CS446: Machine Learning **Spring 2017**

Problem Set 3

Hsiang-Wei Hwang *Handed In: February 27, 2017*

1. Experiment 1

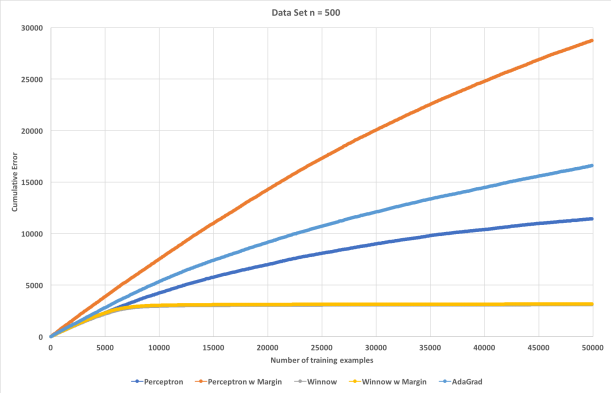
   (a) Data Generating Use attached python code to generate the data set.

```
from __future__ import print_function
from gen import gen
import numpy as np
l = 10
m = 100
n = 500
num = 50000
noise = False
for n in  [500,1000]:
    (y, x) = gen(l, m, n, num, noise)
    strout = ""
    for i in range(y.size):
        f = open("../data/" + str(l)+"_"+str(m)+"_"+str(n)+"_"+str(num)+"_"+str(noise),'w')
        strout = strout+ str(y[i]) + ' ' +' '.join(map(str, x[i]))+ ' '+ "1" + "\n"
    print(strout, file = f)
```

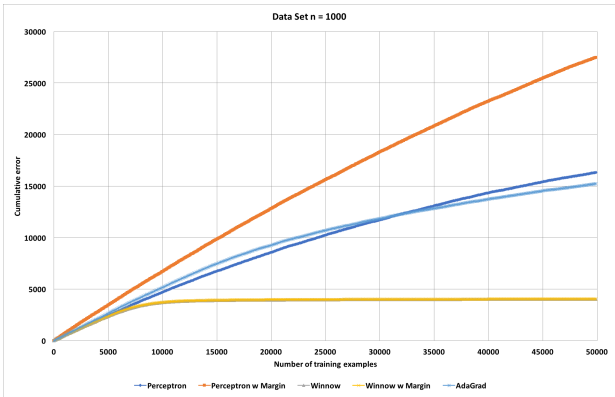
   (b) Parameters Tuning
   1. For Perceptron, $\eta = 1$, $\gamma = 0$. Nothing needs to be swept.
   2. For Perceptron w/margin, choose $\eta \in \{1.5, 0.25, 0.03, 0.005, 0.001\}$, $\gamma = 1$.
   3. For Winnow, choose $\alpha \in \{1.1, 1.01, 1.005, 1.0005, 1.0001\}$, $\gamma = 0$.
   4. For Winnow w/margin, choose $\alpha \in \{1.1, 1.01, 1.005, 1.0005, 1.0001\}$, $\gamma \in \{2.0, 0.3, 0.04, 0.006, 0.001\}$.
   5. For AdaGrad, choose $\eta \in \{1.5, 0.25, 0.03, 0.005, 0.001\}$, $\gamma = 1$.

Algorithm	Dataset n=500	Dataset n=1000
Perceptron	Accu = 0.9718	Accu = 0.9504
Perceptron w/margin	$\eta = 0.005$, Accu = 0.9868	$\eta = 0.03$, Accu = 0.9574
Winnow	$\alpha = 1.1$, Accu = 0.9958	$\alpha = 1.1$, Accu = 0.9954
Winnow w/margin	$\alpha = 1.1$, $\gamma = 2$, Accu = 0.9994	$\alpha = 1.1$, $\gamma = 2$, Accu = 0.997
AdaGrad	$\eta = 0.25$, Accu = 0.9994	$\eta = 0.25$, Accu = 0.997

Then, use the parameters get for the experiment above to gather accumulated error times. Recording the total error times for each 100 training examples. Plot the result for 5 algorithm. X axis means the number of training examples and Y axis means the accumulated error times.



(a) Number of training examples vs Cumulative Error, Dataset n = 500



(b) Number of training examples vs Cumulative Error, Dataset n = 1000

For Perceptron, the increase of γ will take more times to learn the weight vector. For Winnow, the algorithm adjusts the weight vector using exponential coefficient faster than others and has no apparent affect by the n of the dataset. For AdaGrad, it works like Perceptron due to the similarity of the curves. With n = 1000 dataset, performance of AdaGrad is better than Perceptron. That implies AdaGrad is doing better in sparse data.

2. Experiment 2

(a) Data Generating Use attached python code to generate the data set.

```
from __future__ import print_function
from gen import gen
import numpy as np
l = 10
m = 20
n = 40
num = 50000
noise = False
for n in  range(40,201,40):
    (y, x) = gen(l, m, n, num, noise)
    strout = ""
    for i in range(y.size):
        f = open("../data/" + str(l)+"_"+str(m)+"_"+str(n)+"_"+str(num)+"_"+str(noise),'w')
        strout = strout+ str(y[i]) + ' ' +' '.join(map(str, x[i]))+ ' '+ "1" + "\n"
    print(strout, file = f)
```

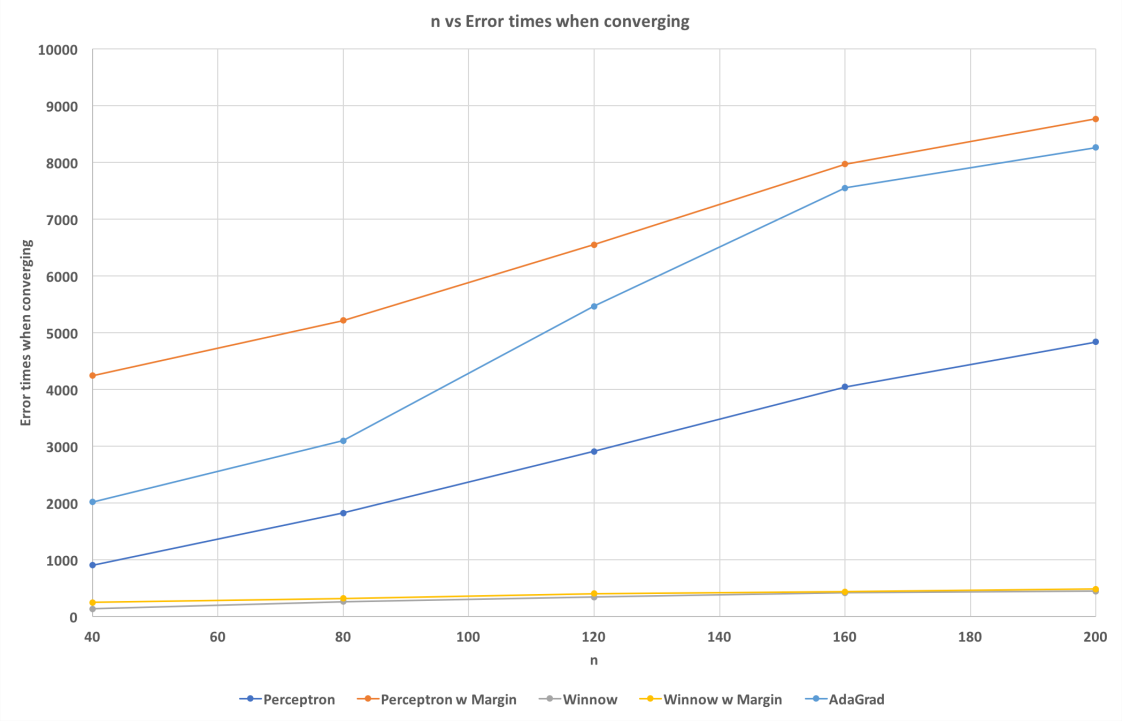
(b) Parameters Tuning

1. For Perceptron, $\eta = 1$, $\gamma = 0$. Nothing needs to be swept.
2. For Perceptron w/margin, choose $\eta \in \{1.5, 0.25, 0.03, 0.005, 0.001\}$, $\gamma = 1$.
3. For Winnow, choose $\alpha \in \{1.1, 1.01, 1.005, 1.0005, 1.0001\}$, $\gamma = 0$.
4. For Winnow w/margin, choose $\alpha \in \{1.1, 1.01, 1.005, 1.0005, 1.0001\}$, $\gamma \in \{2.0, 0.3, 0.04, 0.006, 0.001\}$.
5. For AdaGrad, choose $\eta \in \{1.5, 0.25, 0.03, 0.005, 0.001\}$, $\gamma = 1$.

Algorithm	n=40	n=80	n=120	n=160	n=200
Perceptron	Accu=0.9998	Accu=1	Accu=0.9994	Accu=0.9996	Accu=0.9964
Perceptron w/margin	η=0.03, Accu=1	η=0.03, Accu=1	η=0.03, Accu=1	η=0.03, Accu=0.9998	η=0.03, Accu=0.9974
Winnow	α=1.1, Accu=0.9994	α=1.1, Accu=1	α=1.1, Accu=0.9996	α=1.1, Accu=1	α=1.1, Accu=0.999
Winnow w/margin	α=1.1, $\gamma = 2$, Accu=1	α=1.1, $\gamma = 2$, Accu=1	α=1.1, $\gamma = 2$, Accu=1	α=1.1, $\gamma = 2$, Accu=1	α=1.1, $\gamma = 2$, Accu=1
AdaGrad	η=1.5, Accu=1	η=1.5, Accu=1	η=1.5, Accu=1	η=1.5, Accu=0.9988	η=1.5, Accu=0.9856

(c) Experiment

Choose the parameters we get above, set the iteration limitation to large enough (I set it to 2000), and start to accumulate the number of mistakes happened until no mistake happen in one iteration. Plot the data x axis means n, order of instance space, and y axis means the number of mistakes made until the algorithm converged.



To conclude the experiment, the figure shows that: 1. Winnow is the best algorithm in this experiment. 2. Perceptron with margin needs more mistakes to converge.

3. Experiment 3

(a) Data Generating Use attached python code to generate the data set.

```
from __future__ import print_function
from gen import gen
import numpy as np
l = 10
m = 100
n = 1000
num = 50000;
noise = False
for m in  [100,500,1000]:
    for noise in [True,False]:
        if noise:
            num = 50000
            (y, x) = gen(l, m, n, num, noise)
        else:
            num = 10000
            (y, x) = gen(l, m, n, 10000, noise)
        strout = ""
        for i in range(y.size):
            f = open("../data/" + str(l)+"_"+str(m)+"_"+str(n)+"_"+str(num)+"_"+str(noise),'w')
            strout = strout+ str(y[i]) + ' ' +' '.join(map(str, x[i]))+ ' '+ "1" + "\n"
        print(strout, file = f)
```

(b) Parameters Tuning

1. For Perceptron, $\eta = 1$, $\gamma = 0$. Nothing needs to be swept.
2. For Perceptron w/margin, choose $\eta \in \{1.5, 0.25, 0.03, 0.005, 0.001\}$, $\gamma = 1$.
3. For Winnow, choose $\alpha \in \{1.1, 1.01, 1.005, 1.0005, 1.0001\}$, $\gamma = 0$.
4. For Winnow w/margin, choose $\alpha \in \{1.1, 1.01, 1.005, 1.0005, 1.0001\}$, $\gamma \in \{2.0, 0.3, 0.04, 0.006, 0.001\}$.
5. For AdaGrad, choose $\eta \in \{1.5, 0.25, 0.03, 0.005, 0.001\}$, $\gamma = 1$.

Input 10% of noisy data as training data and another 10% of data as testing data.

Algorithm	m=100		m=500		m=100	
	Accu.	params.	Accu.	params.	Accu.	params.
Perceptron	0.8012	NA	0.5456	NA	0.688	NA
Perceptron w/margin	0.8232	η=0.005	0.6534	η=0.03	0.7174	η=0.03
Winnow	0.8778	α=1.1	0.7934	α=1.1	0.7356	α=1.1
Winnow w/margin	0.8778	α=1.1, γ=0.006	0.7994	α=1.1, γ=0.04	0.7424	α=1.1, γ=0.04
AdaGrad	0.8766	η=0.25	0.7788	η=1.5	0.7352	η=1.5

(c) Experiment
Use the parameters acquired before and input 50000 training noisy examples and test it with 10000 clean test examples.

Algorithm	m=100		m=500		m=1000	
	Accu.	params.	Accu.	params.	Accu.	params.
Perceptron	0.973	NA	0.8114	NA	0.7709	NA
Perceptron w/margin	0.9924	η=0.005	0.9084	η=0.03	0.8308	η=0.03
Winnow	0.9663	α=1.1	0.9366	α=1.1	0.7708	α=1.1
Winnow w/margin	0.9536	α=1.1, γ=0.006	0.9245	α=1.1, γ=0.04	0.7699	α=1.1, γ=0.04
AdaGrad	0.9998	η=0.25	0.991	η=1.5	0.8591	η=1.5

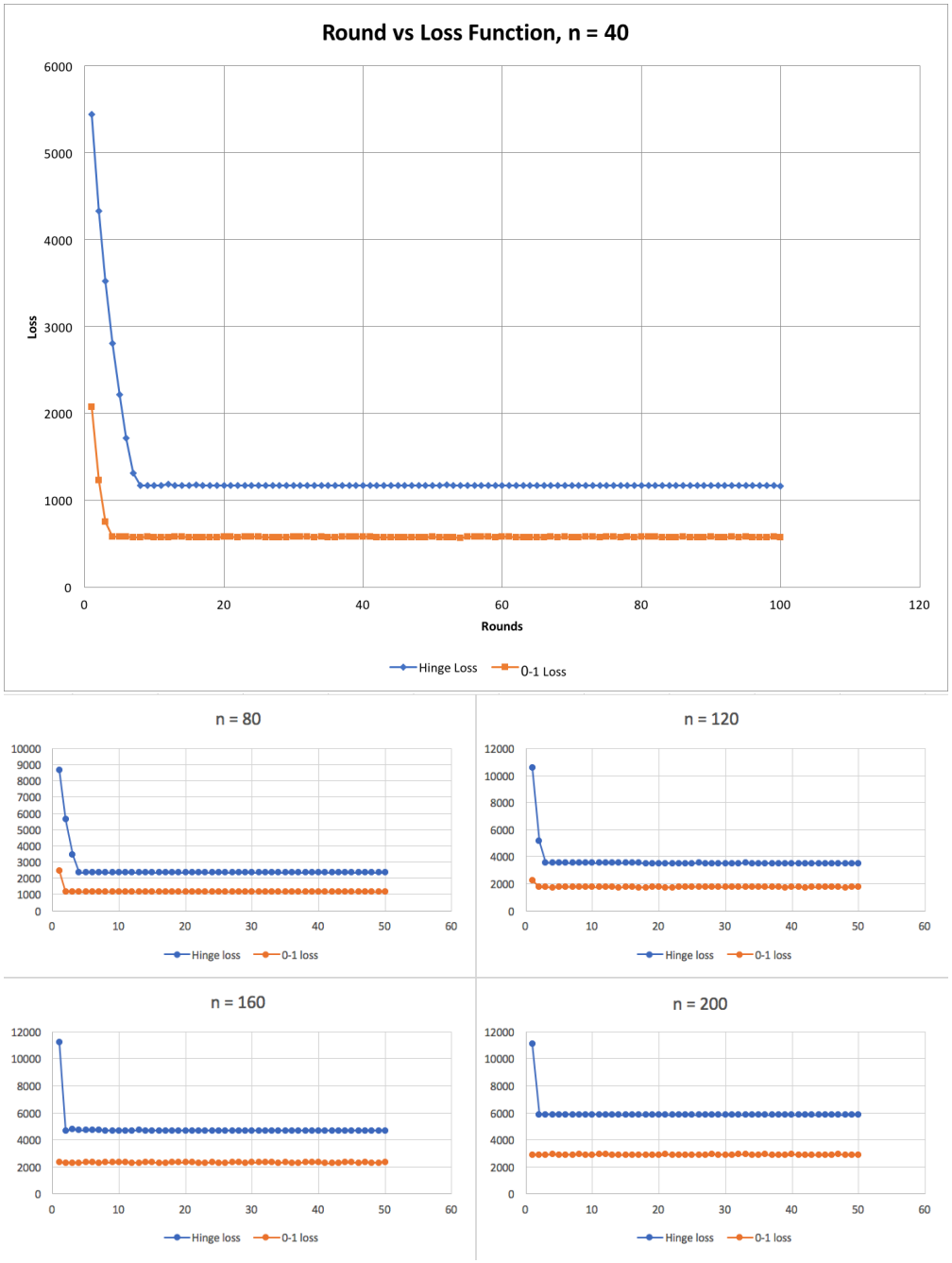
According to the table above, the sparser dataset is the better AdaGrad is comparing to others. Winnow thought to be the best in the past 2 experiments does a bad job in the sparse data.

4. Bonus

(a) Data Generating Use attached python code to generate the data set.

```
from __future__ import print_function
from gen import gen
import numpy as np
l = 10
m = 20
n = 40
num = 10000
noise = True
(y, x) = gen(l, m, n, num, noise)
strout = ""
for n in range(40,201,40):
    for i in range(y.size):
        f = open("../data/" + str(l)+"_"+str(m)+"_"+str(n)+"_"+str(num)+"_"+str(noise),'w')
        strout = strout+ str(y[i]) + ' ' +' '.join(map(str, x[i]))+ ' '+ "1" + "\n"
    print(strout, file = f)
```

(b) Experiment Use the parameter settings from Problem 3, $\eta = 0.25$. Input l =10, m=20, n=40, noisy data as training data. Accumulate the loss after each round of training. For Hinge loss, sum up and record $\max(0, 1 - y(wx + \theta))$ in each round. On the other hand, sum up and record the mistakes made in each round. Then, plot the figure x axis means rounds and y axis means loss.



Observing the figure, we can conclude that after rounds of training process, AdaGrad algorithm converge. The Hinge loss and 0-1 loss are stable at last. It shows that after the algorithm converges the learning process, both 0-1 loss and Hinge loss still show the mistakes happen for training dataset. Comparing two losses, the Hinge loss is larger all the time. It is reasonable that even the classification is correct, $y(wx + \theta) > 0$, the example may still contribute to the Hinge error $\max(0, 1 - y(wx + \theta))$ if $1 - y(wx + \theta) > 0$. Besides, it can be observed that the higher n is the faster the algorithm converge. When n = 200, it converges almost at the first round.