Practice #1 : Binary Classifier

2018008395 박정호

1. Experiment

실험 결과 설명에 앞서, 몇 가지 사항을 짚고 넘어가겠다.

기본적으로, 각 실험에 쓰이는 데이터 셋은 매번 새로 생성했다. 이는 실험 결과가 데이터 셋마다 다를 수 있기에 결정한 사항이다.

또한, 데이터 셋은 기본적으로 numpy array로 생성했으나, element-wise implementation에서는 numpy array의 broadcasting과 벡터 연산을 일체 사용하지 않고, 각 element에 직접 접근하여 계산하는 방식으로 구현했다. 데이터의 물리적 구조를 통일한 것은 python list와 numpy array의 내부 구조 차이로 인한 성능 차이를 배제하고, 순수하게 알고리즘의 성능 차이를 확인하기 위함이다.

마지막으로 두 알고리즘의 input이 되는 W, b의 초기값은 동일하게 설정했다. 이 또한 순수하게 알고리즘의 성능 차이를 확인하기 위해서이다.

1. Time Comparison

```
PS C:\Users\pch68\HYU_ITE4053\practice #1> python .\logistic_regression.py time 1000 100 2000
Training with Element-wise Implementation...
----------------RESULT----------------
Running Time = 26.923014 sec

Training with Vector-wise Implementation...
----------------RESULT----------------
Running Time = 0.327124 sec
```

element-wise implementation의 vectorized implementation에 비해 대략 100배 정도 느린 것을 확인할 수 있다. 이는 아주 큰 차이이며, 이를 통해서 vectorized implementation이 월등하게 성능이 좋다는 것을 알 수 있다.

2. Estimated Function Parameters

```
PS C:\Users\pch68\HYU_ITE4053\practice #1> python .\logistic_regression.py parameter 1000 100 2000
Training with Element-wise Implementation...
----------------RESULT----------------
Estimated W =  [1.37064231 1.39436428]
Estimated B =  -0.1167124863216776

Training with Vector-wise Implementation...
----------------RESULT----------------
Estimated W =  [1.37064231 1.39436428]
Estimated B =  -0.11671248632167767
```

당연하게도 두 구현 모두 같은 W, b를 도출해낸 것을 볼 수 있다. 물론 실수형 자료형의 오차 문제로 인해 미미한 차이는 존재한다.

3. Empirically Determined the Best Alpha Value

```
PS C:\Users\pch68\HYU_ITE4053\practice #1> python .\logistic_regression.py alpha 1000 100 2000
Search the Best Alpha Range...
Search the Best Alpha Value...
----------------RESULT----------------
Best Alpha :  0.2
PS C:\Users\pch68\HYU_ITE4053\practice #1> python .\logistic_regression.py alpha 1000 100 2000
Search the Best Alpha Range...
Search the Best Alpha Value...
----------------RESULT----------------
Best Alpha :  0.30000000000000004
PS C:\Users\pch68\HYU_ITE4053\practice #1> python .\logistic_regression.py alpha 1000 100 2000
Search the Best Alpha Range...
Search the Best Alpha Value...
----------------RESULT----------------
Best Alpha :  0.1
PS C:\Users\pch68\HYU_ITE4053\practice #1> python .\logistic_regression.py alpha 1000 100 2000
Search the Best Alpha Range...
Search the Best Alpha Value...
----------------RESULT----------------
Best Alpha :  0.2
PS C:\Users\pch68\HYU_ITE4053\practice #1> python .\logistic_regression.py alpha 1000 100 2000
Search the Best Alpha Range...
Search the Best Alpha Value...
----------------RESULT----------------
Best Alpha :  0.1
```

가장 좋은 alpha 값을 찾기 위해 10^{-1}부터 10^{-5}까지의 10의 음의 거듭제곱을 alpha로 해서 accuracy를 비교한 후, 가장 accuracy가 높은 alpha값의 대략적인 크기 S를 찾았다. 그리고 이 S에 1~9까지를 곱한 값을 다시 alpha로 두어서 가장 좋은 accuracy를 보이는 값을 찾았다.

여러 번 실험을 거듭한 결과, 생성된 데이터 셋에 따라 최적의 값이 꽤 크게 달라지는 것을 확인할 수 있었다. 다만, 그 범위는 대략 10^{-1} * N의 비교적 큰 값인 것을 확인할 수 있었다. 즉 이 데이터 셋에 대해서는 비교적 큰 alpha 가 더 학습에 도움이 되는 것을 알 수 있었다.

4. Accuracy

	m = 10 n = 100 k = 2000	m = 100 n = 100 k = 2000	m = 1000 n = 100 k = 2000
Train Set Accuracy	100%	100%	99.5%
Test Set Accuracy	85%	98%	99%

	m = 1000 n = 100 k = 20	m = 1000 n = 100 k = 200	m = 1000 n = 100 k = 2000
Train Set Accuracy	81.1%	95.9%	98.8%
Test Set Accuracy	82%	98%	98%

2. Discussion

보고서에서는 언급되지 않았으나, iteration이 반복되면 반복될수록 정확도와 cost의 절댓값이 줄어드는 것을 볼 수 있었다. 다만, 정확도의 급격한 상승이 있는 것은 초반 100회 정도에서 그쳤으며, 이후에는 그리 큰 정확도 상승은 볼 수 없었다. 이는 데이터 셋의 크기, 데이터의 범위 등이 이 실험에서는 상당히 작았기 때문일 것이다.

또한, element-wise implementation은 상상 이상으로 느리다는 것을 알 수 있었다. 물론 필자가 파악하지 못한 구현 상의 naïve한 부분이 있을 수도 있지만, 그걸 고려하더라도 상당히 큰 성능 차이였다. 이는 아무래도 pre-compile 되어있는 numpy 라이브러리의 공이 컸을 것이라 생각한다.

학습의 성능 점검의 기준에 대해서도 고민하게 되었다. Best alpha 값을 구하기 위해서는 어떤 것이 좋은 학습 결과인가에 대한 기준이 필요했는데, training set에 대한 정확도, test set에 대한 정확도, cost, running time 등 꽤 다양한 요소를 고려하게 되었다. 다만, running time은 iteration 횟수가 정해져 있는 이번 실험에서는 배제되었고, 직접 학습한 데이터에 대해서는 classify가 제대로 되는지가 중요하다고 생각하여 training set에 대한 정확도를 기준으로 했다. 만약 다른 기준, 혹은 다른 방법으로 alpha 값을 찾았더라면 어느 정도 통일된 값을 얻었을지도 모르겠다.

마지막으로, 좁은 범위의 데이터 셋이지만, 실험을 반복할 때마다 그 결과값이 어느정도 차이를 보이는 것을 볼 수 있었다. 이는 아무래도 랜덤 기반 데이터 셋이기 때문일 것이며, 그로 인해 데이터 셋의 형태가 매 실험마다 크게 달라지기 때문일 것이다. 다만, iteration 횟수나, 데이터 셋의 크기 등을 크게 하는 것으로 이러한 차이는 쉽게 줄일 수 있었으며, 따라서 충분한 양의 데이터를 확보하는 것이 중요하다는 것을 알 수 있었다.