CS446: Machine Learning	**Spring 2017**
Problem Set 3	
Zhuo Li(zhuol2)	*Handed In: February 22, 2017*

1 Number of examples versus number of mistakes

(a) The data sets generated are under *data/*. They can be generated by *exp1_generate_data.sh*.

(b) By running *exp1_tune_parameters.py*, it will tune all parameters with the five algorithms. Results are saved in *exp1_tuning.txt*.
Dataset $n = 500$ tuning accuracies with optimal parameters are:

- Perceptron: 97.52%
- Perceptron w/ margin: 99.44%
- Winnow: 99.88%
- Winnow w/ margin: 99.88%
- AdaGrad: 98.64%

Dataset $n = 1000$ tuning accuracies with optimal parameters are:

- Perceptron: 81.9%
- Perceptron w/ margin: 96.14%
- Winnow: 99.78%
- Winnow w/ margin: 99.86%
- AdaGrad: 99.02%

The optimal parameters are shown below

Algorithm	Parameters	Dataset n=500	Dataset n=1000
Perceptron	η	$\eta = 1$	$\eta = 1$
Perceptron w/margin	η γ	$\eta = 0.005$ $\gamma = 1$	$\eta = 0.03$ $\gamma = 1$
Winnow	α	$\alpha = 1.1$	$\alpha = 1.1$
Winnow w/margin	α γ	$\alpha = 1.1$ $\gamma = 0.001$	$\alpha = 1.1$ $\gamma = 2.0$
AdaGrad	η	$\eta = 0.25$	$\eta = 0.25$

(c) By running *exp1_plot.py*, it will train the whole dataset($N = 50000$) with five algorithms and record the accumulated mistakes. The number of accumulated mistakes for each algorithm with the optimal tuned parameters are in *exp1_mistakes.txt*.

Figure 1: $l = 10, m = 100, n = 500$

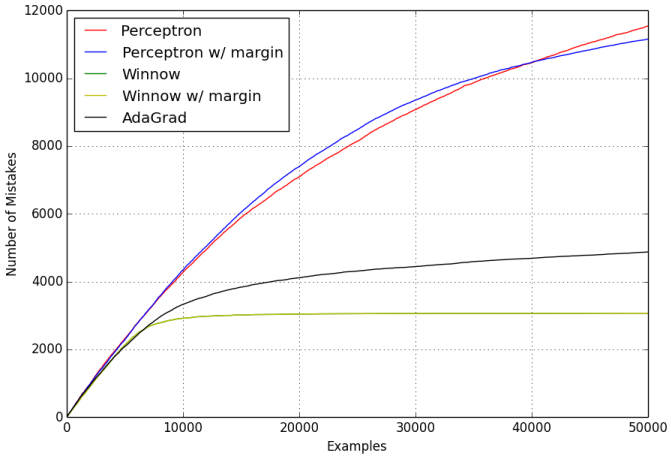
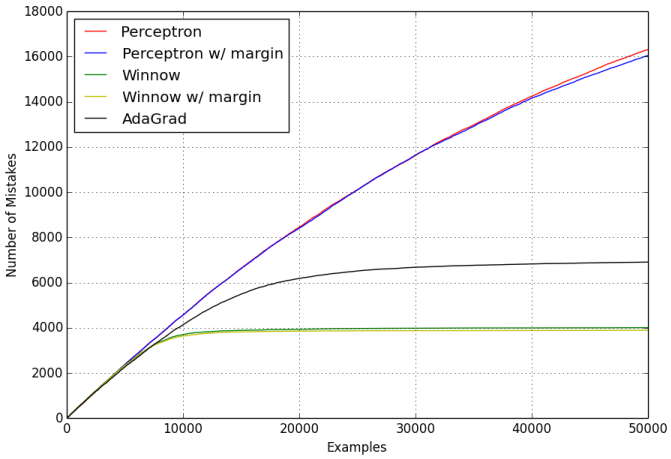


Figure 2: $l = 10, m = 100, n = 1000$



(d) Required graphs are shown above:

Discussion

- From the Figure 1 and Figure 2 (Tuning accuracy as well), we can easily find that when the dataset becomes larger (i.e. from $n = 500$ to $n = 1000$), the number of accumulated mistakes increased and accuracy decreases for the same algorithms. Take AdaGrad algorithm as an example. when $n = 500$, it has about 4,800 mistakes at example 50,000 while 7000 when $n = 1000$. This is because as we increase n, we make the function sparse. It is more difficult for our algorithms to find relevant attributes behind, thus taking more mistakes to learn.
- From the two figures, we can see as the number of examples increases, the number of mistakes increases as well for all algorithms. And the curves bend to be nearly horizontal as the the number of mistakes tends to a bound. Once approaches the

bound, the algorithm will rarely make more mistakes. The Winnow algorithm seems to have the best mistake bound, followed by AdaGrad, while Perceptron plays the worst.

- Both Perceptron and Winnow perform better with margin. Perceptron is even more obvious on the figures as the blue line tends to be under the red line (i.e. mistake # tends to the bound earlier). This is trivial as the margin make the algorithm more robust and in higher tolerance to noise on new data.

2 Learning curves of online learning algorithms

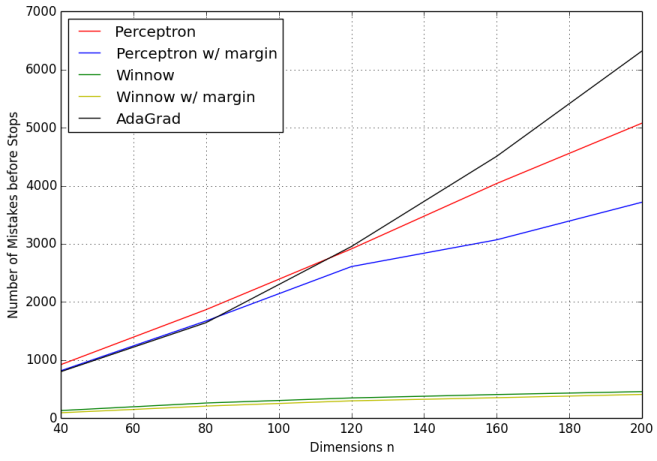
(a) The data sets generated are under *data/*. They can be generated by *exp2_generate_data.sh*.

(b) By running *exp2_tune_parameters.py*, it will tune all parameters with the five algorithms. Results with accuracies at the optimal parameters are saved in *exp2_tuning.txt*. The optimal parameters are shown below

Algorithm	Parameters	n=40	n=80	n=120	n=160	n=200
Perceptron	η	$\eta = 1$	$\eta = 1$	$\eta = 1$	$\eta = 1$	$\eta = 1$
Perceptron w/margin	η γ	$\eta = 0.25$ $\gamma = 1$	$\eta = 0.25$ $\gamma = 1$	$\eta = 0.25$ $\gamma = 1$	$\eta = 0.03$ $\gamma = 1$	$\eta = 0.03$ $\gamma = 1$
Winnow	α	$\alpha = 1.1$	$\alpha = 1.1$	$\alpha = 1.1$	$\alpha = 1.1$	$\alpha = 1.1$
Winnow w/margin	α γ	$\alpha = 1.1$ $\gamma = 2.0$	$\alpha = 1.1$ $\gamma = 2.0$	$\alpha = 1.1$ $\gamma = 2.0$	$\alpha = 1.1$ $\gamma = 2.0$	$\alpha = 1.1$ $\gamma = 2.0$
AdaGrad	η	$\eta = 1.5$	$\eta = 1.5$	$\eta = 1.5$	$\eta = 1.5$	$\eta = 1.5$

(c) By running *exp2_plot.py*, we record the number of mistakes we made before each algorithm stops W, and plot it versus the number of variables n. Here we use the default convergence criterion $R = 1000$. The learning curves are shown in Figure 3.

Figure 3: Learning curves of online learning algorithms



Discussion

- We can see that Perceptron and Winnow with margin perform better than those without as they make less numbers of mistakes before they get a sequence of $R = 1000$ correct predictions. This is obvious as the margin strengthen the corresponding algorithms which separates the datasets better.
- We can also find that all the algorithms makes more mistakes when the number of variables n increases. This is because when n increases, the function becomes sparser, thus making it even harder for the algorithms to learn.

- Another thing is for the three algorithms, AdaGrad performs the worst (with more mistakes before stop), followed by Perceptron and Winnow here works the best. This may because Winnow has the smallest mistake bound, so it doesn't increase much as it soon approach the bound.

3 Use online learning algorithms as batch learning algorithms

(a) The data sets generated are under *data/*. They can be generated by *exp3_generate_data.sh*.

(b) By running *exp3_tune_parameters.py*, it will tune all parameters with the five algorithms. Results with accuracies at the optimal parameters are saved in *exp3_tuning.txt*.

(c)(d) The optimal parameters and accuracies on clean test set are shown below

Algorithm	m=100		m=500		m=1000	
	acc.	params.	acc.	params.	acc.	params.
Perceptron	90.61%	$\eta = 1$	88.6%	$\eta = 1$	74.88%	$\eta = 1$
Perceptron w/margin	97.33%	$\eta = 0.03$ $\gamma = 1$	88.6%	$\eta = 1.5$ $\gamma = 1$	82.3%	$\eta = 0.03$ $\gamma = 1$
Winnow	95.75%	$\alpha = 1.1$	87.07%	$\alpha = 1.1$	76.25%	$\alpha = 1.1$
Winnow w/margin	96.36%	$\alpha = 1.1$ $\gamma = 0.006$	88.39%	$\alpha = 1.1$ $\gamma = 0.001$	74.29%	$\alpha = 1.1$ $\gamma = 1.0$
AdaGrad	99.98%	$\eta = 0.25$	99.73%	$\eta = 1.5$	84.58%	$\eta = 1.5$

Discussion

- As m increases, we have more relevant variables in the function which get more subtle for the algorithms to learn during the same number of traning cycles. So it is not surprising that accuracies of all algorithms somewhat go down.
- We can also find Perceptron and Winnow with margin ususally get higher accuracies than those without. This is because the margin makes the algorithm more robust to noises. There is a exception. At $m = 1000$, Winnow with margin get a relative lower accuracy than without. This may because the parameters are not truly the best. Even though we tune the parameters, there are still cases when a specific parameter happens to be the best, or one the best in the 10% tuning train set but may not be the best of the entire training set. This happens due to the randomly selection of subexamples.
- If we take a look at the accuracies when tuning the parameters in *exp3_tuning.txt*, we can also find the accuracies here is much less than from the first two experiments. This is because we have noise here in the experiment which distract our training to some degree.

4 Bonus

The data sets generated are under *data/*. They can be generated by *exp4_generate_data.sh*. And then we can train the algorithms over the whole dataset and keep track of the misclassification error and hinge loss. The graphs plotted as Figure 4 and Figure 5.

Figure 4: Misclassification Error

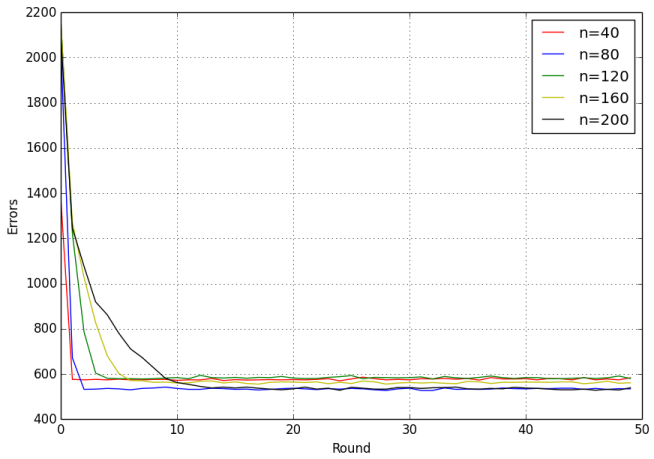
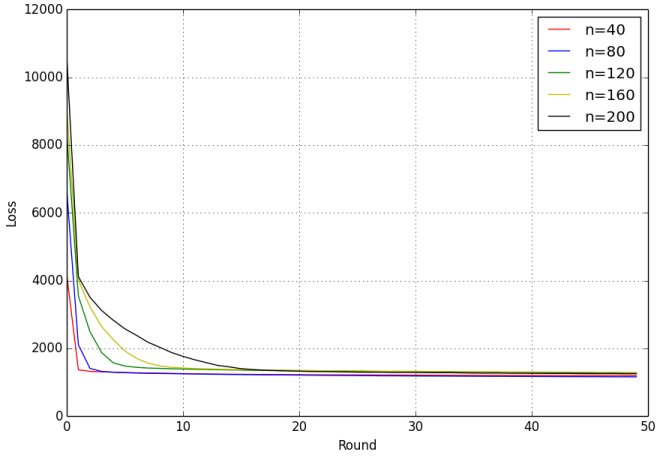


Figure 5: Hinge Loss



Discussion

- In the first round, the Misclassification Error and Hinge Loss decreases dramatically. This is because the initial weight vector is extremely different from what it should be. So this is like a "setup pulse". Then both the Misclassification Error and the Hinge Loss slowly decrease which tends to a bound.

- There are some fluctuations in the figure of Misclassification Error, while it is smoother. This is because the 0-1 loss has no margin and the Hinge Loss tend to separate the data by a margin of 1. This way, there may be less fluctuations.
- Larger the number of variables n, slower converge to the bound. This is because a larger number of variables make the function sparser, thus harder for algorithms to learn the weight. It takes more rounds to reach the bound as well. The bound has nothing to do with n.