Richard Scott McKinney
Linear Classifiers Project

Generating Results/How to run

All of the results in the figures at the end of this report were generated using my program. Running requires no additional inputs and is run out of the Main class. You will then be prompted on the command line for which classifier/dataset/num updates you would like to see. The earthquake and house votes datasets are already in the Examples folder, any additional datasets you wish to test on should be put in the Examples folder.

Many times, when using the Logistic Threshold with the smaller numbers of updates (5,000 and 10,000) it never makes progress in the right direction, and never stabilizes at good weights. You may have to run the program around five times for those numbers of updates to get results that look like mine below. I touch on this more later in this writeup.

Linear Classifiers:

In this project I implemented Linear Classifiers, of both the Perceptron and Logistic variety. A linear classifier is a linear regression which (tries to) split linearly separable data into classes. The linear regression has weights for each of the input variables which affect the *accuracy* of the classification. Accuracy in this context means the percentage of correct classifications over the total number of classifications.

In order to adjust the weights to learn more accurate classifications, both types of linear classifiers use Gradient Descent with a decaying learning rate. The differences between Perceptron and Logistic classifiers comes with their thresholds and update rules. The primary difference is the thresholds, a Perceptron classifier uses a hard threshold learning rule, meaning if the distance between the line and a data point is ≥ 0 , it is classified as class 1, and otherwise it is class 0. Because of this, the update rule for perceptron learning is relatively simple:

$(w_i + \text{alpha}(y - h_w(x))*x_i)$

On the other hand, Logistic classifiers use a soft threshold, meaning the distance from the line is significant to the classification of each data point, and therefore influences the weights. The update rule for Logistic classifiers is also more complicated, because it must take into account classes other than 0/1 as it updates its weights.

My Implementation of LCs:

Because Perceptron and Logistic Classifiers have a lot of shared elements, the parent abstract class LinearClassifier.java contains a lot of their code. The only thing defined in the classes PerceptronClassifier.java and LogisticClassifier.java is their individual update rules and thresholds.

We learned in lecture that Perceptron Classifiers are generally more unpredictable and less robust to noise, and Logistic Classifiers the opposite, but slower. The results I found from my graphs supported this. Both usually performed relatively well (figure 1 (below)) on the clean earthquake data, the Perceptron Classifier even sometimes outperformed the Logistic

Classifier. However, on the noisy data (figure 2) the Logistic Classifier was much more predictable than the perceptron classifier.

The Logistic Classifier was often less predictable with smaller updates. This makes sense because Logistic Classifiers are slower and take longer to stabilize. Often times, with the smaller numbers of updates (5,000 and 10,000) the logistic classifier would never stabilize and perform horribly. However, with the larger number of updates, the logistic classifier is very robust, even with the noisy data set. You can see this trend in figure 1f, where it took the logistic classifier around 10,000 updates to make any progress at all.

Figure 1: Clean Data Set

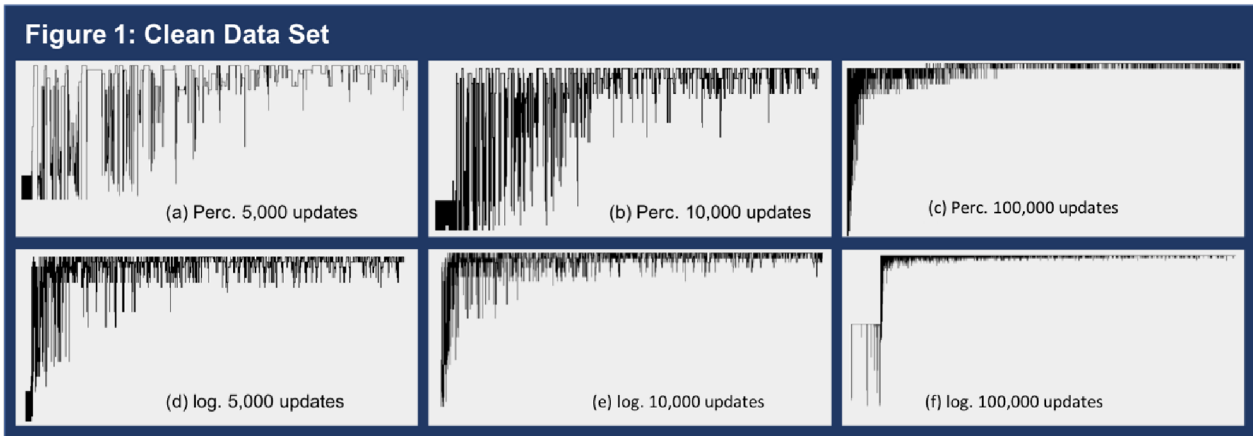

Figure 2: Noisy Data Set