Tutorial 8

CPSC 340

Logistic Regression

Stochastic Gradient Descent

Logistic Regression Model

- A discriminative probabilistic model for classification e.g. spam filtering
- Let $x \in \mathbb{R}^d$ be input and $y \in \{-1, 1\}$

Logistic Regression Model

- A discriminative probabilistic model for classification e.g. spam filtering
- Let $x \in \mathbb{R}^d$ be input and $y \in \{-1, 1\}$
- The probabilistic model with sigmoid function

$$p(y = 1|x) = \sigma(w^T x)$$
$$\sigma(\eta) = \frac{1}{1 + exp(-\eta)}$$

Logistic Regression Model

- A discriminative probabilistic model for classification e.g. spam filtering
- Let $x \in \mathbb{R}^d$ be input and $y \in \{-1, 1\}$
- The probabilistic model with sigmoid function

$$p(y = 1|x) = \sigma(w^T x)$$
$$\sigma(\eta) = \frac{1}{1 + exp(-\eta)}$$

- what is the probabilities of $p(y = -1|x)$?

Logistic Regression Model

- A discriminative probabilistic model for classification e.g. spam filtering
- Let $x \in \mathbb{R}^d$ be input and $y \in \{-1, 1\}$
- The probabilistic model with sigmoid function

$$p(y = 1|x) = \sigma(w^T x)$$
$$\sigma(\eta) = \frac{1}{1 + exp(-\eta)}$$

- what is the probabilities of $p(y = -1|x)$?
- $$p(y = -1|x) = 1 - p(y = 1|x) = \frac{1}{1 + exp(w^T x)}$$

Learning in Logistic Regression

- Let $X \in \mathbb{R}^{n \times d}$ and $y \in \{-1, 1\}^n$ be training data

Learning in Logistic Regression

- Let $X \in \mathbb{R}^{n \times d}$ and $y \in \{-1, 1\}^n$ be training data
- we can use logistic loss to learn the parameter vector w

$$f(w) = \frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$$

Learning in Logistic Regression

- Let $X \in \mathbb{R}^{n \times d}$ and $y \in \{-1, 1\}^n$ be training data
- we can use logistic loss to learn the parameter vector w

$$f(w) = \frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$$

- we want to find

$$w^* = \arg\min_{w \in \mathbb{R}^d} f(w)$$

Learning in Logistic Regression

- Let $X \in \mathbb{R}^{n \times d}$ and $y \in \{-1, 1\}^n$ be training data
- we can use logistic loss to learn the parameter vector w

$$f(w) = \frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$$

- we want to find

$$w^* = \arg\min_{w \in \mathbb{R}^d} f(w)$$

- Since $f(w)$ is convex function w.r.t. w we can use GD to find w^*

Learning in Logistic Regression

- Regularized loss function

$$f(w) = \frac{1}{n}\sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i)) + \frac{\lambda}{2}\|w\|^2$$

Learning in Logistic Regression

- Regularized loss function

$$f(w) = \frac{1}{n}\sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i)) + \frac{\lambda}{2}\|w\|^2$$

- Exercise: find the gradient of regularized f(w)?

Learning in Logistic Regression

- Regularized loss function

$$f(w) = \frac{1}{n}\sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i)) + \frac{\lambda}{2}\|w\|^2$$

- Exercise: find the gradient of regularized f(w)?
- Solution

$$f_i(w) = \log(1 + \exp(-y_i w^T x_i))$$

$$\nabla f_i(w) = \frac{-y_i x_i}{1 + \exp(y_i w^T x_i)}$$

$$\nabla f(w) = \frac{1}{n}\sum_{i=1}^{n} \frac{-y_i x_i}{1 + \exp(y_i w^T x_i)} + \lambda w$$

Learning in Logistic Regression

- Coding Exercise: we want to write a function to calculate the gradient and function. The following code is given. Write the code for **loglos_subfunc**.
-

```
function [f,g] = loglos(X,Y,w,lambda)
[n,d]=size(X);
g=zeros([d,1]);
f=0;
for i=1:n
    [f_i,g_i]=loglos_subfunc(X(i,:),Y(i),w);
    g=g+g_i;
    f=f+f_i;
end
g=g/n+lambda*w;
f=f/n+lambda/2*sum(w.^2);
end
```

Learning in Logistic Regression

- Solution
-

```
function [f_i,g_i]=loglos_subfunc(x_i,y_i,w)

z_i=y_i*x_i;
f_i=1+exp(-z_i*w);

%Gradient
g_i=-z_i*(1-1/f_i);

%function value f_i
f_i=log(f_i);
end
```

Learning Logistic Regression

- Exercise: Let Z be the transformation of X using some non-linear basis. What would be the new probabilistic model, logistic loss and its gradient?

Learning Logistic Regression

- Exercise: Let Z be the transformation of X using some non-linear basis. What would be the new probabilistic model, logistic loss and its gradient?
- Slolution:

$$p(y = 1|z) = \sigma(w^T z)$$

$$f(w) = \frac{1}{n}\sum_{i=1}^{n} \log(1 + \exp(-y_i w^T z_i)) + \frac{\lambda}{2}\|w\|^2$$

$$\nabla f(w) = \frac{1}{n}\sum_{i=1}^{n} \frac{-y_i z_i}{1 + \exp(y_i w^T z_i)} + \lambda w$$

Stochastic Gradient Descent

Gradient Descent Cost for Big Data

- Assume our loss function has the following format:

$$f(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w)$$

e.g.

$$f(w) = \frac{1}{n}\sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$$

Gradient Descent Cost for Big Data

- Assume our loss function has the following format:

$$f(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w)$$

  e.g.

$$f(w) = \frac{1}{n}\sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$$

- Using GD to find the best w:

$$w_{t+1} = w_t - \alpha_t \nabla f(w_t)$$

Gradient Descent Cost for Big Data

- Assume our loss function has the following format:

$$f(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w)$$

e.g.

$$f(w) = \frac{1}{n}\sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$$

- Using GD to find the best w:

$$w_{t+1} = w_t - \alpha_t \nabla f(w_t)$$

- But cost of computing $\nabla f(w)$ is $O(n)$ because of the sum:

$$\nabla f(w) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(w)$$

Gradient Descent Cost for Big Data

- Assume our loss function has the following format:

$$f(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w)$$

  e.g.

$$f(w) = \frac{1}{n}\sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$$

- Using GD to find the best w:

$$w_{t+1} = w_t - \alpha_t \nabla f(w_t)$$

- But cost of computing $\nabla f(w)$ is $O(n)$ because of the sum:

$$\nabla f(w) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(w)$$

- Cost of each iteration in GD could be enormous when n is large!

Stochastic Gradient Descent(SGD) for Big Data

- SGD algorithm: in each iteration pick a f_i randomly and use its gradient

$$i \sim unif(1, n) \qquad\qquad w_{t+1} = w_t - \alpha_t \nabla f_i(w_t)$$

Stochastic Gradient Descent(SGD) for Big Data

- SGD algorithm: in each iteration pick a f_i randomly and use its gradient

$$i \sim unif(1, n) \qquad\qquad w_{t+1} = w_t - \alpha_t \nabla f_i(w_t)$$

- ∇f_i is an unbiased approximation of ∇f

$$\mathbb{E}\left[\nabla f_i(w)\right] = \sum_{i=1}^{n} p(i) \nabla f_i(w) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w) = \nabla f(w)$$

Stochastic Gradient Descent(SGD) for Big Data

- SGD algorithm: in each iteration pick a f_i randomly and use its gradient

$$i \sim unif(1, n) \qquad\qquad w_{t+1} = w_t - \alpha_t \nabla f_i(w_t)$$

- ∇f_i is an unbiased approximation of ∇f

$$\mathbb{E}\left[\nabla f_i(w)\right] = \sum_{i=1}^{n} p(i) \nabla f_i(w) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w) = \nabla f(w)$$

- Cost of each iteration in SGD is constant

Stochastic Gradient Descent(SGD) for Big Data

- SGD algorithm: in each iteration pick a f_i randomly and use its gradient

$$i \sim unif(1, n) \qquad\qquad w_{t+1} = w_t - \alpha_t \nabla f_i(w_t)$$

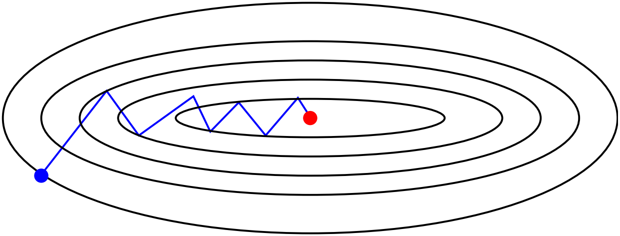
- ∇f_i is an unbiased approximation of ∇f

$$\mathbb{E}\left[\nabla f_i(w)\right] = \sum_{i=1}^{n} p(i) \nabla f_i(w) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w) = \nabla f(w)$$

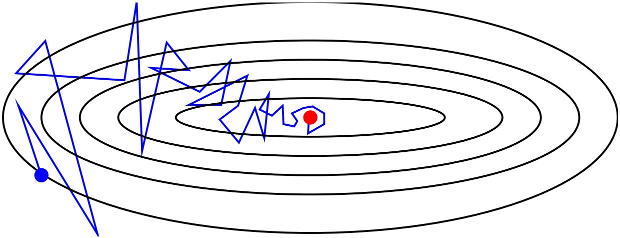
- Cost of each iteration in SGD is constant
- It does not move toward minimizer in each iteration, so is slower than GD

SGD vs GD

- GD



-
- SGD



-

Variance of SGD

- Variance of SGD in each iteration:

$$Var[\nabla f_i(w)] = \frac{1}{n}\sum_{i=1}^{n}\|\nabla f_i(w) - \nabla f(w)\|^2$$

Variance of SGD

- Variance of SGD in each iteration:

$$Var[\nabla f_i(w)] = \frac{1}{n}\sum_{i=1}^{n} \|\nabla f_i(w) - \nabla f(w)\|^2$$

- If variance is small, every step jumps in the right direction

Variance of SGD

- Variance of SGD in each iteration:

$$Var[\nabla f_i(w)] = \frac{1}{n}\sum_{i=1}^{n} \|\nabla f_i(w) - \nabla f(w)\|^2$$

- If variance is small, every step jumps in the right direction
- If variance is large, many steps jump in wrong direction!

Variance of SGD

- Variance of SGD in each iteration:

$$Var[\nabla f_i(w)] = \frac{1}{n}\sum_{i=1}^{n} \|\nabla f_i(w) - \nabla f(w)\|^2$$

- If variance is small, every step jumps in the right direction
- If variance is large, many steps jump in wrong direction!
- Variance can be controlled by decreasing step size or by variance reduction technique

Variance of SGD

- To get convergence we need decreasing step sizes

Variance of SGD

- To get convergence we need decreasing step sizes
- But it cannot shrink too quickly

Variance of SGD

- To get convergence we need decreasing step sizes
- But it cannot shrink too quickly
- Two main conditions for decreasing step sizes:

$$\sum_{t=1}^{\infty} \alpha_t = \infty \qquad \text{we can get everywhere} \tag{1}$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty \qquad \text{effect of variance goes to zero} \tag{2}$$

Variance of SGD

- To get convergence we need decreasing step sizes
- But it cannot shrink too quickly
- Two main conditions for decreasing step sizes:

$$\sum_{t=1}^{\infty} \alpha_t = \infty \qquad \text{we can get everywhere} \tag{1}$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty \qquad \text{effect of variance goes to zero} \tag{2}$$

- Setting $\alpha_t = O(1/t)$ satisfies the above conditions but it is too slow

Variance of SGD

- To get convergence we need decreasing step sizes
- But it cannot shrink too quickly
- Two main conditions for decreasing step sizes:

$$\sum_{t=1}^{\infty} \alpha_t = \infty \qquad \text{we can get everywhere} \tag{1}$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty \qquad \text{effect of variance goes to zero} \tag{2}$$

- Setting $\alpha_t = O(1/t)$ satisfies the above conditions but it is too slow
- In practice:

$$\alpha_t = \beta/(t+\gamma)$$
$$\alpha_t = O(1/\sqrt{t}) \text{ or } O(1/t^{\beta}) \text{ for } \beta \in (0, 1)$$

SGD for logistic Loss

$$\alpha_t = \beta/(t+\gamma), \quad f(w) = \frac{1}{n}\sum_{i=1}^{n} \log(1+\exp(-y_i w^T x_i)) + \frac{\lambda}{2}\|w\|^2$$

- Coding Exercise: Using the **loglos_subfunc** from previous exercise, complete the SGD code for logistic loss.

```
function w=loglosSGD(X,Y,w_0,beta,gamma,lambda, T)

w=w_0;
[n,d]=size(X);

for t=1:T

    ......

end
end
```

- T: # of iteration
- w_0: initial value

SGD for logistic Loss

- Solution:

```
function w=loglosSGD(X,Y,w_0,beta,gamma,lambda, T)

w=w_0;
[n,d]=size(X);

for t=1:T
    %generating i randomly
    i=randi(n);

    %computing the gradient of f_i
    [f_i,g_i]=loglos_subfunc(X(i,:),Y(i),w);

    %step size for this iteration
    alpha_t=beta/(t+gamma);

    %SGD update with l2 regularizer
    w=(1-alpha_t*lambda)*w-alpha_t*g_i;

end
end
```

-

Variance Reduction Technique

- Using mini-batch

Variance Reduction Technique

- Using mini-batch
  - In each iteration t, we make a random mini-batch B_t

Variance Reduction Technique

- Using mini-batch
  - In each iteration t, we make a random mini-batch B_t
  - $w_{t+1} = w_t - \frac{\alpha_t}{|B_t|} \sum_{f_i \in B_t} \nabla f_i(w_t)$

Variance Reduction Technique

- Using mini-batch
  - In each iteration t, we make a random mini-batch B_t
  - $w_{t+1} = w_t - \frac{\alpha_t}{|B_t|} \sum_{f_i \in B_t} \nabla f_i(w_t)$
  - Variance is inversely proportional to the mini-batch size

Variance Reduction Technique

- Using mini-batch
  - In each iteration t, we make a random mini-batch B_t
  - $w_{t+1} = w_t - \frac{\alpha_t}{|B_t|} \sum_{f_i \in B_t} \nabla f_i(w_t)$
  - Variance is inversely proportional to the mini-batch size
- Using auxiliary memory: SAG method

Variance Reduction Technique

- Using mini-batch
  - In each iteration t, we make a random mini-batch B_t
  - $w_{t+1} = w_t - \frac{\alpha_t}{|B_t|} \sum_{f_i \in B_t} \nabla f_i(w_t)$
  - Variance is inversely proportional to the mini-batch size
- Using auxiliary memory: SAG method
  - It uses an extra memory y with n cells and each cell y_i stores d value

Variance Reduction Technique

- Using mini-batch
  - In each iteration t, we make a random mini-batch B_t
  - $w_{t+1} = w_t - \frac{\alpha_t}{|B_t|} \sum_{f_i \in B_t} \nabla f_i(w_t)$
  - Variance is inversely proportional to the mini-batch size
- Using auxiliary memory: SAG method
  - It uses an extra memory y with n cells and each cell y_i stores d value
  - In each iteration t, we pick a f_i randomly and evaluate the gradient $\nabla f_i(w_t)$

Variance Reduction Technique

- Using mini-batch
  - In each iteration t, we make a random mini-batch B_t
  - $w_{t+1} = w_t - \frac{\alpha_t}{|B_t|} \sum_{f_i \in B_t} \nabla f_i(w_t)$
  - Variance is inversely proportional to the mini-batch size
- Using auxiliary memory: SAG method
  - It uses an extra memory y with n cells and each cell y_i stores d value
  - In each iteration t, we pick a f_i randomly and evaluate the gradient $\nabla f_i(w_t)$
  - Store $\nabla f_i(w_t)$ in y_i

Variance Reduction Technique

- Using mini-batch
  - In each iteration t, we make a random mini-batch B_t
  - $w_{t+1} = w_t - \frac{\alpha_t}{|B_t|} \sum_{f_i \in B_t} \nabla f_i(w_t)$
  - Variance is inversely proportional to the mini-batch size
- Using auxiliary memory: SAG method
  - It uses an extra memory y with n cells and each cell y_i stores d value
  - In each iteration t, we pick a f_i randomly and evaluate the gradient $\nabla f_i(w_t)$
  - Store $\nabla f_i(w_t)$ in y_i
  - $w_{t+1} = w_t - \frac{\alpha}{n} \sum_{i=1}^{n} y_i$

Variance Reduction Technique

- Using mini-batch
  - In each iteration t, we make a random mini-batch B_t
  - $w_{t+1} = w_t - \frac{\alpha_t}{|B_t|} \sum_{f_i \in B_t} \nabla f_i(w_t)$
  - Variance is inversely proportional to the mini-batch size
- Using auxiliary memory: SAG method
  - It uses an extra memory y with n cells and each cell y_i stores d value
  - In each iteration t, we pick a f_i randomly and evaluate the gradient $\nabla f_i(w_t)$
  - Store $\nabla f_i(w_t)$ in y_i
  - $w_{t+1} = w_t - \frac{\alpha}{n} \sum_{i=1}^{n} y_i$
  - cost of each iteration is constant

Variance Reduction Technique

- Using mini-batch
  - In each iteration t, we make a random mini-batch B_t
  - $w_{t+1} = w_t - \frac{\alpha_t}{|B_t|} \sum_{f_i \in B_t} \nabla f_i(w_t)$
  - Variance is inversely proportional to the mini-batch size
- Using auxiliary memory: SAG method
  - It uses an extra memory y with n cells and each cell y_i stores d value
  - In each iteration t, we pick a f_i randomly and evaluate the gradient $\nabla f_i(w_t)$
  - Store $\nabla f_i(w_t)$ in y_i
  - $w_{t+1} = w_t - \frac{\alpha}{n} \sum_{i=1}^{n} y_i$
  - cost of each iteration is constant
  - convergence is fast since we use constant step size

Variance Reduction Technique

- Using mini-batch
  - In each iteration t, we make a random mini-batch B_t
  - $w_{t+1} = w_t - \frac{\alpha_t}{|B_t|} \sum_{f_i \in B_t} \nabla f_i(w_t)$
  - Variance is inversely proportional to the mini-batch size
- Using auxiliary memory: SAG method
  - It uses an extra memory y with n cells and each cell y_i stores d value
  - In each iteration t, we pick a f_i randomly and evaluate the gradient $\nabla f_i(w_t)$
  - Store $\nabla f_i(w_t)$ in y_i
  - $w_{t+1} = w_t - \frac{\alpha}{n} \sum_{i=1}^{n} y_i$
  - cost of each iteration is constant
  - convergence is fast since we use constant step size
  - The memory requirement could be restrictive when n is enormous