Large-Scale Machine Learning EECS 6898, Homework 3

Junfeng He and Sanjiv Kumar

November 2, 2010

Notes
20 points in total. 2 points will be dropped for each late day. There are two bonus questions worth 6 points.
The due date is 23:59, Nov. 16, 2010.
Feel free to discuss with anyone. But you should answer the questions, complete the programming implementations, and write the report totally on your own. You can not download code from web and use it, unless specified in the question.

1 Review Exercises (4 points)

1.1 Gradient Descent (2 points)

Prove the rate of convergence for Gradient Descent method is linear.

1.2 Newton's method (2 points)

Prove the rate of convergence for Newton's method is quadratic.

1.3 Bonus Question: Perceptron (2 points)

Prove Perceptron method will always converge if data is linearly separable.

2 Programming assignments (16 points)

Please write an experimental report with your results, observations, etc., together with a brief (a couple sentences) description of each module in your code. Your submitted file should be named as "hw3_uni_name.zip", e.g., "hw3_jh2700_Junfeng.zip", which contains your code and your report. Please do not include data file or intemediate result files in your submission, but store them in your computer. We may request them if necessary. In your report, do not just show your results. Please try to provide your observations, analysis of the results, etc. In other words, show what you have learned from the experiments.

You can use any programming language for the experiments, but Matlab (especially 64-bit Matlab) is highly recommended, which would really make your life much easier.

You are given 1 dataset (MNIST), which contains 10,000 training points and 1000 test points in Matlab format. Each training or test data point has a corresponding binary label $y = 1$ or -1, to represent whether it is digit number 3.

Notes: Each point in the data set is a 784-dim vector, representing the $28 * 28 = 784$ pixels for a handwritten digit image. You can use matlab commands like *im=reshape(x, 28,28);imshow(im);* to see the handwritten digit image, where x is a 784-dim vector.

2.1 Optimization (10 points)

Implement Kernel Logistic Regression with L2 regularizer using empirical kernel map, i.e., optimize,

$$J(w) = -\sum_{i=1,...,N} log(\sigma(y_i w^T k_i)) + \lambda w^T w$$

to get w. Here k_i is a column vector such that $k_i = [k(x_i, x_1), ..., k(x_i, x_j), ..., k(x_i, x_N)]^T$. Here y_i is the given label for each data x_i. Note that function σ is defined as $\sigma(v) = 1/(1 + e^{-v})$. Use RBF (Gaussian) kernel with the same parameter as in the previous assignments.

After w is obtained, for any test data x, compute $p(y = 1/x) = \sigma(w^T k_x)$, where $k_x = [k(x, x_1), ..., k(x, x_j), ..., k(x, x_N)]^T$. if $p(y = 1/x) > 0.5$, the predicted label $y = 1$, otherwise, $y = -1$. Report the accuracy, i.e., the percentage of test data whose predicted label and given groundtruth label is the same.

For optimization of $J(w)$, use the following methods. Experiment with various step sizes and pick that works the best for you. Compare how the value of the cost function decreases with time for different methods. Stop the iterations, if the gradient becomes smaller than epsilon (say, 1e-5).

2.1.1 Gradient descent (3 points)

2.1.2 Stochastic gradient descent (3 points)

For each iteration, use p points to estimate the gradients. Experiment with two values: p = 1, 100.

2.1.3 BFGS (4 points)

Note that you need to keep an estimate of the inverse Hessian. Since the effective data dimensionality with empirical kernel map is $d = n = 10,000$, to reduce the memory needs, randomly sample 4000 points from the training set (2000 each from positive and negative classes). Just use these points to decribe the empirical kernel map and construct the approximation of inverse Hessian using BFGS method. Now you will need to store just 4000x4000 matrix.

2.1.4 Bonus Question: Limited-Memory BFGS (4 points)

Try to repeat the same experiment as for BFGS above except that use a small number of vectors (experiment with a couple of choices) to approximate inverse Hessian. Also compare the convergence with time, and accuracy with other methods mentioned above.

2.2 Kernel linearization (6 points)

Redo the above problem, but instead of using empirical kernel map, use randomized methods to do kernel linearization i.e., approximate $k(x_i, x_j)$ as $k(x_i, x_j) \approx z(x_i)^T z(x_j)$. Then, directly optimize,

$$J(w) = -\sum_{i=1,\ldots,N} log(\sigma(y_i w^T z(x_i))) + \lambda w^T w$$

Use BFGS method implemented in Sec 2.1.3 for optimization, and experiment with a few choices of dimensionality (D) of $z(x)$. Compare the convergence time and accuracy with the methods used in Sec 2.1 for a reasonable choice of D. Note that you should use all the 10,000 samples for training this time since the inverse Hessian is of size $D \times D$ where D should be in hundreds.