CS446: Machine Learning **Fall 2014**

Problem Set 3

Chaitanya Datye *Handed In: October 10, 2014*

1. `Problem 1 Solution`

(a), (b) ***Tuned parameters for the algorithms***
The following results were obtained on running the experiments on the given configurations of the data.

Algorithm	Parameters	Data Set	
		$n = 500$	$n = 1000$
Perceptron w/margin	η	0.03	0.05
Winnow	α	1.1	1.1
Winnow w/margin	α	1.1	1.1
	γ	2	2
AdaGrad	η	0.25	0.25

(c), (d) ***Plots of the mistakes vs number of examples for all algorithms***
The following plots give the cumulative number of mistakes plotted against the total number of examples.

Figure1: Plot of mistakes when $l = 10,\ m = 100,\ n = 500$

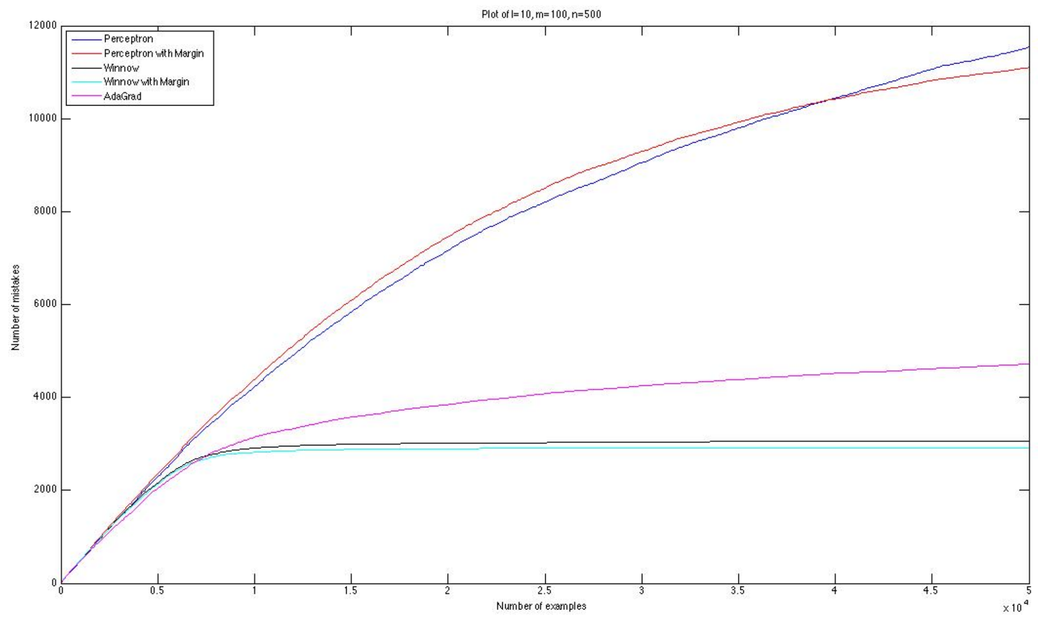
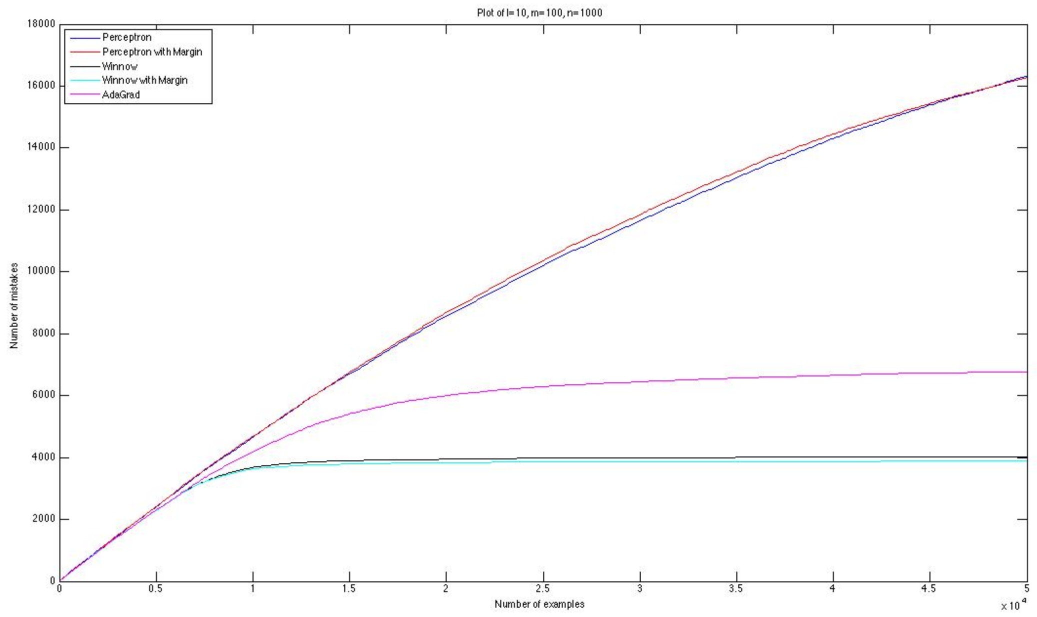


Figure2: Plot of mistakes when $l = 10,\ m = 100,\ n = 1000$



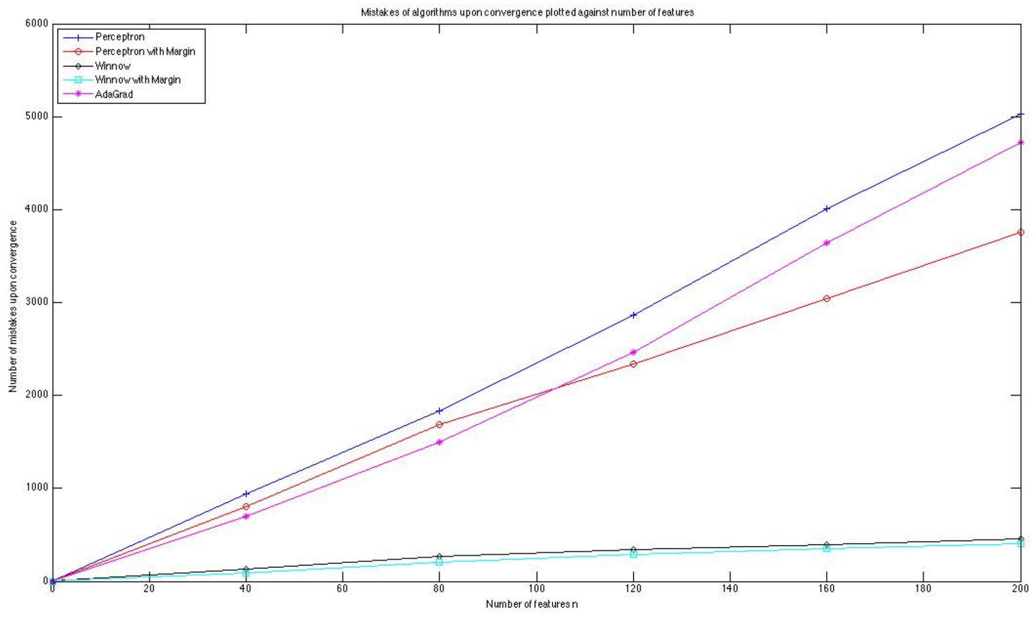
2. `Problem 2 Solution`

Following were the best parameters obtained after the algorithms were tuned.

Algorithm	Parameters	Data Set				
		$n = 40$	$n = 80$	$n = 120$	$n = 160$	$n = 200$
Perceptron w/margin	η	0.25	0.25	0.03	0.03	0.03
Winnow	α	1.1	1.1	1.1	1.1	1.1
Winnow w/margin	α	1.1	1.1	1.1	1.1	1.1
	γ	2	2	2	2	2
AdaGrad	η	1.5	1.5	1.5	1.5	1.5

Once, these best parameters, were obtained, each algorithm was again trained on the set of the training examples. But, this time the algorithm was trained only till we get a sequence of $R = 1000$ examples such that no mistakes were made in these 1000 examples. At this point, the number of mistakes were recorded and were plotted against the different values of n. Following plot depicts the curves for the mistakes made by all the algorithms versus the value of n.

Figure3: Number of mistakes made by each algorithm when algorithm stopped after encountering a mistake-less stretch of $R = 1000$ examples plotted versus n



Explanation —

i. We know that as the number of features n increases, the learning problem becomes harder. We can see from the plot that, as the number of features go on increasing, for each algorithm we get an increasing curve. Since the learning problem becomes harder, the algorithms take more time to converge and thus end up making more mistakes. Hence, as we increase the features n, the curves we see are increasing curves.

ii. **Perceptron**
We can see from the graph that the curve for Perceptron is higher among all the other curves. Since perceptron is a slow learner among all the algorithms, it makes more number of mistakes before convergence and the curve is higher than the remaining curves. For perceptron, since we have the following update rule,
$w_{t+1} = w_t + 1$ ——— Mistake on a positive example
$w_{t+1} = w_t - 1$ ——— Mistake on a negative example
the weight vector always moves closer (by 1) to the value that we wanted.

iii. **Perceptron with margin**
This is better than the normal Perceptron. Since we specify a margin $\gamma = 1$, we reduce the mistake bound of this over the normal perceptron. We also know that Perceptron with margin has a mistake bound of $\frac{R^2}{\gamma^2}$ where γ is the

smallest distance of an example close to the separator. Since we specify a margin, sometimes the weight vector is updated even if there is no mistake and the algorithm thus learns faster than the one without the margin. Specifying a good margin will also ensure that there are less number of mistakes. Since, the updates on the weight vector make sure that a specific distance is always maintained between the separator and the examples, some examples which would have been very close to the separator are automatically taken care of thereby reducing the number of mistakes.

iv. **Winnow**
Unlike Perceptron, Winnow has a logarithmic mistake bound. So, we can see from the graph that the number of mistakes made by Winnow, before it converges are significantly less as compared to the other algorithms. For Winnow, as per the update rule, the new weight vector is a updated by multiplying a factor to the old weight vector. Winnow is a multiplicative algorithm. Hence, the weights for important features are increased at a faster rate and those for unimportant features are decreased at a faster rate. Hence, the algorithm learns very fast and the number of mistakes made are very less. The Winnow algorithm makes $O(k \log n)$ mistakes at max.

v. **Winnow with Margin**
From the graph we can see that there is a slight difference between the curves of Winnow and Winnow with margin. Winnow with margin also has a mistake bound of $O(k \log n)$ but it learns slightly faster than Winnow because of specifying the margin. Since we specify a margin, sometimes the weight vector is updated even if there is no mistake and the algorithm thus learns faster than the one without the margin. Specifying a good margin will also ensure that there are less number of mistakes. Since, the updates on the weight vector make sure that a specific distance is always maintained between the separator and the examples, some examples which would have been very close to the separator are automatically taken care of thereby reducing the number of mistakes.

vi. **AdaGrad**
In case of AdaGrad, unlike Winnow, the updates do not take place by a factor of the previous weight. AdaGrad is an additive algorithm. Here, the learning happens slowly from the frequent features but AdaGrad pays attention to important features. This algorithm updates the weight vector at a faster rate than Perceptron. Hence, the algorithm learns a bit faster than the normal perceptron but slower than winnow. AdaGrad thus makes less mistakes than perceptron but more as compared to Winnow. We thus observe the curve for AdaGrad between that of Perceptron and Winnow. However, the curve is above the perceptron with margin for this particular dataset.

vii. **Rate of Convergence**
We also observed that Winnow and Winnow with margin always converge in the first cycle over the training data for all values of n. So, we can see that Winnow has a faster rate of convergence over the other algorithms. For AdaGrad, Perceptron and Perceptron with margin, for smaller values of n where $n = 40,\ n = 80,\ n = 120$, the algorithms converge in the first cycle but sometime it takes more than one cycle for them to converge when $n = 160$ *or* $n = 200$.

3. `Problem 3 Solution`

Following were the best parameters obtained when the algorithms were tuned on the noisy training data and were tested on the clean data. Also, the table shows the accuracy values of the algorithms over the test data for all the three configurations viz. $n = 100,\ n = 500,\ n = 1000$.

Algorithm	Parm & Accy	Data Set		
		$m = 100$	$m = 500$	$m = 1000$
Perceptron	Accy %	96.82	92.64	82.66
Perceptron w/margin	η	0.03	0.25	0.25
	Accy%	95.32	86.38	79.92
Winnow	α	1.1	1.1	1.1
	Accy%	95.82	93.48	75.34
Winnow w/margin	α	1.1	1.1	1.1
	γ	2	0.3	0.04
	Accy%	95.92	93.92	74.39
AdaGrad	η	0.25	1.5	1.5
	Accy%	99.98	99.42	83.08

Observations –

i. In this experiment, AdaGrad performs better in all cases over all the other algorithms. Hence, AdaGrad is the best performer here. This is different from the best performer in Experiment 2 which was Winnow with margin.

ii. We see that as m increases, the probability of noise in the data increases. With increase in noise, all the algorithms perform bad. Winnow seems to perform worse because it is a multiplicative algorithm and will update the weight vector unnecessarily for some of the weights. However, AdaGrad seems to be more robust to noisy data as compared to the other algorithms and performs better than all the others.

iii. We also see that as the number of features n increase, the accuracy for all algorithms decreases. This is because as the number of features increases, the learning problem becomes harder and the algorithms tend to make more mistakes.

iv. We see that as n increases, the accuracy values for both Winnow algorithms is less than that of the Perceptrons. This is because, as we increase the number of relevant features m, the multiplicative algorithms perform bad. Since the multiplicative algorithms have a mistake bound of $O(m \log n)$ where m is the number of relevant features, we see that as m increases the mistake bound increases and the algorithms make more mistakes thereby reducing the accuracy. For Perceptrons as well the accuracy degrades with the value of m but it stays higher than the Winnows.

v. An interesting observation to note was that for Winnow, the values of best α were always $\alpha = 1.1$ in all the cases. This was seen to be consistent with the values of best α obtained for experiment 2. Also, we see for AdaGrad, that the best value of η obtained was $\eta = 1.5$ in most of the runs and this was seen to be consistent with the values of best η obtained in experiment 2.

4. `Bonus Solution`

For the problem with imbalanced data, we use Perceptron with margin, but instead of using the same margin γ for positive as well as negative classes, we use different margins for γ_+ for positive class and γ_- for negative class. Also, we will let the class having smaller number or training examples have a larger margin since we want to be more careful about the minority class while training. Typically we define a factor M which is the ratio of the number of positive examples to the number of negative examples and then we define the relation between two γs as $\gamma_- = M \times \gamma_+$.

If we make a mistake on the positive example, we update the weights only if the condition $(w^T x + \theta) \leq \gamma_+$ holds true. Similarly, if we make a mistake on a negative example, we update the weights only if the condition $(w^T x + \theta) \leq \gamma_-$ holds true. We then count the number of mistakes the algorithm makes on the positive examples and the number of mistakes it makes on the negative examples. We define three quantities here. $AccPos$, $AccNeg$ and $AccTot$ which represent the accuracy of algorithm over the positive, negative and total number of examples.

We thus use the following update rule:

```
        if(y(i) == 1)
                if(y(i)*label_pred <= gamma_pos)
                        weights = weights + rate*y(i)*transpose(x(
                            i,:));
                        theta = theta + rate*y(i);
                end
            else
                if(y(i)*label_pred <= gamma_neg)
                        weights = weights + rate*y(i)*transpose(x(
                            i,:));
                        theta = theta + rate*y(i);
                end
```

After training the algorithm, we count these accuracies and report these in the following table. We compare these values of modified perceptron with that of the actual perceptron.

Following is the table for this experiment

Algorithm	Parm & Accy	Data Set		
		$m = 100$	$m = 500$	$m = 1000$
Perceptron w/margin Normal	η	1.5	1.5	0.25
	AccyPos%	93.3	94.7	100
	AccyNeg%	100	100	100
	AccyTot%	99.3	99.47	100
Perceptron w/margin Modified	η	0.005	0.25	0.25
	AccyPos%	100	98.5	100
	AccyNeg%	99.91	99.98	100
	AccyTot%	99.92	99.84	100

We can see that in all cases all the accuracy values $AccPos$, $AccNeg\ and\ AccTot$ for the modified perceptron are better than the actual perceptron. In this case, the value of γ is chosen to be $\gamma = 1$ for the normal perceptron with margin and for the modified version of perceptron we choose $\gamma_+ = 1\ and\ \gamma_- = \frac{1}{9}$. Since the ratio of positive to negative examples in the dataset is 1 : 9

We can thus clearly see the improvement of the new modified perceptron algorithm over the normal perceptron algorithm in all the cases.

5. `Instructions to run the code`

   Please follow the following instructions for executing the code

   i Please untar/unzip the ***cdatye2-hw3.tar*** file that was submitted. This will give you the pdf of the report as well as the src directory containing the entire source code. The src directory has four subdirectories namely Q1, Q2, Q3 and Bonus.

   ii For running Experiment 1.

      a Change directory to Q1

      b Run the file driver_algorithms.m

   iii For running Experiment 1.

      a Change directory to Q2

      b Run the file obtain_parameters.m

   iv For running Experiment 3.

      a Change directory to Q3

      b Run the file driver_algorithms.m

   ii For running Bonus Experiment.

      a Change directory to Bonus

      b Run the file driver_algorithms.m