Perceptron

Guo Li
NetID: gli27
CSC 446
Jan 23 2019

Briefly Description

This programming assignment is to implement the simple perceptron algorithm to linear classify the data set we have, adult income data set. Basically, this algorithm is a two-classification model and the data set are also a two-classes set. The main idea is to use features (x) multiply with weights (w) to determine the labels (y). The target is to find an appropriate w set to best classify all data set into two categories. And at the beginning, we start with w = 0 and every time we make a wrong classification, we change the value of w and also, the bias b. The general modification equation is like this:

$$\mathrm{w}^{\tau} \leftarrow \mathrm{w}^{\tau-1} + \eta y_i x_i$$

$$\mathrm{b}^{\tau} \leftarrow \mathrm{b}^{\tau-1} + \eta y_i$$

and in these equation, η means the learning rate. After some iteration times, we perceive that the resulting weights are what we want and implement them to finish the classification work over test data set. Plus, we will have dev data, this can be used to early stop the algorithm and avoid overfitting problem, since we just set some iteration times but do not set an appropriate time to break the algorithm.

Command we use and args we have

The basic command we will use is like this: "python3 Guo_perceptron.py", this does not include any args and will give us the default running result with all args with default values. And here is the explanation of args we might have:

1. '--nodev', which controls whether the algorithm will use the dev data or not. The default value is False, which mean the dev data will be used. And once the --nodev is provided, the value of this arg will be set to True and the dev data will not be used.

2. '--iterations', which means the iteration times, and its default value is 50.

3. '--lr', which means the learning rate we mentioned before and its default value is 1.

4. '--train_file', '--dev_file', '--test_file', these args are the data set path, we don't need to indicate them in command since the path is already given in the program.

5. '--plot', this arg is defined and added by myself and its default value is False. When this arg provided, the program will implement the function to plot and give us the graph of the accuracy-iterations. Running this

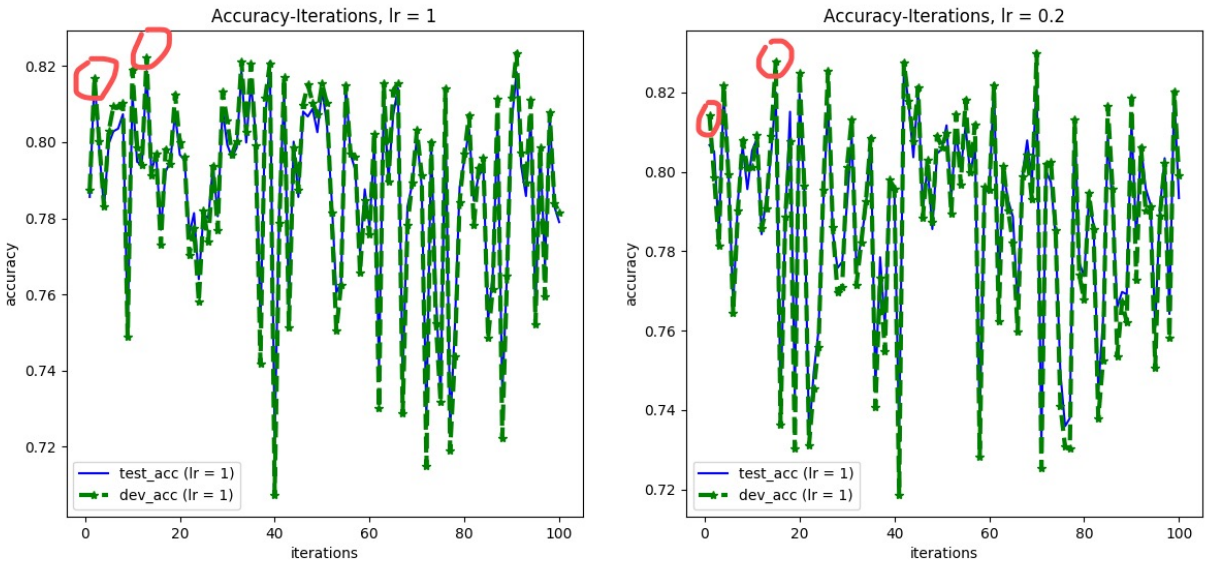
will use a few minutes and all graphs will be inserted and discussed in this report.

6. '--devmode', this is also self-defined and there are two valid value for it, 0 and 1. This arg is used to choose the mode we will implement the dev data set. '0' is the default value and means that we will use the dev data in the way as once we perceive the accuracy of dev data decreased, then we stop and use the weights we calculated. '1' means another mode that we will store some weights until we think the accuracy of dev data is stabled and relatively precise and then we use the best weights over dev data to classify test data set. We will discuss it later for details.

Examples of running command:
"python3 Guo_perceptron.py --nodev --lr 0.5 --iterations 20"
which means not use dev data with learning rate is 0.5, iteration times are 20.
"python3 Guo_perceptron.py --lr 0.2 --iterations 50 --devmode 1"
which means learning rate is 0.2, iteration times are 50, and dev data is implemented and the dev mode is chosen as mode 1 (10 more accuracy result stored and not increased.)

Performance over iteration times

As we stated above, here we provide the accuracy-iterations plot of the result of dev data and test data:



As you can see in the picture. The blue line is the result of the weights over test data set and the green line with '*' is the result of dev data. We can see that, although the iteration times increased to 100, there are no obviously overfitting problem come into our sight. This is because the model we use, perceptron, is a simple linear model and will not have a very strong symbol of overfitting. So, the curve will not be monotonically decreasing (in a big scene, generally). There is no exactly point for the early stop. Also, this cause another interesting thing that we might need to pay some attention to. The basic early stop points we have once we choose the default usage mode of dev data set, which is once the accuracy over dev data decreased, we stop and use current weights as the final result. As you can see the marked points in the graph, the early stop point will come to our eyes very early and even a few more iterations there will be a

better choice for us. This is because the curve of the general increasing part of accuracy over dev data is not monotonical.

And in this case, we choose another mode to implement the dev data set, which will be indicated as '--devmode 1'. This is set as we store 10 accuracy values and their corresponding weights. Once all the next 10 times, the accuracy of the weights all smaller than the one we stored, we will assume that the accuracy is good enough and we will use it as the final one. Note that the value 10 is set by human and this depends on the all iteration times and the decreasing speed of the accuracy curve.

And the result of devmode 1 comparing to devmode 0 is shown as:

For learning rate is 1:

```
[gli27@cycle1 ~]$ python3 Guo_perceptron.py
[-1 -1 -1 ...  1 -1  1]
Test accuracy: 0.7982507977780404
Feature weights (bias last): -6.0 -3.0 3.0 3.0 0.0 2.0 -1.0 7.0 8.0 1.0 1.0 -2.0 0.0 -8.0 5.0 1.0 -2
.0 1.0 -3.0 0.0 -2.0 0.0 0.0 -1.0 2.0 1.0 -5.0 6.0 1.0 0.0 1.0 7.0 -3.0 -7.0 -9.0 0.0 0.0 1.0 5.0 6.
0 -3.0 -7.0 -2.0 -1.0 -5.0 9.0 2.0 1.0 -2.0 5.0 8.0 1.0 0.0 1.0 -1.0 -4.0 -1.0 -1.0 7.0 0.0 5.0 -2.0
 0.0 -2.0 -4.0 0.0 1.0 5.0 -4.0 -2.0 -3.0 -3.0 0.0 -7.0 4.0 -5.0 2.0 -8.0 2.0 0.0 0.0 3.0 4.0 3.0 3.
0 -3.0 8.0 0.0 0.0 -9.0 5.0 -1.0 -1.0 -2.0 7.0 -1.0 0.0 5.0 3.0 1.0 1.0 -5.0 -1.0 -2.0 3.0 0.0 -3.0
-1.0 7.0 0.0 1.0 -6.0 -3.0 3.0 2.0 3.0 -4.0 5.0 2.0 -2.0 -3.0 -2.0 0.0 -3.0
[gli27@cycle1 ~]$ python3 Guo_perceptron.py --devmode 1
[-1 -1 -1 ...  1 -1  1]
Test accuracy: 0.8186975534806761
Feature weights (bias last): -6.0 -3.0 5.0 3.0 -3.0 2.0 0.0 4.0 7.0 2.0 0.0 -6.0 0.0 -8.0 3.0 3.0 -2
.0 0.0 -3.0 0.0 1.0 -1.0 2.0 -2.0 3.0 0.0 -5.0 6.0 1.0 2.0 0.0 5.0 0.0 -13.0 -9.0 -1.0 0.0 1.0 5.0 8
.0 -5.0 -6.0 -4.0 0.0 -5.0 8.0 3.0 0.0 -4.0 4.0 10.0 -1.0 0.0 0.0 -1.0 -3.0 -3.0 -2.0 6.0 0.0 5.0 -2
.0 -1.0 -2.0 -4.0 0.0 1.0 4.0 -4.0 -3.0 -2.0 -4.0 0.0 -8.0 4.0 -6.0 2.0 -5.0 1.0 -4.0 1.0 3.0 5.0 10
.0 2.0 -5.0 8.0 -1.0 0.0 -8.0 4.0 0.0 0.0 -3.0 7.0 -1.0 0.0 4.0 7.0 2.0 2.0 -10.0 0.0 -3.0 9.0 -1.0
-6.0 -4.0 12.0 -3.0 0.0 -9.0 -5.0 2.0 7.0 6.0 -7.0 13.0 2.0 -4.0 -8.0 -4.0 0.0 -4.0
```

For learning rate is 0.2:

```
[gli27@cycle1 ~]$ python3 Guo_perceptron.py --lr 0.2
[-1 -1 -1 ...  1 -1  1]
Test accuracy: 0.8051057794586928
Feature weights (bias last): -1.4 -0.4000000000000001 0.6000000000000001 0.4000000000000001 5.551115
123125783e-17 0.4000000000000001 -0.20000000000000007 0.6000000000000001 1.5999999999999999 0.200000
00000000007 0.4000000000000001 -0.2 0.0 -1.5999999999999999 1.0 0.6000000000000001 -0.600000000000000
01 -0.19999999999999996 -0.6000000000000001 0.19999999999999996 -0.19999999999999996 -5.551115123125
783e-17 0.20000000000000007 -5.551115123125783e-17 0.20000000000000007 -0.2 -1.0 0.4000000000000001
-0.20000000000000007 0.4 5.551115123125783e-17 1.2 -0.20000000000000007 -1.0 -1.7999999999999998 -5.
551115123125783e-17 0.19999999999999996 0.19999999999999996 0.6000000000000001 1.4 -0.600000000000000
01 -1.2 -0.8 -5.551115123125783e-17 -0.6000000000000001 1.0 0.4000000000000001 0.20000000000000007 -
0.20000000000000007 0.4000000000000001 1.5999999999999999 5.551115123125783e-17 0.39999999999999997
-5.551115123125783e-17 0.20000000000000007 -0.8 -0.20000000000000007 -0.4 1.2 0.0 0.6000000000000001
 -0.4000000000000001 -0.20000000000000007 -0.19999999999999996 -0.6000000000000001 5.551115123125783
e-17 0.6000000000000001 0.6000000000000001 -0.8 -0.6000000000000001 -0.6000000000000001 -0.600000000
0000001 -0.20000000000000007 -1.2 0.4000000000000001 -1.0 0.19999999999999996 -0.8 -0.200000000000000
007 -0.4000000000000001 5.551115123125783e-17 0.6000000000000001 1.0 0.4 0.4000000000000001 -0.60000
00000000001 1.2 -0.4000000000000001 0.0 -1.7999999999999998 1.0 -0.2 -5.551115123125783e-17 -0.2 1.0
 -0.4000000000000001 0.0 0.8 0.4000000000000001 0.4000000000000001 0.0 -0.8 -5.551115123125783e-17 -
0.6000000000000001 0.6000000000000001 0.0 -0.6000000000000001 -0.2 1.0 -0.2 0.0 -1.2 -0.4 0.4 0.0 0.
4 -0.4 1.2 0.2 -0.4000000000000001 -0.4 -0.4 0.0 -0.8
```

```
[gli27@cycle1 ~]$ python3 Guo_perceptron.py --lr 0.2 --devmode 1
[-1 -1 -1 ...  1 -1  1]
Test accuracy: 0.8195248788559272
Feature weights (bias last): -1.0 -0.8 0.6000000000000001 0.6000000000000001 5.551115123125783e-17 0
.20000000000000007 -0.20000000000000007 1.2 1.0 0.6000000000000001 0.20000000000000007 -0.4 0.0 -1.4
 0.6000000000000001 0.19999999999999996 -0.4000000000000001 0.4000000000000001 -0.6000000000000001 0
.20000000000000007 0.4000000000000001 -0.20000000000000007 0.20000000000000007 -0.4000000000000001 0
.6000000000000001 -0.20000000000000007 -0.8 0.6000000000000001 -5.551115123125783e-17 0.400000000000
0001 -0.19999999999999996 1.2 -0.20000000000000007 -1.5999999999999999 -1.5999999999999999 -0.200000
00000000007 0.20000000000000007 0.19999999999999996 0.8 1.4 -0.4000000000000001 -1.2 -0.8 -0.2000000
0000000007 -1.0 1.5999999999999999 0.6000000000000001 -0.19999999999999996 -0.4000000000000001 1.0 1
.7999999999999998 5.551115123125783e-17 -0.20000000000000007 0.4000000000000001 -0.19999999999999996
 -0.6000000000000001 -0.39999999999999997 -0.6000000000000001 1.4 0.0 1.0 -0.20000000000000007 -0.60
00000000000001 -0.39999999999999997 -0.20000000000000007 -0.19999999999999996 -0.20000000000000007 0
.8 -0.4000000000000001 -0.4000000000000001 -0.4000000000000001 -0.6000000000000001 -5.55111512312578
3e-17 -1.2 0.6000000000000001 -1.0 0.39999999999999997 -1.0 -5.551115123125783e-17 -5.55111512312578
3e-17 -0.19999999999999996 0.6000000000000001 1.2 0.8 0.6000000000000001 -0.8 1.2 -0.400000000000000
1 0.0 -1.7999999999999998 0.8 0.2 -5.551115123125783e-17 -0.4000000000000001 1.2 -0.2000000000000000
7 0.0 0.8 0.8 0.4000000000000001 5.551115123125783e-17 -1.2 -0.19999999999999996 -0.8 0.8 0.4 -1.2 -
0.4 1.5999999999999999 -0.4000000000000001 0.4 -1.5999999999999999 -0.4000000000000001 0.60000000000
00001 0.4 0.4 -0.4000000000000001 1.5999999999999999 0.4 -0.6000000000000001 -0.8 -0.600000000000000
1 0.0 -0.6000000000000001
```

And you can see that, using this mode can stably increase the accuracy by about 0.02 and this coordinate the expectation we have in the beginning. So, using this mode will just slightly increase the running time and will give us a relatively more confident model for testing and classifying. Since this is a linear model, the increasement might not be that obvious, but this will be better adapted for a nonlinear model.

Notice that here we do not store all result and choose the best one. This is easy to do but will cost a lot running time to finish and there is no point to stop earlier. Besides, this can also be implemented over test result, like we do not choose any model for new prediction but applied all models in the prediction and just present the best one. Then there is no point training the model.

Test for the smoke script

This program can pass the smoke script test smoothly and here is the screenshot of it:

```
[gli27@cycle1 ~]$ python3 perceptron_smoke_test.py
Found Python file to run.
Found data directory.
Running this command:
./Guo_perceptron.py --nodev --iterations 5 --lr 2.0 --train_file /u/cs246/data/adult/a7a.train --tes
t_file /u/cs246/data/adult/a7a.test
Ran Python file.
Test accuracy is correct!
Feature weights are correct!
Congratulations, you passed this simple test! However, make sure that your code runs on the csug ser
ver. Also, don't forget your README!
```