CSC 446 Notes: Lecture 18

Typed by Hao Luo

1 The Problem

To train maximum entropy (logistic regression) models, we maximized the probability of the training data over possible feature weights λ:

$$\max_{\lambda} \prod_{n} P(Y_n|X_n)$$

It is to maximize

$$L = \log \prod_{n} \frac{1}{Z_{X_n}} e^{\sum_i \lambda_i f_i}$$

Of course we can solve it by using gradient ascend, but we today will talk about using an approximation of Newton's method.

2 Preliminary

For quadratic objective function

$$f(x) = \frac{1}{2} x^\top A x + b^\top x + c$$

Newton's iteration is given by

$$x_{k+1} = x_k + (\nabla^2 f(x_k))^{-1} \nabla f(x_k)$$

However, the exact version of Newton's method involves a few problems as follows:

- We need to compute Hessian $\nabla^2 f$, which is expensive.
- We also need to invert Hessian, which is also a costly operation.
- Furthermore, we need to store Hessian, which is expensive in terms of space.

So in order to compute Newton's iteration in a fast but relatively accurate way, an approximation should be developed. L-BFGS is one of them.

3 The L-BFGS Algorithm

Let B_k denote our approximation of the Hessian $\nabla^2 f(x_k)$ and then we can write Newton's iteration as

$$x_{k+1} = x_k - \alpha B_k^{-1} \nabla f(x_k)$$

Because the Hessian can be seen as the second order derivative of f, we wish to choosse B_k such that:

$$B_k(x_{k+1} - x_k) = \nabla f(x_{k+1}) - \nabla f(x_k)$$

Let

$$s_k = x_{k+1} - x_k$$

$$y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$$

then we have

$$B_k s_k = y_k$$

This is to say that, our approximation is a solution of above equation. Consider

$$B_k = \frac{y_k y_k^\top}{s_k^\top y_k}$$

Because

$$B_k s_k = \frac{y_k y_k^\top s_k}{s_k^\top y_k} = \frac{y_k (y_k^\top s_k)}{s_k^\top y_k} = y_k$$

Further, let H_k be our approximation of $(\nabla^2 f(x_k))^{-1}$, the inverse of Hessian. We will have:

$$s_k = H_k y_k$$

H_k is a solution of above equation. One such H_k is given by

$$H_k = \frac{s_k s_k^\top}{s_k^\top y_k}$$

So, we want a direct formula of computing a symmetric H_{k+1} from H_k. That is, we want to fill in the ? term in following equation

$$H_{k+1} = H_k + \frac{s_k s_k^\top}{s_k^\top y_k} + ?$$
$$\text{such that } s_k = H_{k+1} y_k$$

With careful proofs and calculation, we get

$$H_{k+1} = (I - \rho_k s_k y_k^\top) H_k (I - \rho_k y_k s_k^\top) + \rho_k s_k s_k^\top \tag{1}$$

where $\rho_k = \frac{1}{s_k^\top y_k}$.

4 Proof of the method

The problem can be formalized as minimizing $\|H_{k+1} - H_k\|$ such that $H_{k+1} y_k = s_k$ and $H_{k+1}^\top = H_{k+1}$. Here $\|\cdot\|$ is defined as follows:

$$\|A\|^2 = \|W^{\frac{1}{2}} A W^{\frac{1}{2}}\|_f^2$$

where $\|A\|_f$ is defined as the square sum of all entries, $\sum_{i,j} a_{ij}^2$ and W is any matrix such that $W s_k = y_k$.

It is easy to verify that $H_{k+1} y_k = s_k$ and H_{k+1} is symmetric, as follows:

$$\begin{aligned} H_{k+1} y_k &= (I - \rho_k s_k y_k^\top) H_k (I - \rho_k y_k s_k^\top) y_k + \rho_k s_k s_k^\top y_k \\ &= ... H_k (y_k - \rho_k s_k^\top y_k) + \rho_k s_k s_k^\top y_k \\ &= ... H_k (y_k - y_k) + s_k \\ &= s_k \end{aligned}$$

5 L-BFGS Algorithm

L-BFGS algorithm tries to approximate $H_{k+1}\nabla f(x_{k+1})$ together. From 1, we can unroll the last m H_k's. Then we will compute H_{k+1} directly from H_{k-m}.

$$\begin{aligned} H_{k+1} =& V_k^\top ..V_{k-m}^\top H_{k-m} V_{k-m}..V_k \\ &+ \rho_{k-m} V_k^\top ..V_{k-m}^\top s_{k-m} s_{k-m}^\top V_{k-m}..V_k \\ &+ .. \\ &+ \rho_k s_k s_k^\top \end{aligned}$$

In the equation above, $V_k = I - \rho_k s_k y_k^\top$. The algorithm is

Algorithm 1 L-BFGS

Require: H_{k-m}, s_i, y_i
  $q \leftarrow \nabla f_k$
  for $i = k-1..k-m$ **do**
    $\alpha_i \leftarrow \rho_i s_i^\top q$
    $q \leftarrow q - \alpha_i y_i$
  end for
  $r \leftarrow H_{k-m} q$
  for $i = k-m..k-1$ **do**
    $\beta \leftarrow \rho_i y_i^\top r$
    $r \leftarrow r + s_i(\alpha_i - \beta)$
  end for
  return r

In the algorithm, $V_k = I - \rho_k y_k s_k^\top$. This algorithm needs to keep track of s_k and y_k in the last m steps and each step requires $2n$ space(n for s_k and n for y_k). So a total of $O(2mn)$ space is needed.

There are many user libraries that have already implemented this algorithm, so we can just use them for our computing.