2022CCUML_proj1

408410018 資工三 簡新哲

Execution description

1. `python3 make_samples.py` 得到positive_samples 和 negative_samples 各15個。(1)
2. `python3 pla.py` 得到 pla 對 50 個 samples 的圖表結果。
3. `python3 pocket.py` 得到 pocket 演算法對 50 個 samples 的圖表結果。
4. `python3 pla_average.py` 得到對 50 個 samples 做三次 PLA 所得到的平均 iterations 以及各自的圖表。(2)
5. `python3 pla_vs_pocket.py` 得到 pla 與 pocket 演算法所得到的iterations 以及各自的圖表。(3)

Experimental results

Q1. Discuss if your PLA halts or how many iterations it halts.(2)

我的 pla 演算法可以停止，經過 10 次測試得到結果如下：

```
hsinz@hsinz-VirtualBox:~/Documents/CCUML/408410018_proj1_v1$ python3 pla.py
Iteration =  6913
hsinz@hsinz-VirtualBox:~/Documents/CCUML/408410018_proj1_v1$ python3 pla.py
Iteration =  25388
hsinz@hsinz-VirtualBox:~/Documents/CCUML/408410018_proj1_v1$ python3 pla.py
Iteration =  112
hsinz@hsinz-VirtualBox:~/Documents/CCUML/408410018_proj1_v1$ python3 pla.py
Iteration =  4550
hsinz@hsinz-VirtualBox:~/Documents/CCUML/408410018_proj1_v1$ python3 pla.py
Iteration =  423
hsinz@hsinz-VirtualBox:~/Documents/CCUML/408410018_proj1_v1$ python3 pla.py
Iteration =  157
hsinz@hsinz-VirtualBox:~/Documents/CCUML/408410018_proj1_v1$ python3 pla.py
Iteration =  27
hsinz@hsinz-VirtualBox:~/Documents/CCUML/408410018_proj1_v1$ python3 pla.py
Iteration =  114
hsinz@hsinz-VirtualBox:~/Documents/CCUML/408410018_proj1_v1$ python3 pla.py
Iteration =  11
hsinz@hsinz-VirtualBox:~/Documents/CCUML/408410018_proj1_v1$ python3 pla.py
Iteration =  5147
hsinz@hsinz-VirtualBox:~/Documents/CCUML/408410018_proj1_v1$
```

測試中最大的iteration為25388、最小為11

Q2. Implement Pocket Algorithm and compare the execution time to PLA on the same dataset.(3)

設計 threshold = 500, 進行 5 次測試結果如下：

	Task 1	Task 2	Task 3	Task 4	Task 5	Average
PLA	140	249	180	759	43	274.2
Runtime of PLA	3.3038	6.8175	4.8717	16.2264	1.0063	6.44514
Pocket	140	249	180	overflow(500)	43	222.4
Runtime of pocket	6.4970	14.0038	8.4509	22.4037	1.8904	10.64916

得到平均的 runtime 大約差了一倍。

Q3. Report the accuracy of Pocket Algorithm by this setting and that of Problem 3.(4)

調整 threshold 並進行 5 次測試：

	Task 1	Task 2	Task 3	Task 4	Task 5	Average
300	0.57	0.81	0.75	0.60	0.68	0.682
500	0.88	0.66	0.74	0.58	0.91	0.754
700	0.74	0.59	0.89	0.91	0.94	0.814

Conclusion

Q1.

Iteration相對低的情況，如 47 、 64 等等，是屬於在所有產生的 samples 中與理想的 hyperplane 距離最近的正負 sample 。此時兩者的距離夠大，使在更新 hyperplane 時修正的誤差值較小。

而Iteration相對高的形況，如 7849 、 20064 ，則是其正負 sample 的距離過小，使得在更新 hyperplane 時誤差值較大，容易造成 hyperplane 的位置更新在線段左右，無法獲得適當的值。

Q2.

執行時將預設的 threshold = 500。結果上來看 pocket algorithm 在沒有 overflow 的時候，執行時間大約為 PLA 方法的兩倍。而 overflow 的話大約為 PLA 方法的 1.2 倍。

推測結論是在 pocket algorithm 中，如果沒有 overflow ，則會判斷是否更改 w 後會得到更高的 accurancy ，因此會造成大約是 PLA 執行時間一倍的額外延長時間；如果 overflow ，則不會執行該程式碼，僅延長大約是 PLA 執行時間 0.2 倍的額外延長時間。

Q3.

排除 sample 分布的隨機情況，可以發現如果將 threshold 調整的愈高，就會獲得愈高的 accurancy 。

Discussion

前面進行 PLA 程式設計與執行的時候比較沒有遇到什麼問題，但在進行 assignment3 的 `pla_vs_assignment.py` 時常常會無法結束程式，經過逐條排解之後發現是在 `pla.py` 程式中的 while check_error(w, dataset) is not None: 與 if int(np.sign(w.dot(x))) != label: 兩條程式碼卡住，但重複測試多次之後仍然有成功跟不成功的結果，因此可能是在對於 w 的更新狀態有問題，與函式 dot(x) 的不熟悉。

然而將 `pla.py` 的兩條函式直接放在 `pla_vs_pocket.py` 中時，結果就自然解決了，我無法理解這是什麼問題，猜測可能是在跨檔案呼叫函式的過程造成太多的垃圾時間堆積。

也因為這個莫名其妙的 bug 讓我遲交了一天，因為 de 出來了所以重交作業。

感謝老師提供 server ，讓我在處理更多的 threshold 的過程中減少了等待時間，大約在 threshold = 3000 的時候速度會慢得讓我不開心。