Web Economics Assignment

Jun Wang junwang@cs.ucl.ac.uk
Emine Yilmaz e.yilmaz@cs.ucl.ac.uk

Submission Deadline: Midnight 14th March 2016

This assignment is worth 5% of your total grade
Please submit assignment documents to Moodle

In this assignment, you will test regression and logistic regression trained via gradient descent on the Spambase data set.

Step 1: Download the Spambase dataset available from the UCI Machine Learning Repository. The Spambase data set consists of 4,601 e-mails, of which 1,813 are spam (39.4%). The data set archive contains a processed version of the e-mails wherein 57 real-valued features have been extracted and the spam/non-spam label has been assigned. You should work with this processed version of the data. The data set archive contains a description of the features extracted as well as some simple statistics over those features.

Step 2: Partition the data into 10 folds.

To estimate the generalization (testing) error of your learning algorithm, you will perform cross-validation. In *k*-fold cross-validation, one would ordinarily partition the data set randomly into *k* groups of roughly equal size and perform *k* experiments (the "folds") wherein a model is *trained* on *k*-1 of the groups and *tested* on the remaining group, where each group is used for testing exactly once. (A common value of *k* is 10.) The generalization error of the classifier is estimated by the *average* of the performance across all *k* folds.

While one should perform cross-validation with random partitions, for consistency and comparability of your results, you should partition the data into 10 groups as follows: Consider the 4,601 data points in the order they appear in the processed data file. Each group will consist of every 10th data point in file order, i.e., Group 1 will consist of points {1,11,21,...}, Group 2 will consist of points {2,12,22,...}, ..., and Group 10 will consist of ponts {10,20,30,...}. Finally, Fold *k* will consist of *testing* on Group *k* a model obtained by *training* on the remaining *k*-1 groups

Note: The Spambase data set consists of a large block of spam messages followed by a large block of non-spam messages. Depending on how you create your folds, your training and testing data sets will also have this property. For any *batch* learning algorithm, this is not a problem, since the learner or updates will look at the entire data set at once (consider, for example, batch gradient descent). However, for an *on-line* learner (such as stochastic gradient descent), this can be problematic, since the algorithm will process (and make updates) on all positive examples followed by all negative examples---convergence will likely be hindered. To improve convergence for stochastic gradient descent, you may consider *randomizing* the lines in your training set (i.e., after you have created Fold 1). There are a number of Unix tools that can accomplish this easily, such as `rl`, `sort -R`, and various short bash scripts.

Step 3: Precondition your data. Gradient descent will often perform much better on data that has been "normalized" so that the individual features are on a comparable scale. One commonly used normalization is the z-score, sometimes called the standard score. To compute the z-score corresponding to a feature value, one must first compute the mean *mu* and standard deviation *sd* of the feature. You should compute these values yourself, in code, but you can check your results against the Spambase page describing various simple statistics over those features.

Once you have computed *mu* and *sd* (separately for each feature), the z-score for any feature value *x* is simply (*x* - mu)/sd.

For example, Feature 1 has a mean of 0.10455 and a standard deviation of 0.30536. If a particular example has a Feature 1 value of 0.5, the corresponding z-score would be

(0.5 - 0.10455)/0.30536 = 1.295.

Note that the z-score is simply the number of standard deviations a feature value is above or below the mean value of that feature.

Precondition the Spambase dataset using z-scores as described above.

Step 4: Create a *linear regression* learner that is trained via *stochastic gradient descent*.

Initialization: Start with all weights being zero.

Learning rate parameter: One of the challenges in implementing gradient descent is choosing a good fixed learning rate *lambda* or devising a variable learning rate schedule. If the learning rate is set too low, then gradient descent will converge very slowly; on the other hand, if the learning rate is set too high, gradient descent may actually diverge. A compromise is to set an initial "fast" learning rate and to decrease the learning rate as gradient descent converges.

For this assignment, you will explore the effect of different fixed learning rates on gradient descent. (You are welcome to explore the use of a variable learning rate, if you like.)

Step 5: Train your linear regressor. Try various fixed learning rate parameters over a wide range (e.g., 1, 0.1, 0.01, 0.001, 0.0001, etc.), and train your linear regressor via stochastic gradient descent. Explore the phenomenon discussed above, that when the learning rate is too large, gradient descent will diverge, and when the learning rate is too small, gradient descent will converge, but very slowly.

In order to analyze your convergence rate, after each complete pass through the training data, you should compute the mean squared error value at each training iteration (epoch).

Note: One would not ordinarily evaluate the error function after every pass through the data, as this effectively doubles the computation required by gradient descent. We are doing this in order to explore the effect of the learning rate parameter on the convergence of gradent descent.

1. Run your gradient descent algorithm until your error function appears to have converged. (Specify your convergence criterion.)

2. Repeat the above for both a lower and a higher learning rate parameter. (You may choose the lower and higher rate, but please specify it.)
3. Create one plot showing all three learning curves, i.e., mean squared error vs. iteration number for each of the three learning rates.

Step 6: Evaluate your results.

1. *ROC curve:* You will have trained three learners. Pick one of them (presumably the one with the lowest error) and plot an ROC curve corresponding to its performance on the test data.
2. *AUC:* An ROC curve visualizes the tradeoff between false positive rate and true positive rate (which is 1 minus the false negative rate) at various operating points. One measure for summarizing this data is to compute the *area under the ROC curve* or AUC. The AUC, as its name suggests, is simply the area under the ROC curve, which for a perfect curve is 1 and for a random predictor is 1/2. This area has an interesting probabilistic interpretation: it is the chance that the predictor will rank a randomly chosen positive example above a randomly chosen negative example.

   The AUC can be calculated fairly simply using the trapezoidal rule for estimating the area under a curve: If our *m* ROC data points are

   $(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)$ in order, where

   $(x_1, y_1) = (0,0)$ and $(x_m, y_m) = (1,1)$

   then the AUC is calculated as follows:

   $$(1/2) \text{ sum}_{k=2 \text{ to } m} (x_k - x_{k-1}) (y_k + y_{k-1}).$$

   Calculate the AUC of your classifier.

Step 7: Batch gradient descent. Repeat Steps 4 through 6 above for *batch gradient descent*. Note that this should involve only a minor change in your code.

Compare the convergence rates for batch vs. stochastic gradient descent. How many passes through the data are required to obtain a "good" predictor and/or mean squared error?

- **Logistic Regression**

Repeat Steps 4 through 7 above for *logistic regression* with both stochastic and batch gradient descent. Note that this should involve only a minor change in your code, as discussed in class.

- **Prepare a Report**

You should prepare a report describing your results above for linear regression and logistic regression with stochastic and batch gradient descent. You should also submit your code.