CS446: Machine Learning **Spring 2017**

Problem Set 3

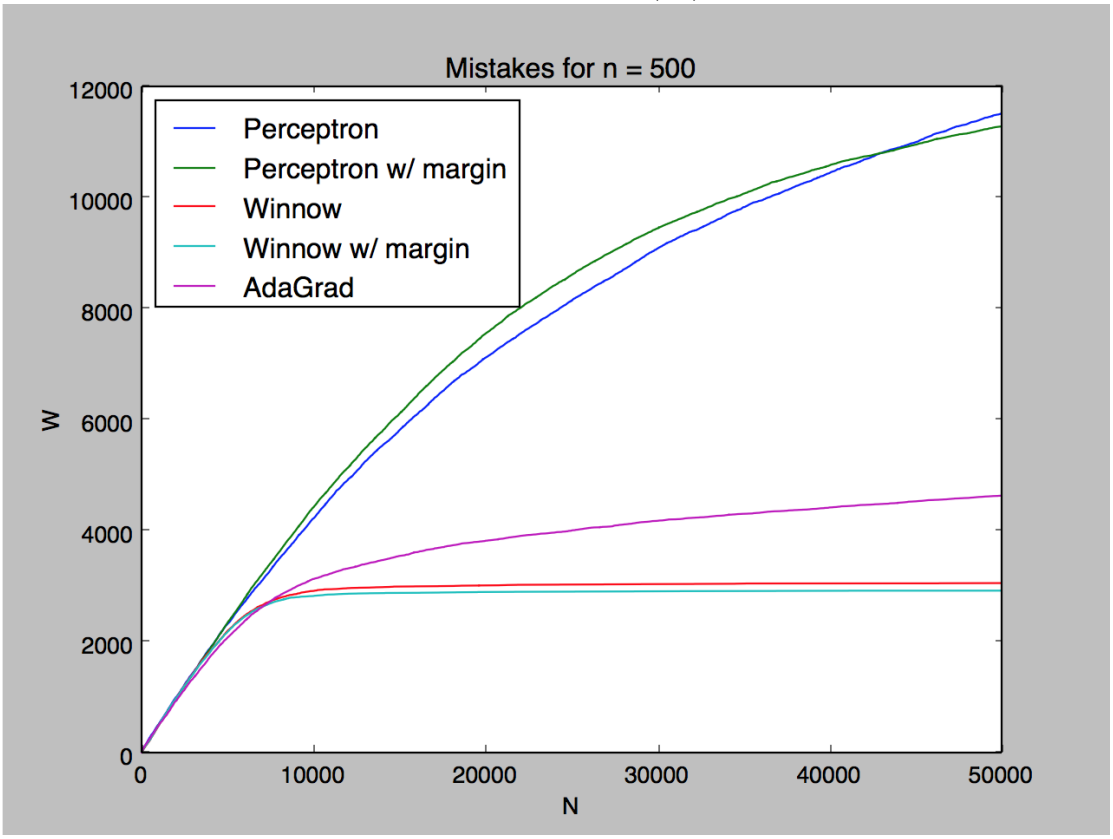
Kexin Hui *Handed In: February 28, 2017*

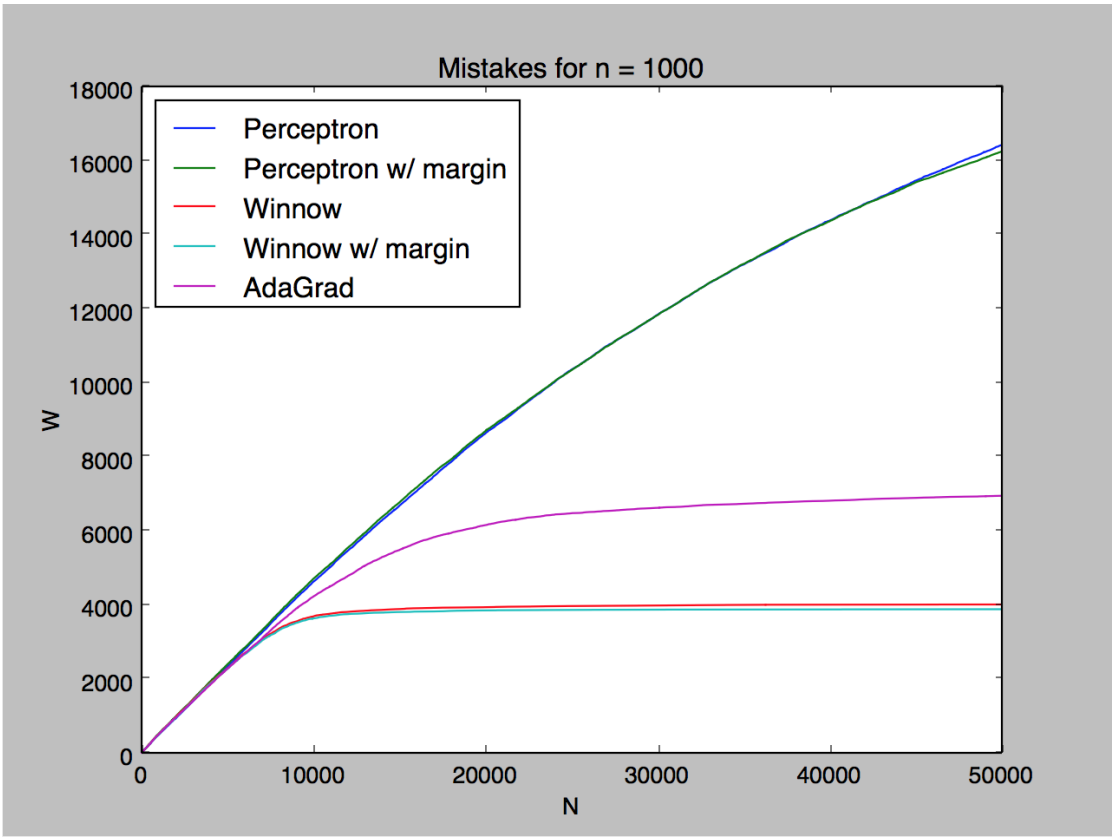
1. Answer to problem 1

   In the beginning each running of my python script *online_learning*, I randomly generate two dataset $\{x_1, y_1\}$ and $\{x_2, y_2\}$ from all examples for training and testing. Five algorithms are implemented by five individual functions, namely *Perceptron*, *Perceptron_with_margin*, *Winnow*, *Winnow_with_margin* and *AdaGrad*. Also function *testing* is used to evaluate and print out the accuracy for each algorithm. In order to tune the parameters, I manually tried different combinations of parameters to record the optimal ones.

Algorithm	Parameters	Dataset n=500	Dataset n=1000
Perceptron	None	None	None
Perceptron w/ margin	η	0.005	0.005
Winnow	α	1.1	1.1
Winnow w/ margin	α, γ	1, 1, 2.0	1.1, 2.0
Adagrad	η	1.5	1.5

Plots for cumulative number of mistakes made (W) on N examples are shown below.

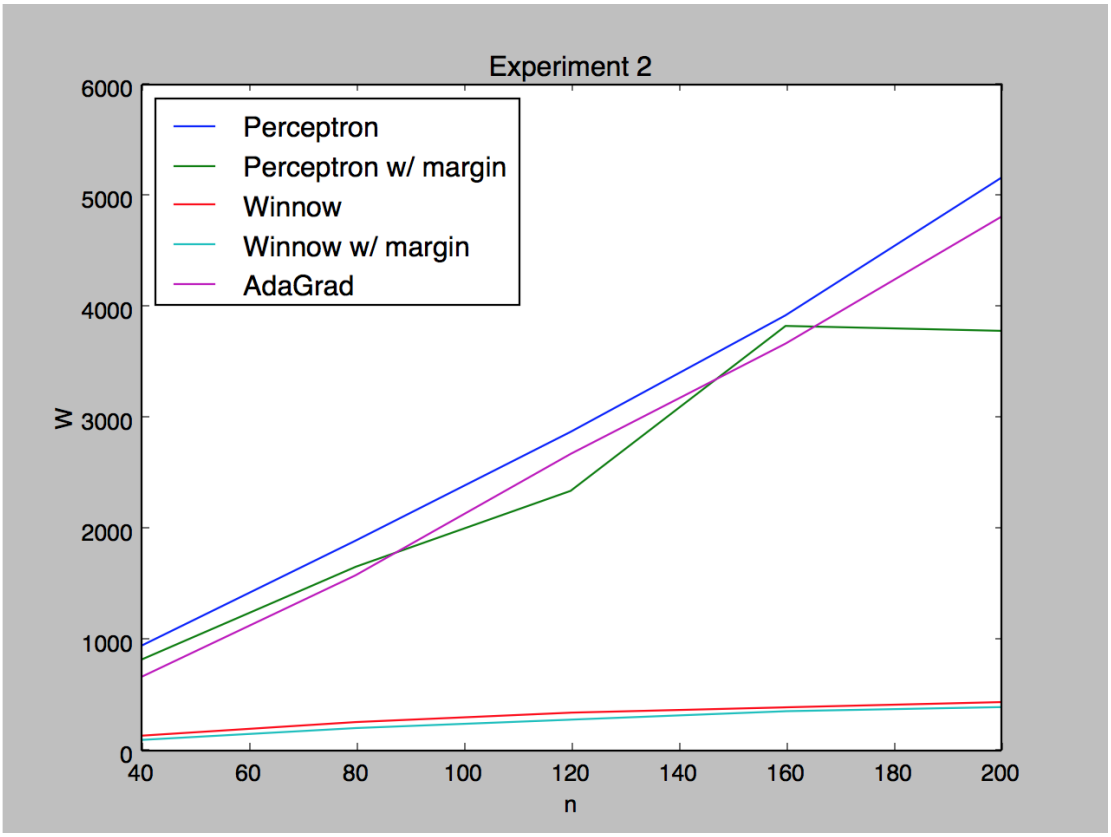




Two plots for the dataset n=500 and n=1000 look similar as we can see. Basically, number of mistakes for *Perceptron* with and without margin grows linearly. Yet for *Winnow* with and without margin, the number of mistakes increase at first and get converged in the end. Generally speaking, $Perceptron > AdaGrad > Winnow$. That makes senses since *Perceptron*'s mistake bound is O(n) and *Winnow*'s mistake bound is O(klog(n)). On the other hand, with or without margin for *Perceptron* and Winnow doesn't make a big difference. Because the margin will mainly affect the robustness of the algorithm. *AdaGrad* takes advantage of historical information by keeping track of hinge loss. It is similar to *Perceptron* with margin 1 but performs better.
As dataset size n increases, *Perceptron* makes more mistakes when n= 1000 than it does when n = 500. This is also consistent with *Perceptron*'s mistake bound as O(n).

2. Answer to problem 2
Parameters tuning follows the same procedure as I do in Problem 1. The results are listed below.

Algorithm	Parameters	n=40	n=80	n=120	n=160	n=200
Perceptron	None	None	None	None	None	None
Perceptron w/ margin	η	0.25	0.03	0.03	0.25	0.03
Winnow	α	1.1	1.1	1.1	1.1	1.1
Winnow w/ margin	α, γ	1, 1, 2.0	1.1, 2.0	1.1, 2.0	1.1, 2.0	1.1, 2.0
Adagrad	η	1.5	1.5	1.5	1.5	1.5



10 cycles are chose for training in order to make sure the algorithm satisfies the converge criterion. Convergence criterion R is set to 1000. As we can see from the table, the optimal parameters don't change as the number of features in each example increases except the margin for perceptron algorithm changes slightly. According to the graph, both *Winnow*s make the least number of mistakes before they converge. They are close to each other yet *Winnow w/ margin* performs a little bit better than *Winnow*. Both *Perceptron*s and also *AdaGrad* makes much more mistakes than *Winnow*s, as they all increase linearly. *Perceptron* is the worst and *Perceptron w/ margin* works a little bit better.*AdaGrad* is close to *Perceptron w/ margin*, sometimes better and sometimes worse, but always better than *Perceptron*. The results are consistent with our knowledge of mistake bound for *Perceptron* O(n) and *Winnow* O(klog(n)).

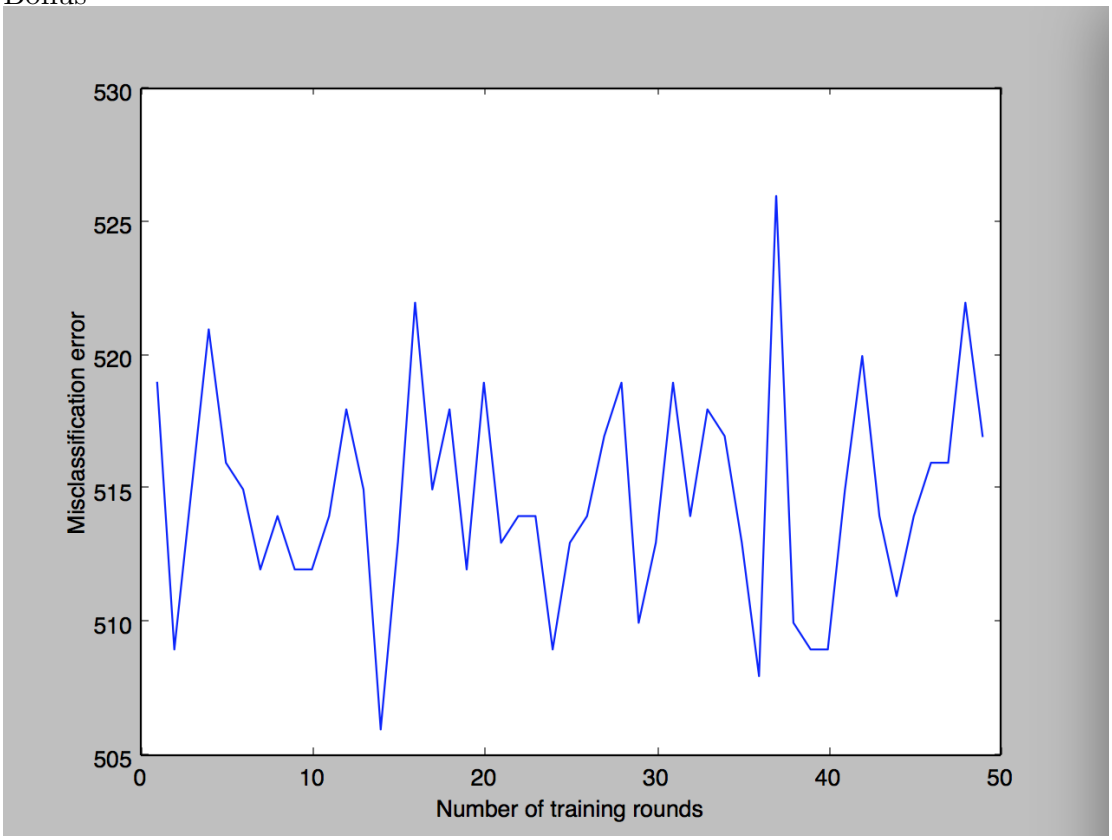
3. Answer to problem 3
numpy.save and *numpy.load* are used to generate the training data and testing data. All the parameters are trained in 20 cycles for each algorithm. The procedure for tuning the parameters is the same as what I do in Problem 1. The results are listed in the table below.

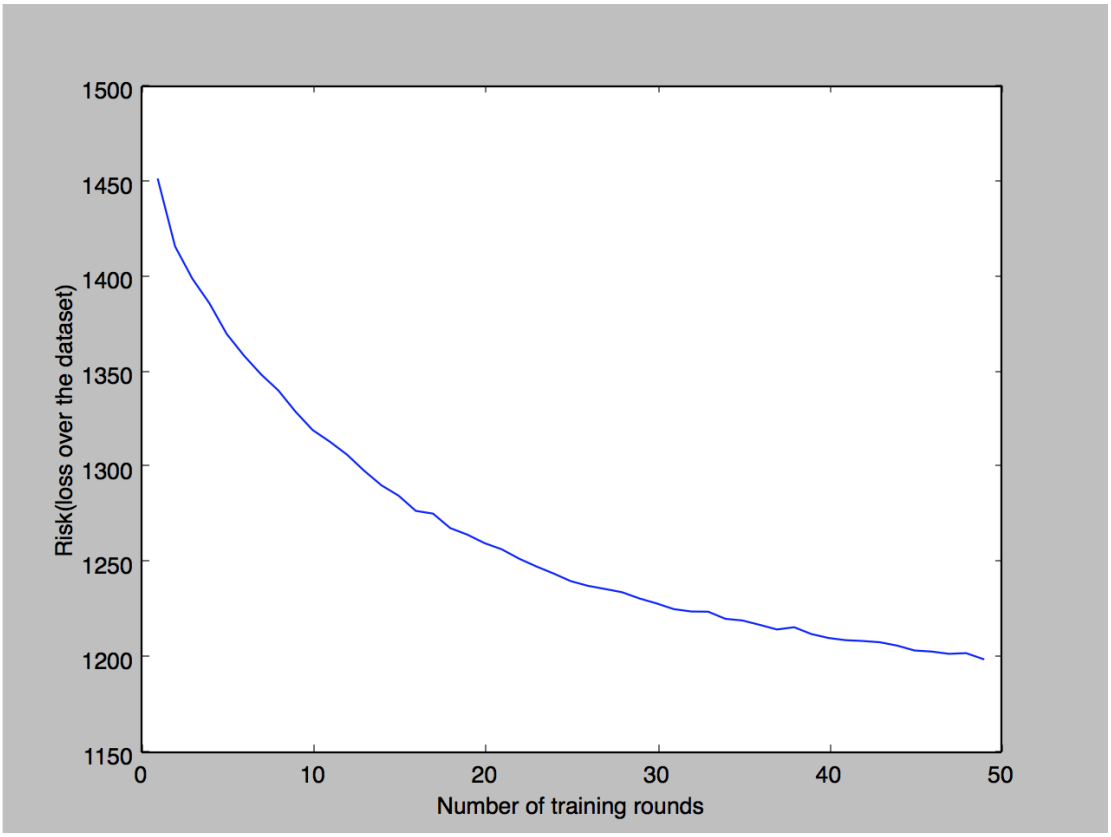
Algorithm	m=100		m=500		m=1000	
	Accuracy	Parameters	Accuracy	Parameters	Accuracy	Parameters
Perceptron	95.84%	None	89.85%	None	75.36%	None
Perceptron w/ margin	97.55%	η=0.005	87.12%	η=0.25	84.98%	η=0.03
Winnow	96.21%	α=1.1	87.4%	α=1.1	75.97%	α=1.1
Winnow w/ margin	94.84%	α=1.1, γ=0.006	89.57%	α=1.1, γ=0.006	76.41%	α=1.1, γ=0.006
AdaGrad	99.64%	η=1.5	86.48%	η=1.5	81.14%	η=1.5

As we can see from the table, the smaller m is, the better performance each algorithm performs. Namely, the accuracy for each algorithm decreases as m increases. On the other hand, as m changes, the optimal parameters stay the same in general. That makes sense because dimensionality is small relative to the algorithm. It is quite surprising to notice this time *Perceptron* has a little bit better performance than *Winnow* compared to the fact I learned from Problem 1 that *Perceptron* normally makes much more mistakes than *Winnow*. This suggests in the batch setting with noisy training data, mistakes are sort of important and reliable since every feature matters. *AdaGrad* also has higher testing accuracies since it is quite similar to *Perceptron* as I discussed before.

To conclude, *Perceptron w/ margin* and *AdaGrad* have the best performance among five algorithms. *Winnow* is okay but not that good.

4. Bonus





Graph for misclassification errors vs. number of training rounds is quite messy, and I cannot find a general trend for that. On the other hand, hinge loss is decreasing as the number of training rounds increases. It is consistent with the fact that *AdaGrad* is a algorithm aimed to minimize hinge loss.