CS 189 Homework4

Xu Zhihao

July 22, 2019

I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted.

Signature: Zhihao Xu

Contents

1 Logistic Regression with Newton's Method

(1) The cost function here is

$$J(w) = \lambda \|w\|_2^2 - \sum_{i=1}^{n} \left(y_i \ln s_i + (1 - y_i) \ln(1 - s_i) \right)$$

Compute the gradient

$$\begin{aligned}
\nabla_w J(w) &= 2\lambda w - \sum_{i=1}^{n} \left[\frac{y_i}{s_i} \nabla_w s_i + \frac{1 - y_i}{1 - s_i} \nabla_w (1 - s_i) \right] \\
&= 2\lambda w - \sum_{i=1}^{n} \left[\frac{y_i}{s_i} - \frac{1 - y_i}{1 - s_i} \right] s_i (1 - s_i) \nabla_w (X_i^T w) \\
&= 2\lambda w - \sum_{i=1}^{n} \left[y_i (1 - s_i) - s_i (1 - y_i) \right] x_i \\
&= 2\lambda w - \sum_{i=1}^{n} (y_i - s_i) x_i \\
&= 2\lambda w - X^T (y - s)
\end{aligned}$$

(2) Derive the Hessian

$$
\begin{aligned}
H = \nabla_w \left[\nabla_w^T J(w)\right] &= \nabla_w \left[2\lambda w^T - (y-s)^T X\right] \\
&= 2\lambda I - \nabla_w (y-s)^T X \\
&= 2\lambda I - \nabla_w \sum_{i=1}^{n} (y_i - s_i) x_i^T \\
&= 2\lambda I + s_i(1-s_i) x_i x_i^T \\
&= 2\lambda I + X^T \Omega X
\end{aligned}
$$

where $\Omega = diag\{s_1(1-s_1), s_2(1-s_2), \cdots, s_n(1-s_n)\}$

(3) In this problem, update equation for one iteration of Newton's method is

$$w^{\eta+1} = w^{\eta} - (2\lambda I + X^T \Omega X)^{-1}(2\lambda w - X^T(y - s))$$

(4) (a) $s^{(0)} = \begin{bmatrix} 0.9526 \\ 0.7311 \\ 0.7311 \\ 0.2689 \end{bmatrix}$

(b) $w^{(1)} = \begin{bmatrix} -0.3865 \\ 1.404 \\ -2.284 \end{bmatrix}$

(c) $s^{(1)} = \begin{bmatrix} 0.8731 \\ 0.8237 \\ 0.2932 \\ 0.2198 \end{bmatrix}$

(d) $w^{(2)} = \begin{bmatrix} -0.512 \\ 1.453 \\ -2.163 \end{bmatrix}$

2 l_1- and l_2-Regularization

(1) Reformulate the cost function

$$\begin{aligned} J(w) &= \|Xw - y\|_2^2 + \lambda \|w\|_1 \\ &= y^T y + w^T X^T X w - 2y^T X w + \lambda \|w\|_1 \\ &= y^T y + \sum_{i=1}^{n} w_i^2 n - 2y^T X_i w_i + \lambda |w_i| \end{aligned}$$

If we take $g(y) = y^T y$ and $f(X_{*i}, w_i, y, \lambda) = w_i^2 n - 2y^T X_{*i} w_i + \lambda |w_i|$, the cost function $J(w)$ is in the form

$$J(w) = g(y) + \sum_{i=1}^{n} f(X_{*i}, w_i, y, \lambda)$$

(2) Take the gradient of cost function, when $w_{*i} > 0$

$$\begin{aligned}
\nabla_w J = 2nw_i - 2X_{*i}^T y + \lambda|1| = 0 \\
\Rightarrow 2w_i n = 2X_{*i}^T y - \lambda \\
\Rightarrow w_{*i} = \frac{2X_{*i}^T y - \lambda}{2n}
\end{aligned}$$

(3) Similarly, when $w_{*i} < 0$,

$$w_{*i} = \frac{2X_{*i}^T y + \lambda}{2n}$$

(4) When $w_{*i} = 0$, the condition needs to satisfy two inequality

$$\begin{cases} 2X_{*i}^T y - \lambda \leq 0 \\ 2X_{*i}^T y + \lambda \geq 0 \end{cases}$$

Therefore the condition for $w_{*i} = 0$ is

$$-\lambda \leq 2X_{*i}^T y \leq \lambda$$

(5) With ridge regression, the gradient of cost function goes to be

$$\nabla_w J = 2nw_i - 2X_{*i}^T y + 2\lambda w_i = 0$$

$$\Rightarrow w_i = \frac{y^T X_{*i}}{n + \lambda}$$

Here the condition for $w_{*i} = 0$ is

$$X_{*i}^T y = 0$$

3 Regression and Dual Solutions

(1) Derive $\nabla|w|^4$,

$$\begin{aligned}
\nabla_w|w|^4 &= \nabla_w(w^Tw)^2 \\
&= 2(w^Tw)\nabla_w(w^Tw) \\
&= 2(w^Tw) \times 2w \\
&= 4(w^Tw)w
\end{aligned}$$

Then for $\nabla_w|X \cdot w - y|^4$

$$\begin{aligned}
\nabla_w|X \cdot w - y|^4 &= 2(X \cdot w - y)^T(X \cdot w - y)\nabla_w(X \cdot w - y)^T(X \cdot w - y) \\
&= 2(X \cdot w - y)^T(X \cdot w - y)\left[2X^T(X \cdot w - y)\right] \\
&= 4(X \cdot w - y)^T(X \cdot w - y)X^T(X \cdot w - y)]
\end{aligned}$$

(2) First we show w^* is unique. We need to show the Hessian of cost function $J(w) = |X \cdot w - y|^4 + \lambda |w|^2$ is positive definite.

$$\begin{aligned}
\nabla_w J(w) &= 4(Xw-y)^T(Xw-y)X^T(Xw-y) + 2\lambda w \\
\nabla_w^2 J(w) &= 4\nabla_w \left[(Xw-y)^T(Xw-y)(Xw-y)^T x\right] + 2\lambda \nabla_w w^T \\
&= 8x^T(Xw-y)(Xw-y)^T x + 4(Xw-y)^T(Xw-y)X^T X + 2\lambda I
\end{aligned}$$

For $\forall a \in \mathbb{R}^d \neq \mathbf{0}$,

$$\begin{aligned}
a^T \nabla_w^2 J(w) a &= 8\|(Xw-y)^T Xa\|_2^2 + 4C\|Xa\|_2^2 + 2\lambda\|a\|_2^2 \\
&> 0
\end{aligned}$$

where C = $(Xw-y)^T(Xw-y)$ is a constant. By $\nabla_w^2 J(w)$ is positive definite. So, the optimum w^* is unique. Then we are going to solve $\nabla_w J(w) = 0$

$$\nabla_w J(w) = 4(Xw-y)^T(Xw-y)X^T(Xw-y) + 2\lambda w = 0$$

we can get

$$w^* = \sum_{i=1}^{n} -\frac{2}{\lambda}(Xw-y)^T(Xw-y)(Xw-y)x = \sum_{i=1}^{n} a_i x_i$$

where $a_i = -\frac{2}{\lambda}(Xw-y)^T(Xw-y)(Xw-y)$

(3) Here the cost function is

$$J(w) = \frac{1}{n}\sum_{i=1}^{n} L(w^T x_i, y_i) + \lambda \|w\|_2^2$$

Take the gradient and set it equals 0

$$\nabla_w J(w) = \frac{1}{n}\sum_{i=1}^{n} L'(w^T x_i, y_i) x_i + 2\lambda w = 0$$

$$\Rightarrow w^* = \sum_{i=1}^{n} -\frac{1}{2\lambda n} L'(w^T x_i, y_i) x_i$$

So, the optimal solution still has the form $w^* = \sum_{i=1}^{n} a_i x_i$, with $a_i = -\frac{1}{2\lambda n} L'(w^T x_i, y_i)$. If the cost function is not convex, $\nabla_w J(w) = 0$ cannot make sure the minimum cost value. So, the optimum will not always have the form $w^* = \sum_{i=1}^{n} a_i x_i$

4 Wine Classification with Logistic Regression

For this question, all the figures are shown with the code after the write up, do not include again in the write up part.

(1) Here we use l_2 regularization. For question 1, we can get $\nabla_w J(w) = 2\lambda w - X^T(y-s)$, So the updating rule is

$$w^{\eta+1} = w^{\eta} - \epsilon[2\lambda w - X^T(y-s)]$$

(2) For stochastic gradient descent, the update equation is

$$w^{\eta+1} = w^{\eta} - \epsilon[2\lambda w - x_i^T(y-s)]$$

The batch gradient descent converges much faster than stochastic gradient descent, however SGD needs less computation in each iteration.

(3) This strategy is better than having a constant learning rate ϵ

(4) Kaggle username: Jack_xzh
Score: 0.97986

- **Code:**

```
In [748]: import numpy as np
          import scipy.io
          from tqdm import tnrange, tqdm_notebook
          import pandas as pd
          def results_to_csv(y_test):
              y_test = y_test.astype(int)
              df = pd.DataFrame({'Category': y_test})
              df.index += 1  # Ensures that the index starts at 1.
              df.to_csv('submission.csv', index_label='Id')

In [749]: path = "data.mat"
          data = scipy.io.loadmat(path)
          data_X = data["X"]
          data_y = data["y"].reshape((6000,))
          data_t = data["X_test"]
          print(data_y)

[0. 1. 0. ... 0. 0. 0.]


In [769]: def sigmoid(x):
              return 1/(1+np.exp(-x))

          def cost(w,lam):
              s = sigmoid(np.dot(X_train,w0))
              y = y_train
              return lam * np.linalg.norm(w,ord=2) \
                  - sum(y*np.log(s)+(1-y)*np.log(1-s))

In [770]: one = np.ones(6000)
          data_X = np.column_stack((one,data_X))
          one = np.ones(497)
          data_t = np.column_stack((one,data_t))

In [752]: X_train = data_X[0:5000]
          y_train = data_y[0:5000]
          X_validate = data_X[5000:6000]
          y_validate = data_y[5000:6000]

In [753]: print(X_train.shape)

(5000, 13)


In [755]: lam = 0.1
          epi = 0.000001
          w0 = np.zeros(13)
          Allcost = []
```

```
In [756]: s = sigmoid(np.dot(X_train,w0))
          for i in tqdm_notebook(range(20000)):
              w0 = w0 - epi * (2 * lam * w0 \
                           - np.matmul(X_train.T,y_train-s))
              s = sigmoid(np.dot(X_train,w0))
              Allcost.append(cost(w0,lam))

          print(Allcost[-1])
          w0
```

```
HBox(children=(IntProgress(value=0, max=20000), HTML(value='')))



664.70472619889
```

```
Out[756]: array([ 0.06912021,  0.85446636,  1.26896844, -0.38979783, -0.15031483
                  0.18845127,  0.03673803, -0.05861698,  0.08519431,  1.09549702
                  0.89099679, -0.64664271, -0.07544602])
```

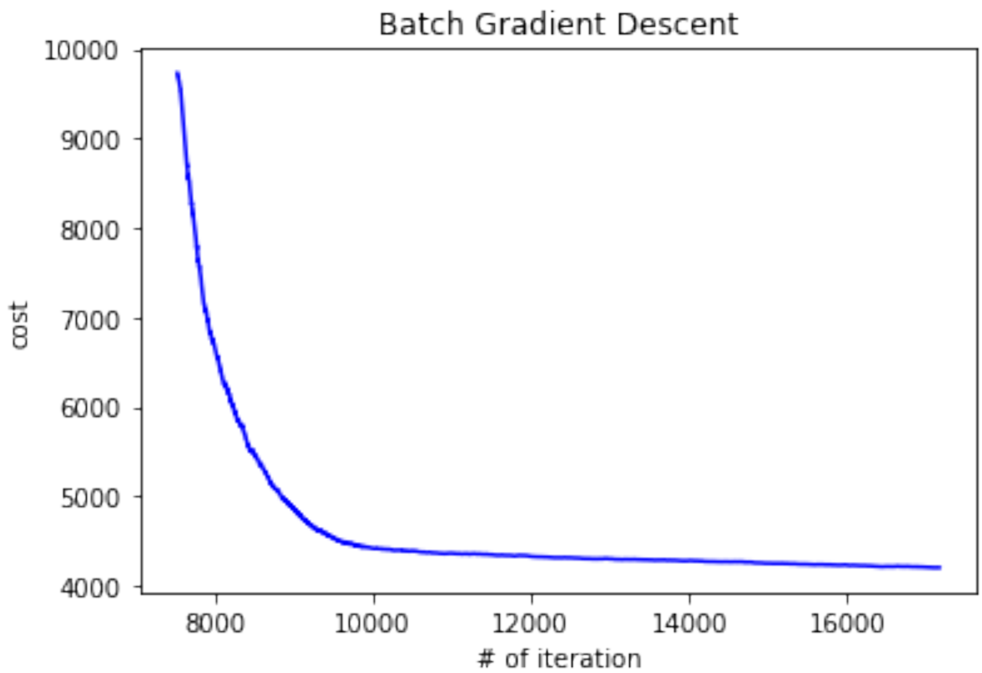
```
In [757]: # validation
          s = sigmoid(np.dot(X_validate,w0))
          y = np.where(s>0.5,1,0)
          sum(y==y_validate)/y_validate.size
```

```
Out[757]: 0.956
```

```
In [758]: s = sigmoid(np.dot(X_train,w0))
          y = np.where(s>0.5,1,0)
          sum(y==y_train)/y_train.size
```

```
Out[758]: 0.9584
```

```
In [765]: import matplotlib.pyplot as plt
          %matplotlib inline
          plt.plot(Allcost,c="blue")
          plt.xlabel("# of iteration")
          plt.ylabel("cost")
          plt.title("Batch Gradient Descent")
          plt.show()
```




```
In [760]: lam = 0.01
          epi = 0.000001
          w0 = np.ones(13)
          Allcost = []
          s = sigmoid(np.dot(X_train,w0))
          w0 = w0 - epi * (2 * lam * w0  \
                           - np.matmul(X_train.T,y_train-s))

In [761]: for j in range(20):
              shuffle = np.arange(X_train.shape[0])
              np.random.shuffle(shuffle)
              X_train = X_train[shuffle]
              y_train = y_train[shuffle]

              s = sigmoid(np.dot(X_train,w0))
              pred = np.where(s>0.5,1,0)
              index = np.where(pred!=y_train)
              # print(len(index[0]))

              for i in index[0]:
                  w0 = w0 - epi * (2*lam*w0 - X_train[i,].T * \
                                   (y_train[i] - sigmoid(np.dot(X_train[i,].T,w0)
                  Allcost.append(cost(w0,lam))

In [762]: print(Allcost[-1])
          w0
```

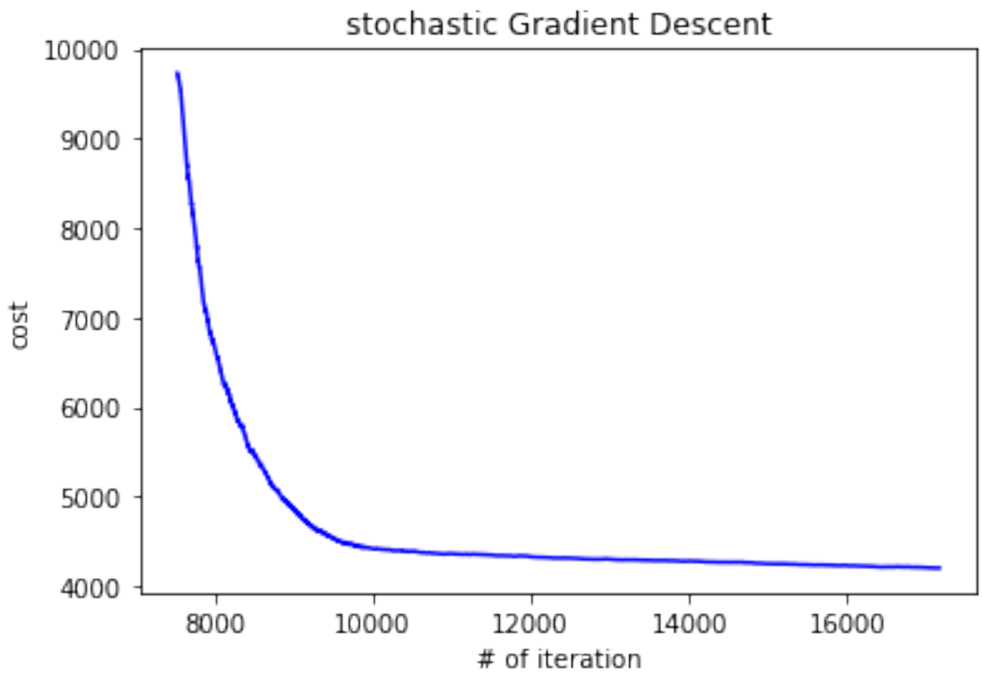
```
4203.8182363069545
```

```
Out[762]: array([ 0.98748506,  0.91948041,  0.9973006 ,  0.99548446,  0.9101857
                  0.99927577,  0.54598668, -0.48987923,  0.98757335,  0.96138626
                  0.99443415,  0.8661378 ,  0.99220328])
```

```
In [763]: # validation
          s = sigmoid(np.dot(X_validate,w0))
          y = np.where(s>0.5,1,0)
          sum(y==y_validate)/y_validate.size
```

```
Out[763]: 0.912
```

```
In [766]: import matplotlib.pyplot as plt
          %matplotlib inline
          x = list(range(1,len(Allcost)+1))
          plt.plot(x,Allcost,c="blue")
          plt.xlabel("# of iteration")
          plt.ylabel("cost")
          plt.title("stochastic Gradient Descent")
          plt.show()
```




```
In [557]: s = sigmoid(np.dot(X_train,w0))
          y = np.where(s>0.5,1,0)
          sum(y==y_train)/y_train.size
```

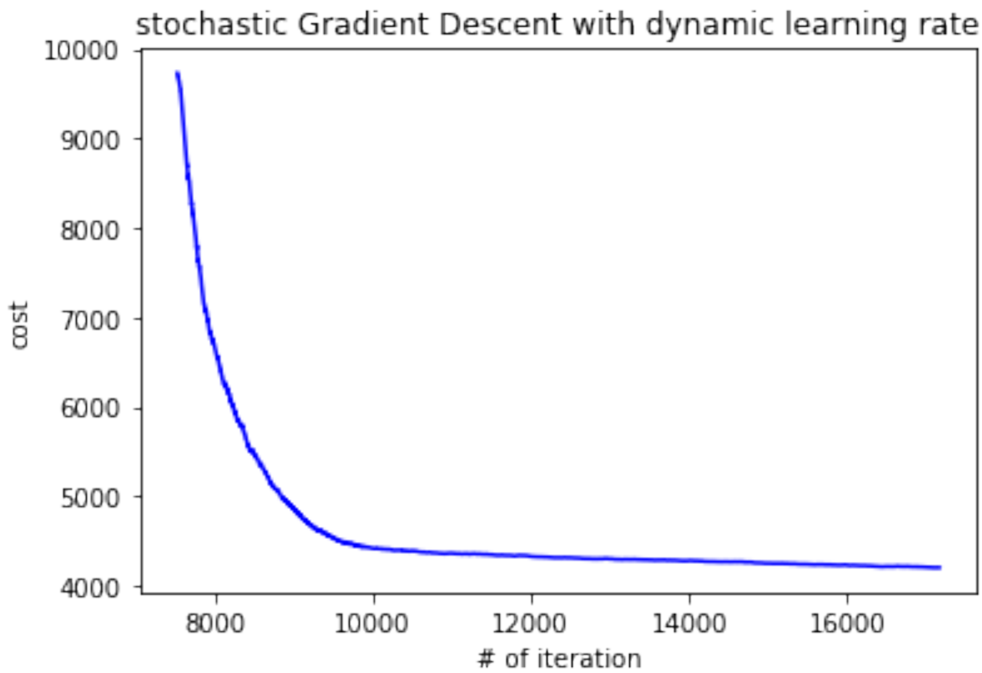
```
Out[557]: 0.9072
```

```
In [627]: # 3
          lam = 0.01
          epi0 = 0.01
          w0 = np.ones(13)
          Allcost = []
          s = sigmoid(np.dot(X_train,w0))
          w0 = w0 - epi * (2 * lam * w0 - np.matmul(X_train.T,y_train-s))
          t = 1
          for j in range(20):
              shuffle = np.arange(X_train.shape[0])
              np.random.shuffle(shuffle)
              X_train = X_train[shuffle]
              y_train = y_train[shuffle]

              s = sigmoid(np.dot(X_train,w0))
              pred = np.where(s>0.5,1,0)
              index = np.where(pred!=y_train)
              # print(len(index[0]))

              for i in index[0]:
                  epi = epi0/t
                  t += 1
                  w0 = w0 - epi * (2*lam*w0 - X_train[i,].T * \
                                   (y_train[i] - sigmoid(np.dot(X_train[i,].T,w0)
                  Allcost.append(cost(w0,lam))
```

```
In [768]: x = list(range(1,len(Allcost)+1))
          plt.plot(x,Allcost,c="blue")
          plt.xlabel("# of iteration")
          plt.ylabel("cost")
          plt.title("stochastic Gradient Descent with dynamic learning rate")
          plt.show()
```




```
In [695]: # 4
          X_train = data_X[0:6000]
          y_train = data_y[0:6000]
          lam = 0.1
          epi = 0.000001
          w0 = np.zeros(13)

In [706]: s = sigmoid(np.dot(X_train,w0))
          for i in tqdm_notebook(range(4000000)):
              w0 = w0 - epi * (2 * lam * w0 - np.matmul(X_train.T,y_train-s))
              s = sigmoid(np.dot(X_train,w0))

          print(cost(w0,lam))
          w0

HBox(children=(IntProgress(value=0, max=4000000), HTML(value='')))

322.19474913165055

Out[706]: array([-9.9197196 ,  0.87526119, 10.45840276, -0.09418173, -0.13610911
                  9.77873157,  0.04754924, -0.0626801 , -8.89426113,  4.48064516
                  8.47994628, -0.70052877, -0.37847519])
```

```
In [679]: s = sigmoid(np.dot(data_t,w0))
          y = np.where(s>0.5,1,0)
          results_to_csv(y)
```

5 Real World Spam Classification

I think the main reason is the timestamp is not linear separable. For example, 23:59 and 0:01 cannot be separated in linear SVM model. If we just use the time directly, the time cannot be linear separate even in a quadratic kernel.

I think Daniel can adjust on the time data first to make it linear separable. For the time t from 12:00 - 24:00, he can adjust it to 24 - t. For example, 23:59 should be adjusted to -0:01. Then he can use a quadratic kernel to separate the time linearly. Additionally, the mid-point can also be adjusted to the midnight like 3/4 o'clock, which can be get by averaging the sleeping time and waking up time.