$\frac{\partial}{\partial w^j} \log P(Y = y|X = \mathbf{x}, \mathbf{w}) = \begin{cases} \frac{1}{p} p(1-p)x^j = (1-p)x^j & \text{if } y = 1 \\ \frac{1}{1-p}(-1)p(1-p)x^j = -px^j & \text{if } y = 0 \end{cases}$$

Finally, these cases can be combined to give the very simple gradient

$$\frac{\partial}{\partial w^j} \log P(Y = y|X = \mathbf{x}, \mathbf{w}) = (y - p)x^j$$

So, taking a small step in this direction would be to increment $\mathbf{w}$ as follows:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \alpha(y - p)\mathbf{x}$$

where α is the learning rate. Compare this to the perceptron update rule: it's not very different. Note that the reason we are providing these intermediate steps here is for you to better understand the algorithm. For your implementation, you do not need the detailed derivation.

1.2 L_2-Regularization

Logistic regression tends to overfit when there are many rare features. One fix is to penalize large values of w^j, by optimizing, instead of just optimizing LCL (log-conditional likelihood) in training set (size I, index by i):

$$LCL = \sum_{i}^{I} \Big(y_i \log p + (1 - y_i) \log(1 - p) \Big)$$

we can add a L_2 norm:

$$LCL - \lambda ||\mathbf{w}||^2$$

Here λ controls how much weight to give to the penalty term. Then instead of

$$\frac{\partial}{\partial w^j} \log P(Y = y|X = \mathbf{x}, \mathbf{w}) = (y - p)x^j$$

we have

$$\frac{\partial}{\partial w^j} \log P(Y = y|X = \mathbf{x}, \mathbf{w}) - \lambda \sum_{j=1}^{d} (w^j)^2 = (y - p)x^j - 2\lambda w^j$$

and the update for $\mathbf{w}$ becomes

$$\mathbf{w^{(t+1)}} = \mathbf{w^{(t)}} + \alpha((y - p)\mathbf{x} - 2\lambda \mathbf{w^{(t)}})$$

.

1.3 Numerical Optimization

In this homework, you will need to implement your own numerical optimizer using the standard **stochastic gradient descent (SGD)** algorithm. Here is the algorithm:

Algorithm 1 The SGD algorithm for optimization

```
Initialize the weight vector $\mathbf{w}$, and the learning rate $\alpha$;
while objective function not converged do
    Randomly shuffle the order of the examples in the training set.
    for each example in the training set do
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha((y - p)\mathbf{x} - 2\lambda\mathbf{w})$
    end for
end while
```

To determine when to stop this optimization or the convergence, you need to keep track of the function value in $LCL - \lambda||\mathbf{w}||^2$. If the change in function value is less than a small number (e.g. 1.0×10^{-3} or 1.0×10^{-4}), it is likely that your algorithm has found the global optimum.

In this homework, you are NOT required to use an adapative learning rate. Therefore, although it takes longer time to converge, you only need to set α once. Typically the value of α should be small. You can try multiple values (e.g. 0.001, 0.01, 0.05, 0.1, 0.2 ...) and see which one converges faster.

1.4 Making Predictions

In the testing time for binary logistic regression, assuming $\mathbf{x}$ is a feature vector that you see in the testing time, you can calculate:

$$P(Y = y|X = \mathbf{x}, \mathbf{w}) = \begin{cases} \frac{1}{1+e^{-\mathbf{x}\cdot\mathbf{w}}} & \text{if } y = 1 \\ 1 - \frac{1}{1+e^{-\mathbf{x}\cdot\mathbf{w}}} & \text{if } y = 0 \end{cases}$$

1.5 Implementation Notes

You need to select the best regularization coefficient λ based on 10-fold cross-validation (CV) on training set. `http://en.wikipedia.org/wiki/Cross-validation_(statistics)` To do this, you can define some candidate λs (e.g. $\lambda = 1\times10^{-7}, 1\times10^{-6}, ..., 1$), run CV using each candidate λ, and then compare the averaged results of each fold in CV. You select the λ that gives you the best performance on training set, and use that best parameter, retrain the model on the entire training set, and make prediction on the development/test set.

You should implement your own logistic regression algorithm using Octave (very similar as Matlab, actually most of the time they are compatible with each other). If you want, you can also develop in Matlab, but you need to verify that you code also runs in Octave on *linux.andrew.cmu.edu* machines.

Your implementation should have at least three functions with the following signatures:

```
model = lr_train(Xtrain, Ytrain)
Pred_lr = lr_test(model, Xtest)
lr_run(Xtrain, Ytrain, Xtest, Ytest)
```

Here X, Y are sets of document feature vectors and labels respectively, *model* is the model parameter $\vec{W}$. *Pred_lr* is the predicated label for the input test data *Xtest* based on your model (its true labels contained in *Ytest*). The last lr_run function should basically call lr_train first and then lr_test, and store the final predication *Pred_lr* in **Pred_lr.mat**. Finally, it should print out your testing precision, recall and accuracy on screen.
You can use

```
save Pred_lr.mat Pred_lr;
```

```
fprintf('P:%.3f, R:%.3f, A:%.3f', precision, recall, accuracy);
```

to save the **.mat** file and output the three values on screen.

Note: Matlab or Octave's programs (scripts/functions) are saved as the .m files. However, files with a .mat extension contain Matlab formatted data, which can be loaded from or written to these files using the functions "load" and "save".

Hints: to make your program efficient, please use matrix multiplication and pre-allocation. `http://www.mathworks.com/help/matlab/matlab_prog/vectorization.html`

2 Data Set

For this assignment, you should download the handout data from `http://curtis.ml.cmu.edu/w/courses/images/3/3a/Handout.mat`.

If you load the **handout.mat** into Octave (or Matlab) with

```
load 'handout.mat'
```

there are four objects have been included: "Xtrain", "Ytrain", "Xtest", "Ytest"'.

Note that this time we provide you with **sparse matrices** rather than dense matrices as the input to your programs. Similar to the previous homework, Xtrain and Xtest are training and testing feature matrices. Ytrain, and Ytest are training and testing labels. Sparse matrices is more practical than dense ones: for many large-scale problems, there is no way to fit the large dense matrices into the memory. If you are unfamiliar with the properties of sparse matrix, you might want to use the $full$ command in Matlab/Octave, which converts the sparse matrices back to dense matrices (though of course this will slow your code down).

The task is called *sentiment analysis*. There are two different kind of documents:

- **positive** (labeled with 1), containing positive reviews.
- **negative** (labeled with 0), containing negative reviews.

You goal is to classify the positive/negative reviews using the logistic regression code you have written. Your prediction output $y \in \{0, 1\}$. Note that this dataset and the task is much more difficult than the dataset and the problem in previous homework. If your accuracy is around 70-80%, then this is a very reasonable performance.

3 Suggestions

- You might want to develop/test your code in Matlab, but you **must verify** that your programs work in octave as the last step.

- You do not need to install Matlab or Octave, since they are already installed on the clusters and andrew machines. (You can use the following command to connect to andrew linux machines:

  ```
  ssh yourandrewid@linux.andrew.cmu.edu
  ```

4 Deliverables

Submit your codes of algorithm implementations via AutoLab. You should implement the logistic regression algorithm by yourself instead of using any built-in functions. Also, you are NOT allowed to use any off-the-shelf optimizer.

You should upload your codes (including all your function files) along with a report, which should solve the following question:

Investigate the effects of regularization on your training data. For this purpose, try $\lambda = \{0, 1\times10^{-7}, 1\times10^{-6}, 1\times10^{-5}, 1\times10^{-4}, 1\times10^{-3}, 1\times10^{-2}, 1\times10^{-1}, 1\}$ and report the accuracy of cross-validation on the training set. Plot the accuracy as a function of the λs (x-axis should be "lambda" and y-axis should be "accuracy").

You should tar gzip the following items into **hw3.tgz** and submit to the homework 3 assignment under Autolab:

- lr_train.m
- lr_test.m
- lr_run.m
- and all other auxiliary functions you have written
- report.pdf

Tar gzip the files directly using "tar -cvf hw3.tgz *.m report.pdf". Do **NOT** put the above files in a folder and then tar gzip the folder. You do not need to upload the saved predicted labels (i.e. the .mat files). Please make sure your code is working fine under Octave before you submit.

5 Submission

You must submit your homework through Autolab via the "Homework3-submission" link. In this homework, we provide an additional tool called "Homework3-validation":

- Homework3-validation: You will be notified by Autolab if you can successfully finish your job on the Autolab virtual machines. Note that this is not the place you should debug or develop your **algorithm**. All development should be done on linux.andrew.cmu.edu machines. This is basically a Autolab debug mode. There will be **NO** feedback on your **performance or score** in this debugging mode. You have unlimited amount of submissions here.
- Homework3-submission: This is where you should submit your validated final submission. You have a total of 5 possible submissions. Your performance will be evaluated, and feedback will be provided immediately.