1 The Problem

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

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

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

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

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 choose $B_k$ such that:

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

Let

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

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

$$B_k s_k = y_k$$

Therefore

$$B_k y_k = s_k$$

$$B_k y_{k+1} = s_{k+1}$$

$$B_k y_{k+2} = s_{k+2}$$

...
\[ y_k = \nabla f(x_{k+1}) - \nabla f(x_k) \]

then we have

\[ B_k s_k = y_k \]

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

\[ B_k = \frac{y_k y_k^\top}{s_k y_k} \]

Because

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

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

\[ H_k = \frac{s_k s_k^\top}{s_k y_k} \]

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

\[ H_{k+1} = H_k + \frac{s_k s_k^\top}{s_k y_k} + \rho_k s_k \]

such that \( s_k = H_{k+1} y_k \)

With careful proofs and calculation, we get

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

(1)

where \( \rho_k = \frac{1}{s_k y_k} \).

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:

\[ \| A \|^2 = \| W^\frac{1}{2} AW^\frac{1}{2} \|^2 \]

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:

\[ H_{k+1} y_k = (I - \rho_k s_k y_k^\top) H_k (I - \rho_k y_k s_k^\top) y_k + \rho_k s_k s_k^\top y_k \]
\[ = \ldots H_k (y_k - \rho_k s_k y_k) + \rho_k s_k s_k^\top y_k \]
\[ = \ldots H_k (y_k - y_k) + s_k \]
\[ = s_k \]
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 $H_{k-m}$.

$H_{k+1} = V_k^\top V_{k-m} H_{k-m} V_{k-m} \ldots V_k$

$$
+ \rho V_{k-m} \ldots V_{k-m} s_{k-m} s_{k-m}^\top V_{k-m} \ldots V_k
+ \ldots
+ \rho k s_k s_k^\top
$$

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

\begin{algorithm}
\caption{L-BFGS}
\begin{algorithmic}
\Require $H_{k-m}, s_i, y_i$
\State $q \leftarrow \nabla f_k$
\For {$i = k - 1 \ldots k - m$}
\State $\alpha_i \leftarrow \rho_i s_i^\top q$
\State $q \leftarrow q - \alpha_i y_i$
\EndFor
\State $r \leftarrow H_{k-m} q$
\For {$i = k - m \ldots k - 1$}
\State $\beta \leftarrow \rho_i y_i^\top r$
\State $r \leftarrow r + s_i (\alpha_i - \beta)$
\EndFor
\Return $r$
\end{algorithmic}
\end{algorithm}

In the algorithm, $V_k = I - \rho k s_k y_k^\top$. 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.