# CS534 — Implementation Assignment 1 — Due April 23 in class, 2012

## General instruction.

1. You are strongly recommended to use Matlab for your implementation. However, Java, C/C++ are also accepted.
2. You can work solo or in team of 2 people. Each team will only need to submit one copy of their solution.
3. Your source code will be submitted through the TEACH site

`https://secure.engr.oregonstate.edu:8000/teach.php?type=want_auth`

Please include a readme file, which should contain the team member information.
4. You will also need to bring a hard copy of your report to the class on the due date. Similarly, clearly indicate the team members on your report.

## Linear regression

For the first part of the assignment, you need to implement the *batch gradient descent* and the *stochastic gradient descent* algorithms for linear regression. In particular, for stochastic gradient descent, given the training data, you will go through the training data (in random order) one example at a time and performs the update at each step, whereas for batch gradient descent, you perform the update only after going through all training examples. Please perform the following tasks for your report.

1. Please test your implementation on the provided regression datasets (will be posted soon). Note that you should learn from the training set and predict for the test set. Report the weight vector learned by your algorithm for each data set, and its Sum of Squared Error (SEE) for the corresponding test set. If your result varies significantly depending on the initial weights (seeds) used, please report the averaged results across 10 random runs with different initial seeds.

2. We are interested in testing the convergence behavior of the two different algorithms. To achieve this, you will start the two algorithms with the same initial seeds and plot the Sum of Squared Error on the training set as a function of the number of updates performed. Obviously we expect to perform many more updates for stochastic gradient descent. To make the two curves comparable, we will consider one update for batch gradient descent as equivalent to $n$ updates for stochastic gradient descent where $n$ is the total number of examples in the training data set. Please repeat this process for different initial seeds and comment on whether you observe a general trend. Please present in your report one or more sets of convergence curves that are representative based on your observation.

**Further clarifications.**

**Learning rate.** To make direct comparison between the batch and stochastic gradient descent methods, please use the same learning rate for the two methods. In particular, if the learning rate for batch method is 1, i.e., $\mathbf{w}(t+1) = \mathbf{w}(t) - \sum_{m=1}^{n}(\hat{y} - y_m)\mathbf{x}_m$, you should also use the same learning rate of 1 for stochastic gradient descent, i.e., $\mathbf{w}(t+1) = \mathbf{w}(t) - (\hat{y} - y_m)\mathbf{x}_m$. It is typical to use a schedule for learning rate, that is to decrease the learning rate gradually as learning goes on. However, to simplify things for this task, I suggest using a small constant learning rate that is proportional to $\frac{1}{n}$, where $n$ is the total number of instances. You may need to try out different values to avoid none-convergent behavior.

**Convergence test.** Convergence test can be performed based on the magnitude of gradient for the batch method. However, this is not very useful for the stochastic gradient descent method. A common approach is to test the change (e.g., the reduction of training data SSE, or the change of the weight vector) produced in the last $l$ steps, where $l$ is fixed number, e.g., 100. If the change is small enough, we consider it as evidence of convergence. Again, to simplify things, one suggestion is to use just a fixed number of epoches (an epoch is one round of training through all training examples) for both methods, such that you can directly compare them. This way, you don't have to calibrate the strictness of the convergence test for different methods.

In any case, feel free to explore different choices for the above two settings. These suggestions are provided to help make things easier for you.

## Perceptron and voted perceptron

For the second part, you will code the *batch perceptron* and *voted perceptron* algorithms. Note that for voted perceptron training, you should perform a random shuffle of the input training examples in each training epoch, such that we are not stuck with a particular ordering of the training examples. You are provided with two data sets for this task. You will test your batch perceptron algorithm on the first data set (twogaussian), which contains two linearly separable gaussian classes. Your write up needs to include:

1. A plot of the classification error on the training set as a function of the number of training epoches. Note that one epoch of training goes through the full training set exactly once.
2. A scatter plot of the training data using different colors for different classes. On this scatter plot, please plot the final linear decision boundary learned by your perceptron algorithm (please also provide the weights output by your perceptron algorithm).

You will test your voted perceptron implementation on the second data set iris-twoclass. This is based on a commonly used bench-mark data set iris. In particular, we extracted two classes, and two input features from the original problem. You need to include the following in your write-up:

1. A plot of the classification error on the training set as a function of the number of training epoches (up to 100 epoches). Note that one epoch of training goes through the full training set exactly once.
2. Please visualize the final decision boundary produced by your voted perceptron algorithm (trained for 100 epoches). Note that to produce this decision boundary, one strategy is to sample a large number of $\mathbf{x}$ points on a fine grid in the input space, and compute their predicted output. You can then visualize the decision boundary by plotting these points using different colors based on their predicted classes.
3. Note that for voted perceptron, there is a simple modification people often use in order to avoid storing all intermediate weight vectors. That is to compute

   $$\mathbf{w}_{avg} = \sum_{n=0}^{n=N} c_n \mathbf{w}_n$$

   and use the following decision rule:

   $$h(\mathbf{x}) = sgn\{\mathbf{w}_{avg} \cdot \mathbf{x}\}$$

   Please compute $\mathbf{w}_{avg}$ based on the weights that are learned by your voted perceptron algorithm (100 epoches), and plot the linear decision boundary it produces on a scatter plot of the training data. Are these two decision boundaries equivalent? Why?

**Further clarifications for learning rate.** For batch perceptron, please use $\lambda = 1$, which is referred to as the fixed increment perceptron (see slide 12). For voted perceptron, we do not use a learning rate (see slide 21).