CS446: Machine Learning **Spring 2017**

Problem Set 3

Haoran Wang *Handed In: February 28, 2017*

1. **Answer to problem 1**
   Number of examples versus number of mistakes

Algorithm	Parameters	Dataset = 500	Dataset =1000
Perceptron	N/A	N/A	N/A
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	0.25	0.25

Table 1: Parameter Choice for Five Algorithms

After the process of parameter tuning, the best parameter for each algorithm is shown in the Table 1. As for the cumulative number of mistakes made on N examples, we could represent it as a function of N. The graphs for n=500 and n=1000 are:

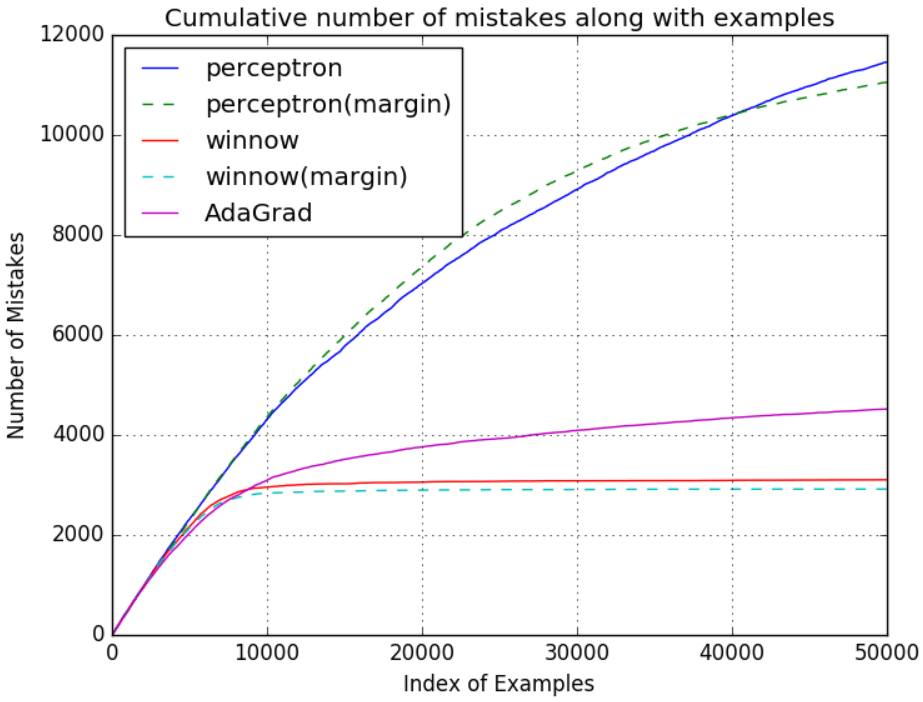

Figure 1: Cumulative mistakes made for n = 500

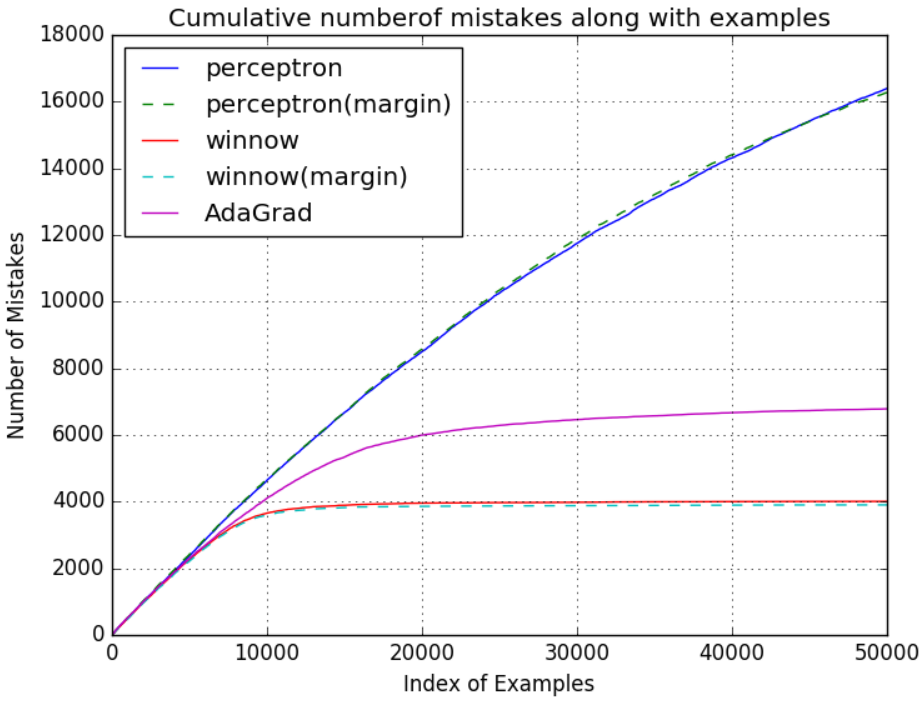

Figure 2: Cumulative mistakes made for n = 1000

Observations and Conclusions:

From both figures, we could see that the mistakes made by perceptron is approximately in a linear increasing way with the number of examples. However, the situation is changed to logarithmic increasing in winnow cases(with/without margin). And the number of mistakes made by AdaGrad is between the two perceptron and two winnow cases. AdaGrad is bounded like winnow instead of a linear increasing trend like perceptron. But it would take more examples to converge compared with the winnow cases.

As for the comparison between dense and sparse variables for n=500 and n=1000, we notice that the difference of the trend between two perceptrons is almost ignorable in the sparse case(n=1000), but the trend is obviously different in the dense case(n=500). And the it could also apply on two winnow cases. We notice that the winnow with margin algorithm makes less mistakes in dense case(n=500), but the winnow and the winnow with margin performs almost the same in sparse case(n=1000). As for AdaGrad algorithm, we notice that it take more examples to get a convergence trend in the sparse case(n=1000), and the performance looks like winnow though it is not as good as winnow cases.

2. **Answer to problem 2**
   Learning curves of online learning algorithms

Algorithm	Parameters	n=40	n=80	n=120	n=160	n=200
Perceptron	N/A	N/A	N/A	N/A	N/A	N/A
Perceptron (w/margin)	η	1.5	0.25	0.25	0.03	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

Table 2: Parameter for Five Algorithms with n increments by 40

The parameter used for the learning curve is:

$$R = 1000$$

Where R is the convergence criterion

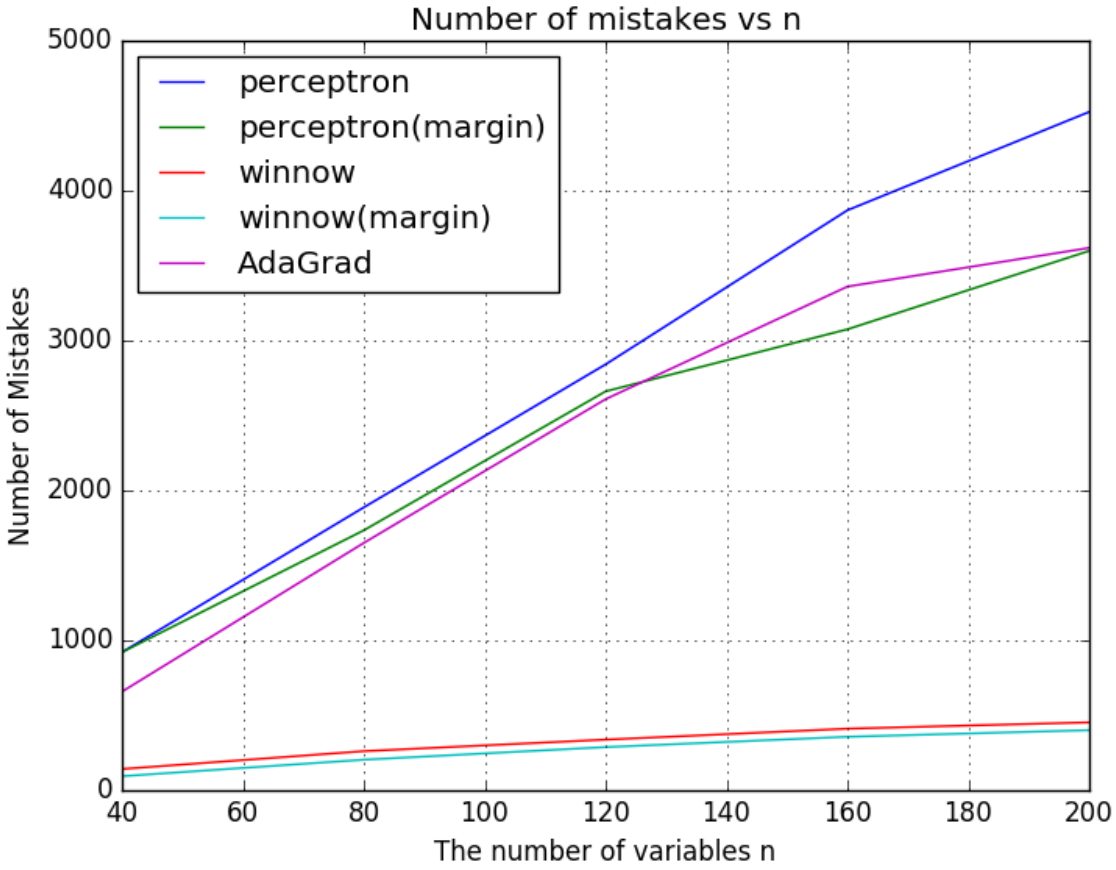

Figure 3: Learning Curves of Five Algorithms

Observations and Conclusions:

From the above figure, we could notice that winnow(with margins) makes the fewest mistakes for these dimensions(n). Though the winnow case makes the second fewest mistakes among these algorithms for different n, it is close to the winnow(with margin) case, which indicates the difference is not very large, they have similar performance on different dimensions(n) for convergence. So the dimensions have little influence on two winnow cases. As for perceptron(with margin) case, it makes less mistakes at these dimensions(n) than perceptron though they have similar linearly increasing trend. So this trend indicates that the perceptron algorithms are easily influenced by the dimensions of the functions to learn, which means it will make more mistakes before learning a good model if the variables has more dimensions. In addition, perceptron(with margin) performs a little bit better than perceptron for these dimensions(n). By the way, winnow and winnow(with margin) always made less mistakes than other algorithms. The number is only always below 1000.

As for AdaGrad algorithm, it has a linear trend for these dimensions. And the performance is influenced by the dimensions of possible variables like perceptron algorithm. As we can see from the figure 3, AdaGrad performs makes less mistakes compared with perceptron(with margin) for the first three dimensions(n=40, 80, 120), however, it gets worse for the last two dimensions(n=160, 200) than perceptron(with margin). Finally, it is a same trend for every algorithm: the number of mistakes goes up along with the number of dimensions(n). Because more dimensions, in other words, more variables in the input function to learn, more difficult to learn a good model. So we may make more mistakes in larger dimensionality of the variables.

3. **Answer to problem 3**
   Use online learning algorithms as batch learning algorithms

   The tuned parameters and resulting accuracy is shown as:

Algorithm	m=100		m=500		m=1000	
	acc.	params.	acc.	params.	acc.	params.
Perceptron	0.9836	N/A	0.8512	N/A	0.8106	N/A
Perceptron (w/margin)	0.9878	η =0.03	0.8790	η =0.25	0.8834	η =0.03
Winnow	0.9597	α =1.1	0.8664	α =1.1	0.7741	α =1.1
Winnow (w/margin)	0.9703	α=1.1, γ=0.3	0.8694	α=1.1, γ=0.3	0.7794	α=1.1, γ=0.006
AdaGrad	0.9992	η =0.25	0.9799	η =1.5	0.8661	η =1.5

Table 3: Parameters for m=100, m=500, m=1000

Observations and Conclusions:

Before any observation, one thing we need to clarify is that there may exists the same accuracy when doing the parameter tuning. So for this tie-breakers, I just select the random one with a little higher accuracy or the same accuracy(e.g. 0.9544 vs 0.9530, 1.0 vs 1.0). According to the table 3, we could notice that for each algorithm, when m = 100, it could achieve the best accuracy among these three experiments(m=100, m=500, m=1000). It indicates that the model is more easier to learn after training 20 epochs even if with noisy data. Besides, we notice that even after training 20 epochs, we still get errors in the test data. This phenomenon implies that the noisy training data has influenced all the algorithms, even the best accuracy cannot reach 1.0(0.9992 from AdaGrad). We could notice that with different m, which is the required underlying positive features for the entire feature set, the AdaGrad has the best accuracy. The reason I think it is because the η is good for this algorithm to maintain a range of different learning rate for each weight in the feature function.And we will add more weights to the corresponding feature x_i that contributes more as a positive feature. The different learning rate actually performs better in this problem.

Besides, we also notice that the perceptron with margin performs better than winnow and winnow with margin in this problem. And the performance of perceptron is similar to winnow. The possible reason is there are more positive features to learn when m gets larger(e.g. m=100, 500, 1000) compared with previous cases (e.g. m = 20). And another reason is that we trained the model 20 rounds instead of only once, which means perceptron can adjust its weights multiple times compared with previous cases, so it is reasonable that the feature functions $\vec{w}$ becomes as good as the that of winnow.

A general trend is that when m gets larger, the accuracy of five algorithms will go down. I think the reason is that we have more positive features to learn within n features, which causes the weights of these features hard to converge to the correct level in order to make a correct prediction. In other words, the weights of the features become evenly if we have a large m to learn. Besides, the noisy of the training data will have greater influence on the model if we have more positive features to learn now. So the noisy data counts more if we need to learn more features.

For different m, we notice the best parameter doesn't change a lot from these algorithms, which means that the number of positive features m has little impact on the choice of parameters. But it doesn't mean it has no influence on the parameter tuning. For example, we notice that the best parameter for perceptron with margin is different for m=100 and m=500,m=1000. So the noisy data is influencing

4. **Answer to Bonus question**
The parameter for generating data is chosen as: $l = 10$, $m = 20$, $n = 40$. The size of training data is 10000 and the data is generated with noise.

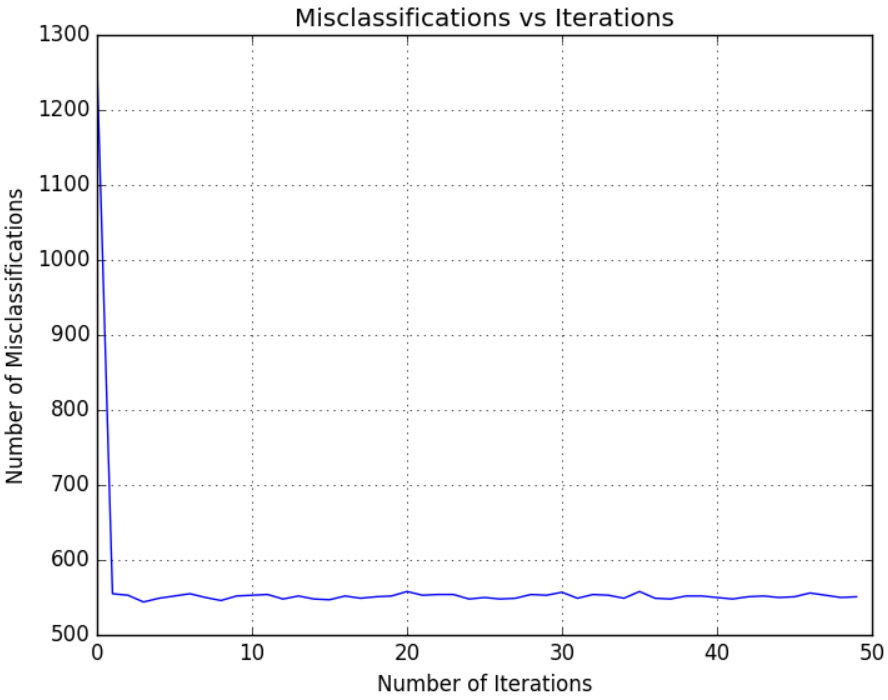

Figure 4: Misclassification error against the number of training rounds(general scale)

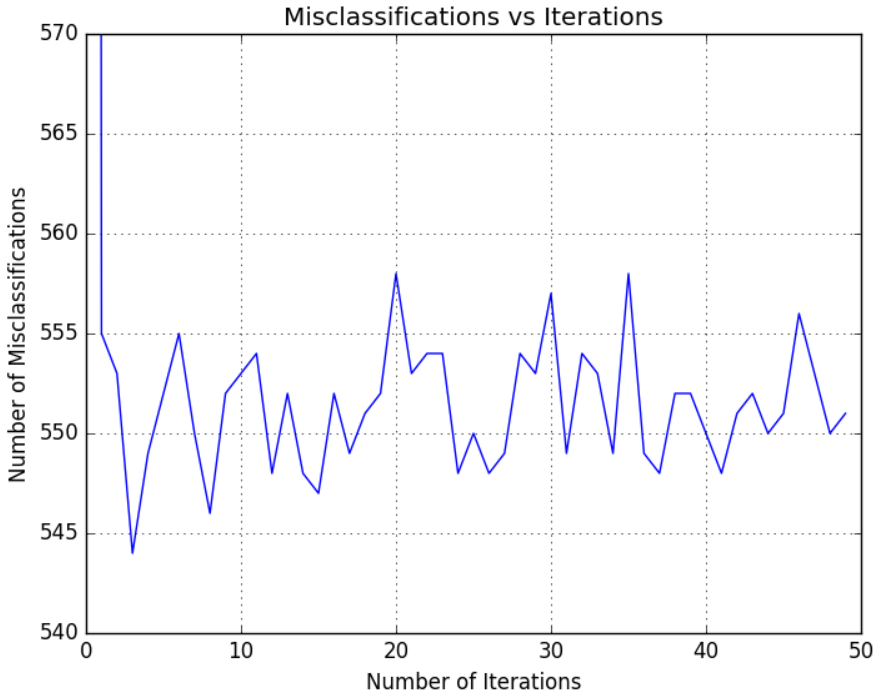

Figure 5: Misclassification error against the number of training rounds(detailed scale)

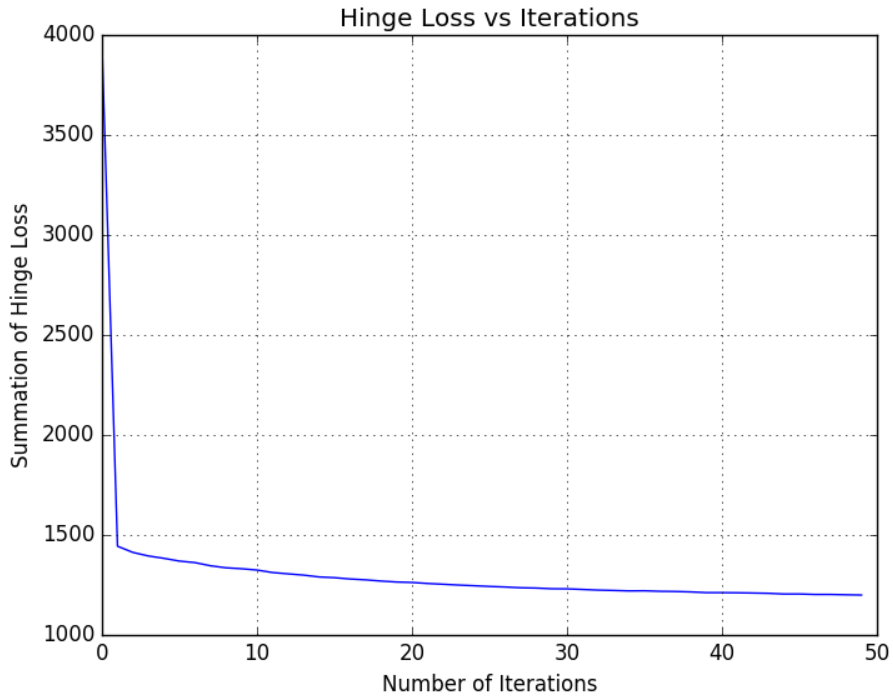

Figure 6: Risk against the number of training rounds(general scale)

Figure 7: Risk against the number of training rounds(detailed scale)

From the above figures, we could see that the risk(loss over dataset) is going down along with the number of rounds. So the trend is fixed as going down with the number of rounds. But the misclassification error is at a random pattern as shown in the figure 5. And we notice that the number of misclassification error is within a range(e.g. m∈(540, 560)) from the general scale in figure 4. The reason why there is a random pattern in the misclassification error is that the hinge-loss function provides a different update rule compared with the traditional 0-1 update rule. When $y_i \cdot (\vec{w}^T x_i + \theta) <= 1$, we will update the weights. However, in 0-1 based loss function, we update the weights when $y_i \cdot (\vec{w}^T x_i + \theta) <= 0$, which is exactly the same as the definition of misclassification(mistakes). Back to the hinge loss function, we may update the weights even if we don't make a mistake on this training example, so the update rule is different with the definition for a mistake. That's the reason why we get a random pattern along with the number of rounds. Mathematically speaking, the curve of the hinge loss function is always over the 0-1 loss function(misclassification error):

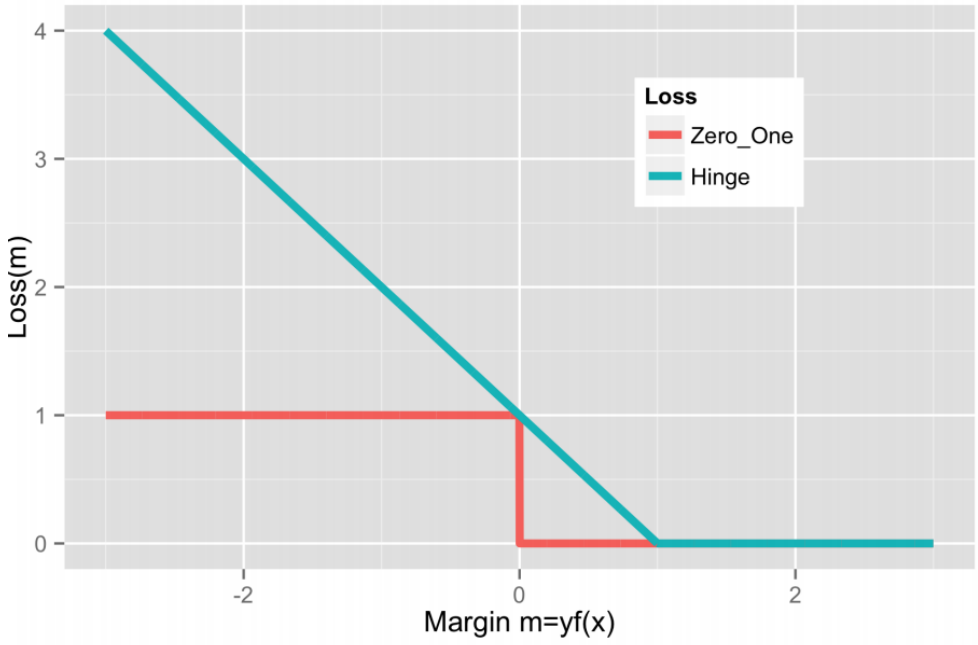

Figure 8: Hinge loss function vs 0-1 loss function(credit: wikipedia)