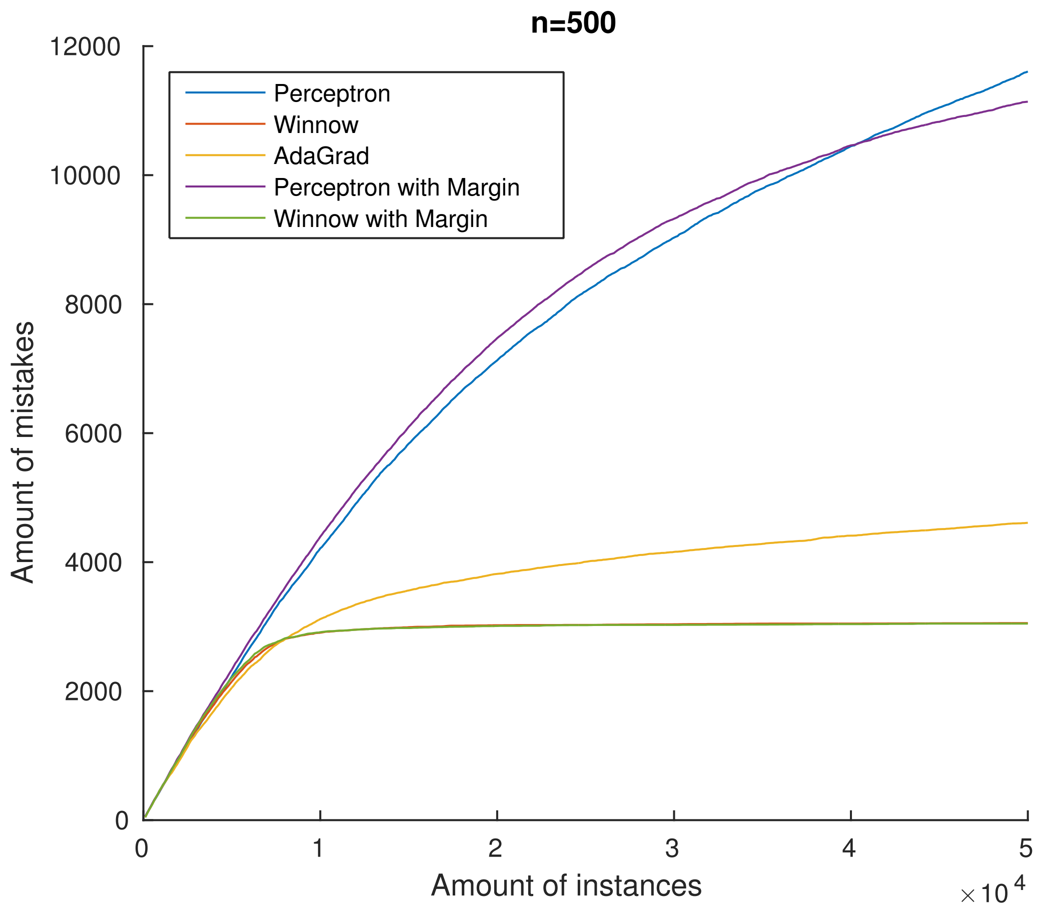
CS446: Machine Learning **Fall 2015**

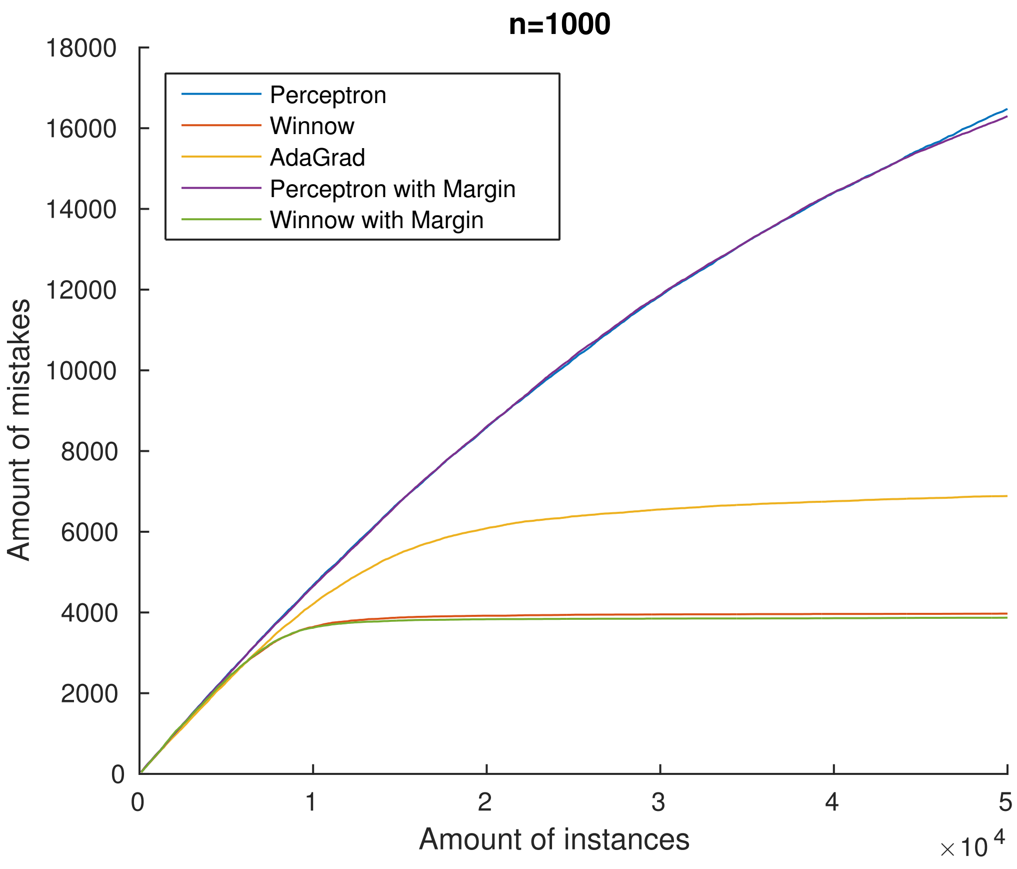
Problem Set 3

Dominik Durner *Handed In: October 8, 2015*

Question 1

Algorithm	Parameters	Dataset n=500	Dataset n=1000
Perceptron	-	-	-
Perceptron w/margin	η	$\eta = 0.005$	$\eta = 0.005$
Winnow	α	$\alpha = 1.1$	$\alpha = 1.1$
Winnow w/margin	α, γ	$\alpha = 1.1, \gamma = 2$	$\alpha = 1.1, \gamma = 2$
AdaGrad	η	$\eta = 0.25$	$\eta = 0.25$



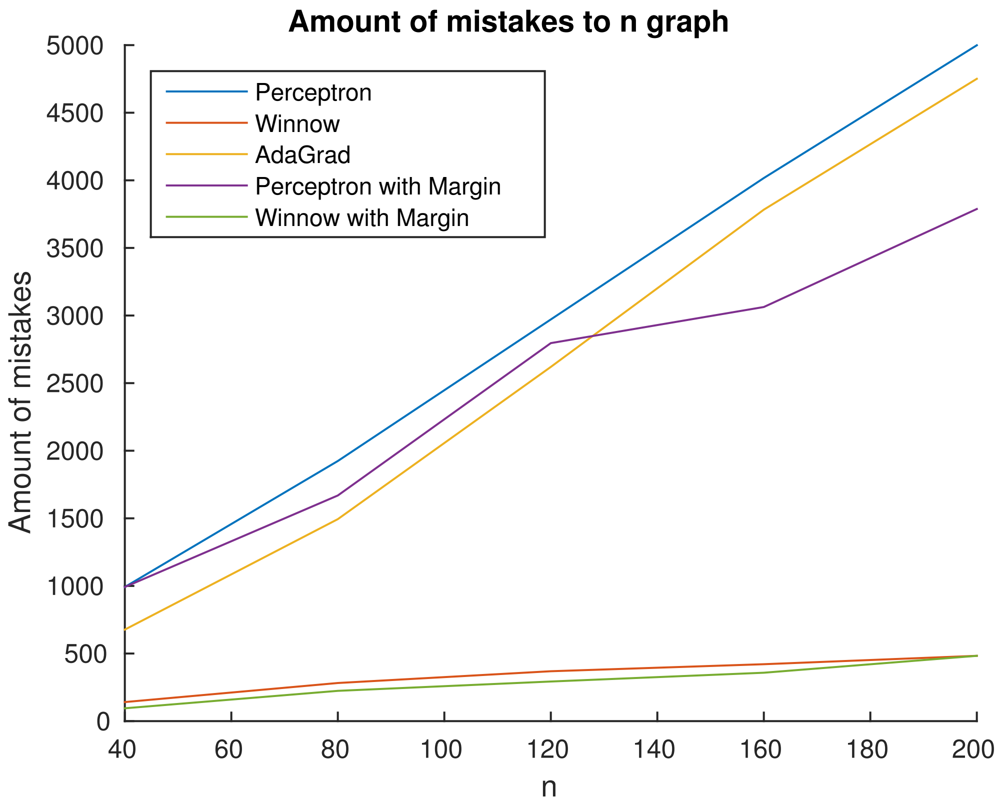


We can see that the Winnow based algorithms converging to the right weight vector very quickly. In the second experiments the Winnow with margin based algorithm is a little better than the standard one. The AdaGrad algorithm is in between Perceptron and Winnow. The small amount of mistakes for Winnow is a result of its properties to only need logarithmic many mistakes for a k out of n problem. This function can only be solved with linear amount of mistakes from the Perceptron based systems. Therefore, our results strengthen the results found in the lecture. The number of the instance vector plays also an important role for the amount of mistakes. Graph 1 does have a lower number of mistakes in all algorithms compared to the second Graph. On the other hand the parameters for the algorithms didn't differ between those two test settings.

Question 2

Algorithm	Parameters	n=40	n=80
Perceptron	-	-	-
Perceptron w/margin	η	$\eta = 1.5$	$\eta = 0.03$
Winnow	α	$\alpha = 1.1$	$\alpha = 1.1$
Winnow w/margin	α, γ	$\alpha = 1.1, \gamma = 2$	$\alpha = 1.1, \gamma = 2$
AdaGrad	η	$\eta = 1.5$	$\eta = 1.5$

Algorithm	n=120	n=160	n=200
Perceptron	-	-	-
Perceptron w/margin	$\eta = 0.25$	$\eta = 0.03$	$\eta = 0.03$
Winnow	$\alpha = 1.1$	$\alpha = 1.1$	$\alpha = 1.1$
Winnow w/margin	$\alpha = 1.1, \gamma = 2$	$\alpha = 1.1, \gamma = 2$	$\alpha = 1.1, \gamma = 2$
AdaGrad	$\eta = 1.5$	$\eta = 1.5$	$\eta = 1.5$



Again, we see that the Winnow based algorithms are quickly converging and only do minimal mistakes during update. Hereby, the Winnow with margin performs a little better than the plain Winnow. Especially with increasing variable space the other algorithm make way more mistakes until they converge. The Perceptron with margin is performing better than the normal Perceptron since it updates also close to the border values and converges therefore faster. One interesting finding is that AdaGrad is not good in finding a a weight vector that is stable for a long set of data (even though it is already quite good). Therefore, it performs worse than the Perceptron with margin algorithm in this experiment.

Question 3

Algorithm	Parameters	Accuracy
Perceptron	-	0.9717
Perceptron w/margin	$\eta = 0.005$	0.9853
Winnow	$\alpha = 1.1$	0.9610
Winnow w/margin	$\alpha = 1.1, \gamma = 0.04$	0.9638
AdaGrad	$\eta = 0.25$	0.9985

Figure 1: Results with m=100.

Algorithm	Parameters	Accuracy
Perceptron	-	0.7214
Perceptron w/margin	$\eta = 0.03$	0.9380
Winnow	$\alpha = 1.1$	0.8140
Winnow w/margin	$\alpha = 1.1, \gamma = 0.001$	0.8140
AdaGrad	$\eta = 0.25$	0.9366

Figure 2: Results with m=500.

Algorithm	Parameters	Accuracy
Perceptron	-	0.6824
Perceptron w/margin	$\eta = 0.03$	0.8012
Winnow	$\alpha = 1.1$	0.7707
Winnow w/margin	$\alpha = 1.1, \gamma = 0.3$	0.7624
AdaGrad	$\eta = 1.5$	0.8313

Figure 3: Results with m=1000.

The best algorithm is now AdaGrad since the accuracy for all m's is quite high. The only other really good performing algorithm is the Perceptron with margin. For this task the Winnow algorithms are performing worse than before on clean data. This is due to the flipped labels which updates the weight vector and θ more drastically in the wrong direction as for other algorithms. Interestingly, the α (which is the important factor for the strength of updates) for example does not change between different m values nor between the clean and dirty data. Furthermore the η values of AdaGrad and Perceptron are also quite robust regarding dirty and clean data. A general trend is that for noisy data, the sparsity/density of the training function is really important. So with increasing m the accuracy drops drastically for all algorithms. This is expected since all variables get more important. Therefore the combined weight is split across many variables. Hence, misclassified train data decreases the already relatively smaller values even further. AdaGrad and Perceptron with margin increase their η, the more dense the function is. This helps those algorithm to still be able to perform strong enough updates to get a pretty good function.

Bonus Question

First, I started to experiment with the perceptron with margin algorithm. I introduced an additional parameter which indicates how strong the learning rate should be pushed on positive examples in comparison to negative ones. Unfortunately, the parameter was most of the times not successful. Therefore, I recalled the sentence that γ and learning rate are closely related to each other. Therefore, I decided to play with the γ of perceptron and fix the learning rate. For the learning rate I use $\eta = 0.03$ since in previous results it was the most prominent one. The γ values I tested are from the winnow test description. As mentioned I also try here to push positive values. Therefore, I just increase the width of the margin in which positive values are affected for updates. Therefore, even tho the function would be already right, some positive values are used to further tweak the weight vector. Because I increase the thickness for positive ones, I do more updates on positive values than on negative ones. I considered those values for positive pushing: $posPush = [10, 9, 8, 7, 6, 5]$. In the end I increased the average Perceptron accuracy from 0.9945 to 0.9975 with the Perceptron for unbalanced data. As a side note, the plain Perceptron is already working quite well on the given examples and therefore it might be already fine for most of the applications.

Algorithm	Parameters	Accuracy
Perceptron	-	0.9904
Perceptron for unbalanced	$\eta = 0.03, \gamma = 0.006, posRate = 9$	0.9983

Figure 4: Results with m=100.

Algorithm	Parameters	Accuracy
Perceptron	-	0.9951
Perceptron for unbalanced	$\eta = 0.03, \gamma = 0.001, posRate = 10$	0.9951

Figure 5: Results with m=500.

Algorithm	Parameters	Accuracy
Perceptron	-	0.9998
Perceptron for unbalanced	$\eta = 0.03, \gamma = 0.04, posRate = 6$	0.9999

Figure 6: Results with m=1000.