逻辑回归：从入门到精通

柳超
腾讯

September 2, 2013

Abstract

逻辑回归(Logistic Regression, 简称LR)可以说是互联网领域应用最广的自动分类算法：从单机运行的垃圾邮件自动识别程序到需要成百上千台机器支撑的互联网广告投放系统，其算法主干都是LR。由于其普适性与重要性，大家在工作中都或多或少的谈论着LR，但是笔者发现很多同学对于LR的理解可以进一步提高与深化。所以，笔者准备了这样一个关于逻辑回归从入门到精通的文章和同学们一同探讨。本文的目标不是像基维百科那样泛泛而谈、面面俱到地介绍LR，相反而是更注重对LR的理解和其背后的优化算法的掌握，从而使大家更有信心的实现现实中需要的大规模LR模型，并根据实际问题持续地改进它。另外，由于求解LR是一个性质很好的优化问题，本文也借此机会比较系统的介绍了从最速梯度下降法，到牛顿方法，再到拟牛顿方法（包括DFP, BFGS, L-BFGS）这一系列数值优化算法的脉络，也算是对数值优化算法中文教程的一个补充吧。最后，还请各位领导、大拿、和冲在第一线的研究猿与攻城狮们不吝赐教、切磋琢磨、一同进步！

1 动机与目标读者

大家在平时的工作和学习当中经常会遇到各种决策问题：例如这封邮件是不是垃圾邮件，这个用户是不是对这个商品感兴趣，这个房子该不该买等等。熟悉或者接触过机器学习(Machine Learning, 简称ML)的同学知道如果我们需要对这类问题进行决策的时候，最常用的方法就是构建一个叫做分类器(Classifier)的程序。这种程序的输入是待决策问题的一系列特征(feature)，输出就是程序判定的结果。以垃圾邮件分类为例，每一封邮件就是一个待决策问题，而通常使用的特征就是从这个邮件本身抽取的一系列我们认为可能相关的信息，例如，发件人、邮件长度、时间、邮件中的关键词、标点符号、是否有多个收件人等等。给定了这些特征，我们的垃圾邮件分类器就可以判定出这封邮件是否是垃圾邮件。至于怎么得到这个垃圾邮件分类器程序，通常的做法是通过某种机器学习算法。之所以称其为“学习”，是因为这些算法通常需要一些已经标注好的样本（例如，100封邮件，每封信已经被明确标注为是否是垃圾邮件），然后这个算法就自动的产生一个关于这个问题的自动分类器程序。我们在这篇文章中将要讲的逻辑回归(Logistic Regression, 简称LR)就是最常用的一个机器学习分类算法。

很多同学可能知道机器学习中有几十种分类器，那么我们为什么偏偏挑LR来讲呢？原因有三

1. LR模型原理简单，并且有一个现成的叫LIBLINEAR的工具库[1]，易于上手，并且效果不错。

2. LR可以说是互联网上最常用也是最有影响力的分类算法。LR几乎是所有广告系统中和推荐系统中点击率(Click Through Rate (CTR))预估模型的基本算法。

3. LR同时也是现在炙手可热的”深度学习“(Deep Learning)的基本组成单元，扎实的掌握LR也将有助于你学好深度学习。

但是本文并不是一篇关于LR的科普性文章，如果你想泛泛的了解LR，最好的办法是去基维百科或者找一本像样的机器学习教材翻一下。相反的，本文的目标是使你不仅仅“知其然”，并且更“知其所以然”，真正做到从入门到精通，从而能更加有信心的解决LR实践中出现的新问题。我们可以粗略的把从入门到精通分为三个层次

- **了解LR**：了解LR模型、L1和L2规则化(Regularization)、为什么L1规则化更倾向于产生稀疏模型(Sparse Model)、以及稀疏模型的优点。

- **理解LR**：理解LR模型的学习算法、能够独自推导基于L-BFGS的L1和L2规则化的LR算法，并将其在MPI平台上并行化实现。

- **改进LR**：能够在实际中自如的应用LR，持续改进LR来解决实际中未曾预见到的问题。例如，当数据中的正样本远远小于负样本的情况下（例如，广告点击率预估问题），该怎么调整LR？当数据中有相当部分缺失时该如何调整算法？

由于求解LR是一个性质很好的无约束优化问题，本文在介绍LR的同时，也相对系统的介绍了无约束优化问题中几个常用的数值方法，包括最速梯度下降方法、牛顿方法，和拟牛顿方法中的DFP, BFGS, 与L-BFGS。期望同学们能在知道了解这些算法的同时能真正明白其原理与应用场景，例如理解为什么L-BFGS中的二次循环方法(two iteration method)能够近似计算牛顿方向，并且可以轻松的并行化。这些算法是关于无约束问题的通用优化算法，应用场景非常广泛，但笔者并未发现关于它们比较系统化的、又同时比较容易理解的中文教程，本文也算是填补这方面空白的一个尝试吧。所以，希望能在学习LR的同时将优化算法一并学了，相得益彰吧。

所以，本文预期的读者大概有如下几类

1. **高阶机器学习人员：硕士毕业后5年及以上的机器学习经历**：您就把这个当做一个关于LR和无约束优化算法的回顾材料好了，如有错误和不足请加以斧正。

2. **中阶机器学习人员：硕士毕业后3~5年的机器学习经历**：您可以把这个当做一个学习材料，把以前学过的东西串在一起，查漏补缺，做到真正懂得LR和相关的优化算法，从而能对工程实践作出正确的指导。

3. **入门机器学习人员：硕士毕业后少于3年的机器学习经历**：请您拿出纸和笔，把里面的公式一个个推导一遍，从而当你的leader告诉你做某些事情的时候，你知道如何下手。

[1] http://www.csie.ntu.edu.tw/~cjlin/liblinear/

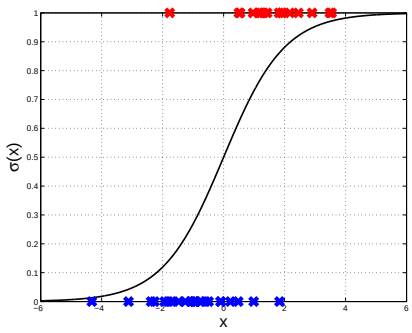


Figure 1: 逻辑函数与逻辑回归

4. **机器学习人员以外的高富帅与白富美们**: 您只需要知道LR是一个好用的自动分类算法，吩咐研究猿和攻城狮做就好了。另外，还可以用这篇文章嘲弄机器学习屌丝们：累死累活，死那么多脑细胞，挣那两儿钱;-)

总而言之，不管你在机器学习上的造诣几何，希望这篇文章或多或少的都能给你带来点什么。笔者非常欢迎各方人士对本文以及任何与机器学习、数据挖掘相关问题的垂询、切磋、与探讨。好吧，闲话不多说，进入正题。

2 初识逻辑回归

作为一个基本的机器学习场景，给定N个训练样本$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)$，其中$\mathbf{x}_i \in \mathbb{R}^n$是一个$n$维向量用来表示第$i$个样本在这$n$个特征上的取值，而$y_i \in \{-1, +1\}$表示了此样本是正样本(+1)还是负样本(-1)。逻辑回归模型就是通过一个叫逻辑函数(Logistic Function)的函数$\sigma(x) = \frac{1}{1+exp(-x)}$将第$i$个样本的特征向量$\mathbf{x}_i$与该样本为正样本的概率联系了起来

$$p(y_i = +1|\mathbf{x}_i, \mathbf{w}) = \sigma(y_i \mathbf{w}^T \mathbf{x}_i) = \frac{1}{1 + exp(-\mathbf{w}^T \mathbf{x}_i)}.$$

同时，由于

$$p(y_i = -1|\mathbf{x}_i, \mathbf{w}) = 1 - p(y_i = +1|\mathbf{x}_i, \mathbf{w}) = \sigma(y_i \mathbf{w}^T \mathbf{x}_i).$$

所以，我们可以把他们综合为一个式子

$$p(y_i = \pm 1|\mathbf{x}_i, \mathbf{w}) = \sigma(y_i \mathbf{w}^T \mathbf{x}_i) = \frac{1}{1 + exp(-y \mathbf{w}^T \mathbf{x}_i)}. \quad (1)$$

可以看出，参数$\mathbf{w}$对$\mathbf{x}$的n个维度做不同的加权，从而算出分数$\mathbf{w}^T \mathbf{x}_i$。这个分数然后被S形的逻辑函数将压到0到1，作为$\mathbf{x}$对应样本为正样本的概率。逻辑回归的目标就是要找到最好的$\mathbf{w}$，使得正样本的分数都比较大，同时负样本的分数都比较小。

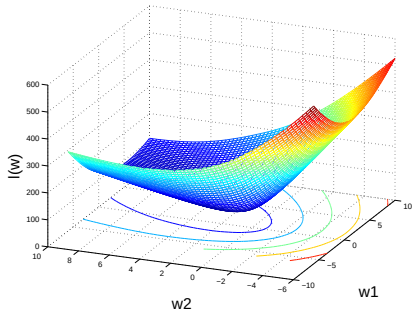


Figure 2: 逻辑回归损失函数

图 1画出了S形的逻辑函数，和一些由红蓝符号标记的正负样本。可以看出，恰当的$\mathbf{w}$将尽可能多的正（负）样本放在分数大于（小于）0的部分，而这恰恰是如何寻找最好$\mathbf{w}$的基本原则。更严格的讲，LR是通过最大似然估计(Maximum Likelihood Estimation, 简称MLE)原则来找到最好的$\mathbf{w}$。其基本想法是找到这样一个$\mathbf{w}$使得所有样本在根据公式1的情况下最大，即

$$max_{\mathbf{w}} \sum_{i=1}^{N} log(p(y_i = \pm 1|\mathbf{x}_i, \mathbf{w})) = -\sum_{i=1}^{N} log(1 + exp(-y_i \mathbf{w}^T \mathbf{x}_i))$$

由于通常来说，优化问题倾向于解决最小化问题，所以，上述最大化问题等价于最小化下面的$l(\mathbf{w})$

$$l(\mathbf{w}) = \sum_{i=1}^{N} log(1 + exp(-y_i \mathbf{w}^T \mathbf{x}_i)). \tag{2}$$

我们说$(\mathbf{w})$是一个凸函数，从而最小化它是一个相对容易的事情。为什么这么说呢？图 2给出了$l(\mathbf{w})$在二维情况下的一个图示，可以看出$l(\mathbf{w})$是向下凸的，所以直观来看，当我们处在不是最小点的某个点上，我们可以沿着这个曲面滑下去，从而达到，并且稳定在最小点。各种优化算法的区别就在于选择沿哪个方向滑下去，而这个方向通常是由目标函数在当前点的一阶倒数（又名梯度、函数值增大最快的方向）与二阶倒数（又名海森矩阵）决定的。那么下面我们先计算出$l(\mathbf{w})$一阶及二阶导数。

利用逻辑函数的性质$\sigma'(x) = \sigma(x)(1 - \sigma(x))$和导数的链式法则，我们可以算出$l(\mathbf{w})$对$\mathbf{w}$的梯度为

$$\mathbf{g} = \nabla_{\mathbf{w}} l(\mathbf{w}) = \sum_{i}^{N} (\sigma(y_i \mathbf{w}^T \mathbf{x}_i) - 1) y_i \mathbf{x}_i.$$

$l(\mathbf{w})$的二阶导数（海森矩阵）$\nabla^2_{\mathbf{w}} l(\mathbf{w})$相对来说不是很显而易见（除非你对矩阵微积分(Matrix Calculus)相当熟悉），所以，我们就先算出其中的一个元素，

然后在把它总结为矩阵形式。海森矩阵的(k,s)元素B(k,s)可通过如下方式算出

$$\begin{aligned} B(k,s) &= \nabla^2_{\mathbf{w}} l(\mathbf{w})(k,s) = \frac{\partial \mathbf{g}(k)}{\partial \mathbf{w}(s)} \\ &= \frac{\partial}{\partial \mathbf{w}(s)} \sum_i^N y_i \mathbf{x}_i(k)(\sigma(y_i \mathbf{w}^T \mathbf{x}_i) - 1) \\ &= \sum_i^N y_i \mathbf{x}_i(k)\sigma(y_i \mathbf{w}^T \mathbf{x}_i)(1 - \sigma(y_i \mathbf{w}^T \mathbf{x}_i)) * (y_i \mathbf{x}_i(s)) \qquad (3) \\ &= \sum_i^N \sigma(\mathbf{w}^T \mathbf{x}_i)(1 - \sigma(\mathbf{w}^T \mathbf{x}_i))\mathbf{x}_i(k)\mathbf{x}_i(s) \qquad (4) \end{aligned}$$

其中$\mathbf{x}(s)$表示向量$\mathbf{x}$的第s个元素。

式子4将式子3中的y_i完全抛去是因为首先y_i^2恒为1，并且$\sigma(x)(1-\sigma(x))$是偶函数，从而y_i是+1还是−1都无所谓了。也就是说$l(\mathbf{w})$的海森矩阵并不依赖于各个训练样本到底是正样本还是负样本。有了$B(s,k)$，将B写成矩阵形式就简单了许多

$$\begin{aligned} B = \nabla^2_{\mathbf{w}} l(\mathbf{w}) &= \sum_i^N \sigma(\mathbf{w}^T \mathbf{x}_i)(1 - \sigma(\mathbf{w}^T \mathbf{x}_i))\mathbf{x}_i \mathbf{x}_i^T \qquad (5) \\ &= \mathbf{X} A \mathbf{X}^T \succ 0, \qquad (6) \end{aligned}$$

其中$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n]$，$A$为对角矩阵，其第$i$个元素$A(i,i) = \sigma(\mathbf{w}^T \mathbf{x}_i)(1 - \sigma(\mathbf{w}^T \mathbf{x}_i)) > 0$。式子6表明$l(\mathbf{w})$的海森矩阵为正定矩阵，从而表明$l(\mathbf{w})$是关于$\mathbf{w}$的严格凸函数。严格凸函数是我们做优化时候最想遇到的，因为它的最小值存在并唯一，也就是说，沿着某种方向持续下降总能找到最小点，尽管有时候速度会比较慢。我们在第4和第4章节具体讲解如何下降找到最好的$\mathbf{w}$，但在那之前，我们需要先理解规则化(Regularization)的作用。

3 L1 vs. L2规则化

规则化的意义从学术上讲是用来避免学到的模型过度拟合(overfitting)训练数据。以式子2为例。如果我们的目标仅仅是最小化$l(\mathbf{w})$，而对$\mathbf{w}$没有任何限制，那么我们可以随意的调节$\mathbf{w}$各个维度的值使得它最好的符合这N个样本点，从而最小化$l(\mathbf{w})$。这样得到的$\mathbf{w}$的各个维度的绝对值通常会很大：一些大正数，一些大负数。这种模型虽然很好的匹配训练样本，但在对新样本做预测的时候，这些绝对值很大的取值就会使预测值偏离真正值很远。

图3给出了一个过度拟合的例子，其中红点是已知的数据点，蓝色虚线是产生这些数据点的真正趋势线，而黑色代表一个过度拟合的模型，它严格的拟合了所有已知点，但可以看出这个上上下下的折叠线并没有很好的捕获数据点代表的趋势，所以并不是一个很好的预测模型。

为避免发生这种情况，我们在$l(\mathbf{w})$加一个关于$\mathbf{w}$大小的项，称为规则化因子(Regularizer)。我们通常使用$\mathbf{w}$的模作为规则化因子

$$f(\mathbf{w}) = l(\mathbf{w}) + \lambda \|\mathbf{w}\| \qquad (7)$$

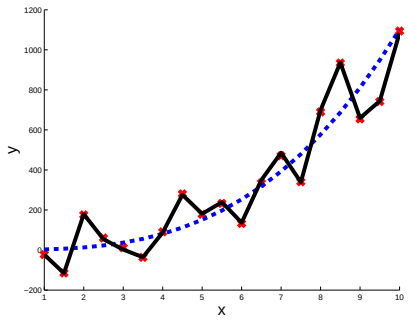


Figure 3: 模型过度拟合(overfitting)

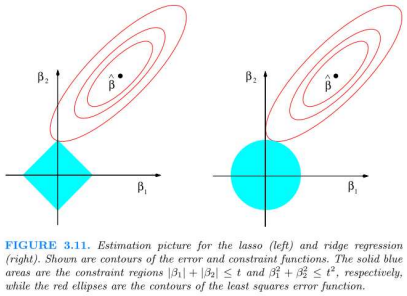


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \le t$ *and* $\beta_1^2 + \beta_2^2 \le t^2$*, respectively, while the red ellipses are the contours of the least squares error function.*

Figure 4: L1和L2规则化的对比：图片来源于[5]第3.4章

其中$\|\mathbf{w}\|$可以是L1模，称为L1规则化

$$\|\mathbf{w}\|_1 = \sum_{i=1}^{n} |\mathbf{w}(i)|$$

或者L2模(严格意义上的L2模还需要对下面开根号)，称为L2规则化

$$\|\mathbf{w}\|_2 = \sum_{i=1}^{n} \mathbf{w}^2(i) = \mathbf{w}^T\mathbf{w}.$$

不管用L1、L2，还是其他模做规范化，其基本原理都是一样的，即在最小化损失函数$l(\mathbf{w})$的同时，我们还要考虑$\mathbf{w}$的模带来的贡献，从而避免$\mathbf{w}$取一些绝度值很大的值。

那么L1和L2规范化有什么区别呢？答案是L1规则化通常导致稀疏的模型，也就是使用L1规则化得到的$\mathbf{w}$中有更多的维度为0，而这些为0的维度就代表了不是很相关的维度，从而起到了特征选择(Feature Selection)的作用。那么为什么同样为$\mathbf{w}$的模，L1规则化就能得到比L2规则化更稀疏的模型呢？通常的理解是根据经典教材[5]的图示4：L1规则化代表$\mathbf{w}$可取的值是转置的方形而L2对应的是圆形，这样损失函数$l(\mathbf{w})$的最小值更容易在L1对应的边角上取得，从而

这些维度就变成0了。作为这种解释的补充，下面笔者给出一种基于解析的解释。

我们首先看一下L1规则化因子对第$\mathbf{w}(k)$项的一阶微分

$$\frac{\partial \|\mathbf{w}\|_1}{\partial \mathbf{w}(k)} = \begin{cases} \delta(\mathbf{w}(k)) & \text{if } \mathbf{w}(k) \neq 0 \\ \text{undefined} & \text{if } \mathbf{w}(k) = 0 \end{cases} \tag{8}$$

其中$\delta(x)$是关于x的符号函数: x为正值时取值为1，为负值时取值为-1。当我们试图最小化$\|\mathbf{w}\|_1$的时候，不管$\mathbf{w}(k)$离0有多远或多近，我们把它往0推的“力量”是恒定的，要么+1要么-1，直到0为止。相对的，L2规则化因子的一阶微分是

$$\frac{\partial \|\mathbf{w}\|_2}{\partial \mathbf{w}(k)} = 2\mathbf{w}(k) \tag{9}$$

所以，L2规则化把$\mathbf{w}(k)$往0推的“力量”是随着$\mathbf{w}(k)$趋近于0而减小的，从而$\mathbf{w}(k)$是慢慢的趋近于0，而非像L1那样子干净利落的达到0。所以，L1给出的模型通常更稀疏，而稀疏模型的一个显而易见的优点就是它在部署时可以节省很多内存。例如，在我们常见的广告推荐系统中，上百亿的特征，对于非稀疏模型，仅仅存储$\mathbf{w}$（而不考虑相应的数据结构）将占据40G的内存。而且通常这种大的模型都需要发布在多个数据中心，所以稀疏的模型具有很强的实践意义。当然，L1在统计学习理论上也有很多优点，这里就不再深入讨论，感兴趣的同学可以参考斯坦福教授Andrew Ng的文章[13]。

最后再说明一点，由于任何模$\|\mathbf{w}\|$都是关于$\mathbf{w}$的凸函数，所以以任何模作规则化因子都不会改变LR的凸函数性质。唯一的不同就在于我们是最小化式子7中的$f(\mathbf{w})$，而非仅仅$l(\mathbf{w})$。下面我们分别介绍如何找到LR在L2与L1规则化下的最优$\mathbf{w}$。

4 求解L2规则化的逻辑回归

L2规则化的LR表示如下

$$min_{\mathbf{w}} f(\mathbf{w}) = \sum_{i=1}^{N} log(1 + exp(-y_i \mathbf{w}^T \mathbf{x}_i)) + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w},$$

其中λ是用于调节损失函数与规则化因子之间的相对重要程度。对于这样的无约束、连续可微的凸函数优化问题，最直接的方法就是最速梯度下降法：假设$\mathbf{w}_t$是在第t个循环的当前点，那么第$t+1$个点通过下述方式得到

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \alpha \nabla f(\mathbf{w}_t), \tag{10}$$

其中参数α是所谓的步长用于控制每个循环下降的程度以保证收敛。由于这种方法沿着（反向）梯度的方向搜索，所以被称为最速梯度下降法。

虽然最速梯度下降法简单明了易于实现，其收敛速度却不尽人意，尤其是在最小值附近会以一种曲折的形式慢速的逼近最小点，所以更常用的方法是著名的牛顿方法(Newton's Method)。其基本原理是在当前的点处用二次泰勒展开来近似需要最小化的目标函数，找出此近似函数的极小点方向，然后做线搜

索(line search)找到合适的下降步长。具体来说，假若我们要最小化的函数为二次可微的$f(\mathbf{x})$，我们用f_k指代$f(\mathbf{x})$在当前点x_k的取值，即$f_k = f(x_k)$，然后再第$k+1$循环，我们想要找到点x_{k+1}，使得f_{k+1}拥有足够的下降来保证最终收敛。将$f(\mathbf{x})$在x_k处二阶泰勒展开为关于$p = x - x_k$的函数

$$m_k(p) \equiv f_k + p^T \nabla f_k + \frac{1}{2} p^T B_k p \approx f(\mathbf{x}), \tag{11}$$

其中$B_k = \nabla_x^2 f(\mathbf{x})$为$f(\mathbf{x})$在$x_k$处的海森矩阵，然后我们通过最小化$m_k(p)$来找到对应的$p$。应用极小点的必要条件为导数为0，令$m_k(p) = 0$

$$\nabla m_k(p) = \nabla f_k + B_k p = 0$$

通过求解此线性方程组或者等价的直接计算，我们可以得到在第k个循环时的搜索方向

$$p_k = -B_k^{-1} \nabla f_k \tag{12}$$

此方向又称为牛顿方向。最后我们在此牛顿方向上进行线搜索以找到合适的步长α进行下降

$$x_{k+1} = x_k + \alpha p_k, \quad \alpha \in (0, +\infty).$$

回溯线搜索(backtrack line search)是一种通用使用的线搜索方法。

尽管牛顿方法具有很好的收敛速度，但其代价也是显而易见的。首先，每一个循环都需要计算并存储$B_k \in \mathbb{R}^{n \times n}$。这对常见的大规模LR应用场景是很大的挑战，甚至是不可能的。另外，我们还需要求B_k的逆矩阵，或者等价的求解一个大型的（n个变量）线性方程组。所以，严格的牛顿方法更适用于中小型的无约束优化问题。

为了避免每个循环都要重新计算并求解海森矩阵的逆，研究人员提出了一系列拟牛顿(Quasi-Newton)方法。之所以称其为拟牛顿方法是因为这些方法通过拟合的方式找到一个近似的海森矩阵用于计算下降方向。拟牛顿法的可行性在于严格的牛顿方法也是一种（基于泰勒的二次展开）近似方法，所以对海森矩阵的近似只是近似方法上的进一步近似，只要不是差的特别远，也能起到相应调整下降方向的效果。按照拟牛顿方法的进化历史，我们按照DFP、BFGS、L-BFGS的顺序介绍这一系列拟牛顿方法。

4.1 DFP方法

DFP是历史上第一个拟牛顿方法，为William C. **D**avidson在1959年发明，并由Roger **F**letcher和Michael J. D. **P**owell完善，所以此方法被称为DFP。DFP的主要思想是在第k个循环时，在上一个循环中的海森矩阵B_{k-1}的基础上得到在当前点x_k的海森矩阵B_k，进而通过式子12得到牛顿方向。具体来说，在第k个循环，我们已经知道$x_k, x_{k-1}, \nabla f(x_k), \nabla f(x_{k-1})$，需要算出这个牛顿迭代时候需要的$B_k$，进而通过式子12算出来怎么从$x_k$找到$x_{k+1}$。DFP方法通过求解下面这个优化问题得到$B_k$

$$\begin{aligned} \mathbf{min} \quad & \|B - B_{k-1}\| \\ \mathbf{s.t.} \quad & B = B^T, \quad B s_{k-1} = y_{k-1}, \end{aligned} \tag{13}$$

其中$s_{k-1} = x_k - x_{k-1}$, $y_{k-1} = \nabla f(x_k) - \nabla f(x_{k-1})$。

此优化问题中的目标函数要求最终找到的B和B_{k-1}越像越好，即B在每次迭代中变化不要太大。约束条件$B = B^T$要求B是对称的，因为海森矩阵都是对称的。$Bs_{k-1} = y_{k-1}$这个限制是最重要的，没有它，显然$B = B_{k-1}$就是最优解。那么为什么要有这个限制呢？因为它是B_k需要满足的一个必要条件：把式子Eqn 11两边分别对x取导数，我们得到

$$\nabla f(\mathbf{x}) \approx \nabla f_k + B_k(x - x_k).$$

由于x_k和x_{k-1}离得比较近，我们可以把$x = x_{k-1}$代进上面的式子得到

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

其实就是

$$y_{k-1} = B_k s_{k-1}.$$

这个式子被称为*secant*等式，其作为x_k点处海森矩阵需要满足的必要条件作为优化问题13的一个约束条件。幸运的是，优化问题13存在如下解析解

$$B_k = (I - \rho_{k-1} y_{k-1} s_{k-1}^T) B_{k-1} (I - \rho_{k-1} s_{k-1} y_{k-1}^T) + y_{k-1} \rho_{k-1} y_{k-1}^T, \qquad (14)$$

其中$\rho_k = (y_k^T s_k)^{-1}$。

所以，对任何无约束的二次可微的优化问题（包括L2-LR），DFP先计算出在初始点x_0处的海森矩阵B_0，求逆算出在x_0处的牛顿方向，并通过线搜索找到x_1。找到了x_1，通过式子14计算出在x_1处的近似海森矩阵B_1，求逆，线搜索得到算出x_2等等直至收敛。可以看出，相比牛顿方法，DFP省去了每次重新计算海森矩阵的运算，但它仍然需要在每此存储并更新海森矩阵，并通过求逆算出牛顿方向。下面的BFGS方法将展示如何省去求逆的运算。

4.2 BFGS方法

既然DFP可以通过每步增量的方式更新海森矩阵B，但每一步又要对得到的B做求逆运算得到牛顿方向，那么有没有可能直接近似海森矩阵的逆呢？在1970年，四位大师Charles G. **B**royden, Roger **F**letcher, Daniel **G**oldfarb和David F. **S**hanno各自**独立地**发现这种可行性，并发明了这个所谓的BFGS算法。由于BFGS要增量的近似海森矩阵的逆$H_k = B_k^{-1}$，所以secant等式就变成了$H_k y_{k-1} = s_{k-1}$, 从而我们要求解的优化问题如下所示

$$\begin{aligned} \mathbf{min} \quad & \|H - H_{k-1}\| \\ \mathbf{s.t.} \quad & H = H^T, \quad By_{k-1} = s_{k-1}, \end{aligned} \qquad (15)$$

我们可以发现这个问题和DFP中的优化问题13其实是完全一样的。唯一不同的是y_k和s_k的关系互换了。所以，就像我们能猜到的，最优解H_k也与DFP中的B_k具有相同的形式，仅仅是把y和s互换罢了，即

$$H_k = (I - \rho_{k-1} s_{k-1} y_{k-1}^T) H_{k-1} (I - \rho_{k-1} y_{k-1} s_{k-1}^T) + s_{k-1} \rho_{k-1} s_{k-1}^T. \qquad (16)$$

所以，从DFP到BFGS的进步是通过直接近似海森矩阵的逆省去了求牛顿方向中需要的矩阵逆运算，从而达到了既不需要每步都重新算，又不需

要求逆的效果。在维度n不是很高的情况下，BFGS在内存中保留并持续更新$H_k \in \mathbb{R}^{n\times n}$，进而通过矩阵和向量相乘$H_k\nabla f_k$达到牛顿方法的收敛速度。但是，在$n$比较大的情况下，由于$H_k$占有$\Theta(n^2)$的内存量，例如广告推荐中的十亿量级的特征$n \sim 10^9$对应着100万TB内存，存储并更新$H_k$就会变得非常不方便甚至不可能。这种需求导致了下面这个天才算法的出现：Limited-memory BFGS，简称L-BFGS。

4.3 L-BFGS方法

L-BFGS在每一个（假设第k个）循环，保存之前m个步骤中的$y \in \mathbb{R}^n$和$s \in \mathbb{R}^n$来直接近似计算出牛顿方向$H_k\nabla f_k \in \mathbb{R}^n$。从而不仅像BFGS一样省去了每个循环计算海森矩阵并求逆的运算，而且省去了存储H_k的必要，毕竟最终只有牛顿方向才是我们想要的。我们马上可以看到L-BFGS的内存需求是$O(m*n)$而非$\Theta(n^2)$。

那么，L-BFGS是怎么通过前m个循环中的y和s来近似$H_k\nabla f_k$的呢？由于L-BFGS依赖前m个循环中记录的信息，那么我们来看看H_k和前m个循环中的y和s是什么关系。H_k和H_{k-1}的关系已经由式子16给出。如果我们定义$V_k = I - \rho_k y_k s_k^T$，那么BFGS给出的$H_k$更新规则就可以表示如下

$$H_k = V_{k-1}^T H_{k-1} V_{k-1} + s_{k-1}\rho_{k-1}s_{k-1}^T = V_{-1}^T H_{-1} V_{-1} + s_{-1}\rho_{-1}s_{-1}^T,$$

其中为了符号清晰我们省去了角标k。请注意这一次展开(unrolling)给出了$s_{-1}\rho_{-1}s_{-1}^T$这一项和$V_{-1}^T \dots V_{-1}$这样一个装饰H_{-1}的装饰器。

如果我们进一步将上式中的H_{-1}用同样的方法展开，我们将会得到

$$\begin{aligned} H_k &= V_{-1}^T H_{-1} V_{-1} + s_{-1}\rho_{-1}s_{-1}^T \\ &= V_{-1}^T V_{-2}^T H_{-2} V_{-2} V_{-1} + V_{-1}^T s_{-2}\rho_{-2}s_{-2}^T V_{-1} + s_{-1}\rho_{-1}s_{-1}^T \end{aligned}$$

其中蓝色的那一项是由H_{-1}展开产生的$s_{-2}\rho_{-2}s_{-2}^T$经过现有的装饰器$V_{-1}^T \dots V_{-1}$装饰而来，并且同时此时装饰器叠加成了$V_{-1}^T V_{-2}^T \dots V_{-2} V_{-1}$这样一个新的装饰器。那么当我们向后展开$m$步时，我们是否应该得到如下的结果呢？

$$\begin{aligned} H_k &= V_{-1}^T H_{-1} V_{-1} + s_{-1}\rho_{-1}s_{-1}^T \\ &= V_{-1}^T V_{-2}^T H_{-2} V_{-2} V_{-1} + V_{-1}^T s_{-2}\rho_{-2}s_{-2}^T V_{-1} + s_{-1}\rho_{-1}s_{-1}^T \\ &= \cdots \\ &= (V_{-1}^T V_{-2}^T \cdots V_{-m}^T) H_{-m} (V_{-m} V_{-(m-1)} \cdots V_{-1}) \qquad (17) \\ &\quad + (V_{-1}^T V_{-2}^T \cdots V_{-(m-1)}^T) s_{-m}\rho_{-m}s_{-m}^T (V_{-(m-1)} V_{-(m-2)} \cdots V_{-1}) \\ &\quad + (V_{-1}^T V_{-2}^T \cdots V_{-(m-2)}^T) s_{-(m-1)}\rho_{-(m-1)}s_{-(m-1)}^T (V_{-(m-2)} V_{-(m-3)} \cdots V_{-1}) \\ &\quad + \cdots \\ &\quad + V_{-1}^T s_{-2}\rho_{-2}s_{-2}^T V_{-1} \qquad (18) \\ &\quad + s_{-1}\rho_{-1}s_{-1}^T \qquad (19) \end{aligned}$$

为了确认我们确实理解了这种迭代所产生的结构，让我们考虑一下在最后一步到底发生了什么。最后一步是指当我们展开了$m-1$次后，这时我们拥有的装饰器是

$$(V_{-1}^T V_{-2}^T \cdots V_{-(m-1)}^T) \dots (V_{-(m-1)} V_{-(m-1)} \cdots V_{-1}),$$

而我们需要的是用

$$H_{-(m-1)} = V_{-m}^T H_{-m} V_{-m} + s_{-m}\rho_{-m}s_{-m}^T$$

做最后一步展开。具体表述如下所示。

$$\begin{aligned}
H_k &= (V_{-1}^T V_{-2}^T \cdots V_{-(m-1)}^T) H_{-(m-1)} (V_{-(m-1)} V_{-(m-2)} \cdots V_{-1}) \\
&\quad + (V_{-1}^T V_{-2}^T \cdots V_{-(m-2)}^T) s_{-(m-1)}\rho_{-(m-1)} s_{-(m-1)}^T (V_{-(m-2)} V_{-(m-3)} \cdots V_{-1}) \\
&\quad + \cdots \\
&\quad + V_{-1}^T s_{-2}\rho_{-2} s_{-2}^T V_{-1} \\
&\quad + s_{-1}\rho_{-1} s_{-1}^T \\
&= (V_{-1}^T V_{-2}^T \cdots V_{-m}^T) H_{-m} (V_{-m} V_{-(m-1)} \cdots V_{-1}) \\
&\quad + (V_{-1}^T V_{-2}^T \cdots V_{-(m-1)}^T) s_{-m}\rho_{-m} s_{-m}^T (V_{-(m-1)} V_{-(m-2)} \cdots V_{-1}) \\
&\quad + (V_{-1}^T V_{-2}^T \cdots V_{-(m-2)}^T) s_{-(m-1)}\rho_{-(m-1)} s_{-(m-1)}^T (V_{-(m-2)} V_{-(m-3)} \cdots V_{-1}) \\
&\quad + \cdots \\
&\quad + V_{-1}^T s_{-2}\rho_{-2} s_{-2}^T V_{-1} \\
&\quad + s_{-1}\rho_{-1} s_{-1}^T,
\end{aligned}$$

这个式子展示了如何通过前m个步骤中的y和s，以及H_{k-m}来表示H_k。但是仅仅把H_k表述出来并没有真正解决$O(n^2)$的存储问题（其中的任何一项都是一个$n \times n$的矩阵），上面的式子应用的仍是BFGS的迭代公式，还没有牵扯到L-BFGS的任何事情。那么下面我们看看L-BFGS是如何天才性的解决这一问题的。

Algorithm 1 L-BFGS: Two Iteration Algorithm

$q = \nabla f_k \in R^n$
for $i = 1, 2, \cdots, m$ **do**
  $\alpha_i = \rho_{-i} s_{-i}^T q \in R$
  $q = q - \alpha_i y_{-i} \in R^n$
end
$r = H_{-m} q \in R^n$
for $i = m, m-1, \cdots, 1$ **do**
  $\beta = \rho_{-i} y_{-i}^T r \in R$
  $r = r + s_{-i}(\alpha_i - \beta) \in R^n$
end
return $r \in R$

算法 1，也就是L-BFGS，揭示了如何通过两个循环的方法(two iteration method)计算出$H_k \nabla f_k$。首先，在第一个循环中，假若我们用q_i来指代q在第i个

循环结束时候的取值，并且令$q_0 = \nabla f_k$，那么

$$\begin{aligned} q_i &= q_{i-1} - \rho_{-i} y_{-i} s_{-i}^T q_{i-1} = (I - \rho_{-i} y_{-i} s_{-i}^T) q_{i-1} \\ &= V_{-i} q_{i-1} = V_{-i} V_{-(i-1)} q_{i-2} = \cdots \\ &= (V_{-i} V_{-(i-1)} \cdots V_{-1}) q_0 \\ &= (V_{-i} V_{-(i-1)} \cdots V_{-1}) \nabla f_k, \end{aligned}$$

所以，在整个第一个大循环（02行到05行）结束的时候，我们得到

$$q_m = (V_{-m} V_{-(m-1)} \cdots V_{-1}) \nabla f_k,$$

注意这正是式子17第一项右半部分。除了最后算出q_m，第一个大循环的每一个循环还都产生了一个α，将第i个循环产生的记为α_i，那么

$$\alpha_i = \rho_{-i} s_{-i}^T q_{i-1} = \rho_{-i} s_{-i}^T (V_{-(i-1)} V_{-(i-2)} \cdots V_{-1} \nabla f_k)$$

这些正是式子17除第一项各项的右半部分。或许我们从$i = 1$看起更加清晰

$$\alpha_1 = \rho_{-1} s_{-1}^T \nabla f_k,$$

正是Eqn. 19的右半部分。

同样的$i = 2$

$$\alpha_2 = \rho_{-2} s_{-2}^T V_{-1} \nabla f_k,$$

是Eqn. 18的右半部分。所以，在第一个大循环结束的时候，我们就有了计算$H_k \nabla f_k$所需要的$m + 1$项中每一项的右半部分：q_m, α_m, α_{m-1}, $\cdots$, α_2, α_1。下面我们看看第二个大循环是怎么将每一项的左半部分凑出来的。

对应第二个大循环来说，我们用r_t来指代当$i = t$这个循环（而非第t个循环）结束后r的取值，06行其实算出来了$r_{m+1} = H_{-m} q_m$, 所以我们需要的就是这$m + 1$项中左边的部分。第二个大循环给出的迭代关系如下

$$\begin{aligned} r_t &= r_{t+1} + s_{-t}(\alpha_t - \rho_{-t} y_{-t}^T r_{t+1}) \\ &= (I - s_{-t} \rho_{-t} y_{-t}^T) r_{t+1} + s_{-t} \alpha_t \\ &= V_{-t}^T r_{t+1} + s_{-t} \alpha_t, \qquad t = m, m-1, \ldots, 2, 1 \end{aligned}$$

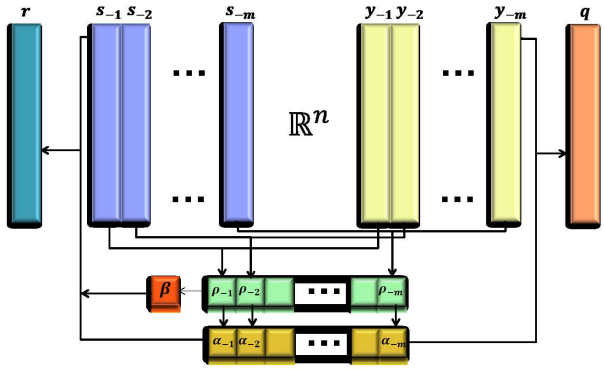


Figure 5: 图示L-BFGS的内存消耗

根据这个公式我们来看看算法1的第11行返回的r_1到底是什么

$$
\begin{aligned}
r_1 &= V_{-1}^T r_2 + s_{-1}\alpha_1 \\
&= V_{-1}^T r_2 + s_{-1}\rho_{-1}s_{-1}^T\nabla f_k \\
&= V_{-1}^T V_{-2}^T r_3 + V_{-1}^T s_{-2}\alpha_2 + s_{-1}\rho_{-1}s_{-1}^T\nabla f_k \\
&= V_{-1}^T V_{-2}^T r_3 + V_{-1}^T s_{-2}\rho_{-2}s_{-2}^T V_{-1}\nabla f_k + s_{-1}\rho_{-1}s_{-1}^T\nabla f_k \\
&= V_{-1}^T V_{-2}^T V_{-3}^T r_4 + V_{-1}^T s_{-2}\rho_{-2}s_{-2}^T V_{-1}\nabla f_k + s_{-1}\rho_{-1}s_{-1}^T\nabla f_k \\
&= \cdots \\
&= (V_{-1}^T V_{-2}^T \cdots V_{-m}^T) r_{m+1} + (V_{-1}^T V_{-2}^T \cdots V_{-(m-1)}^T) s_{-m}\alpha_m \\
&\quad + \cdots + V_{-1}^T s_{-2}\rho_{-2}s_{-2}^T V_{-1}\nabla f_k + s_{-1}\rho_{-1}s_{-1}^T\nabla f_k \\
&= (V_{-1}^T V_{-2}^T \cdots V_{-m}^T) H_{-m} q_m \\
&\quad + (V_{-1}^T V_{-2}^T \cdots V_{-(m-1)}^T) s_{-m}\rho_{-m}s_{-m}^T (V_{-(m-1)} V_{-(m-2)} \cdots V_{-1})\nabla f_k \\
&\quad + \cdots + V_{-1}^T s_{-2}\rho_{-2}s_{-2}^T V_{-1}\nabla f_k + s_{-1}\rho_{-1}s_{-1}^T\nabla f_k \\
&= (V_{-1}^T V_{-2}^T \cdots V_{-m}^T) H_{-m} (V_{-m} V_{-(m-1)} \cdots V_{-1})\nabla f_k \\
&\quad + (V_{-1}^T V_{-2}^T \cdots V_{-(m-1)}^T) s_{-m}\rho_{-m}s_{-m}^T (V_{-(m-1)} V_{-(m-2)} \cdots V_{-1})\nabla f_k \\
&\quad + \cdots + V_{-1}^T s_{-2}\rho_{-2}s_{-2}^T V_{-1}\nabla f_k + s_{-1}\rho_{-1}s_{-1}^T\nabla f_k
\end{aligned}
$$

而这个正是前面式子17中计算$H_k\nabla f_k$的$m+1$项！

回顾一下，L-BFGS通过一系列的向量与向量的点积与加减操作直接计算出了（反）牛顿方向$H_k\nabla f_k$。图5给出了L-BFGS在每一循环中需要保存在内存中的数据，可以看出其内存消耗为$\Theta(mn)$，避免了存储$O(n^2)$的海森矩阵。另外，仔细查看算法1会发现L-BFGS需要的操作仅限于向量的点积与加减操作。不管是基于MPI还是MapReduce，向量的点积与加减操作都可以很容易的并行，所以实现大规模的L-BFGS理论上没有任何难度。

最后请注意，由于这里讲的其实是并行化的L-BFGS，所以现实中任何大规模的可微无约束优化问题都可通过L-BFGS解决。所以，利用L-BFGS求

Figure 6: 优化方法比较

解L2-LR只需要将L2-LR对应的梯度告诉L-BFGS就可以了。但是，由于L1-LR并不可微（$|x|$在$x = 0$处的微分并不存在），所以L-BFGS并不能直接用来解决L1-LR。

最后，图6给出了牛顿方法与DFP/BFGS/L-BFGS这一系列拟牛顿方法的比较。与牛顿方法比，DFP方法将牛顿方法中每循环重新计算海森矩阵改为了每循环中的增量计算。BFGS采纳了DFP的思想，直接增量计算海森矩阵的逆，从而比DFP省去了求逆的运算。最后，L-BFGS又省去了存储海森矩阵或其逆，直接算出了牛顿方向。

5 OWL-QN：用L-BFGS求解L1规则化的逻辑回归

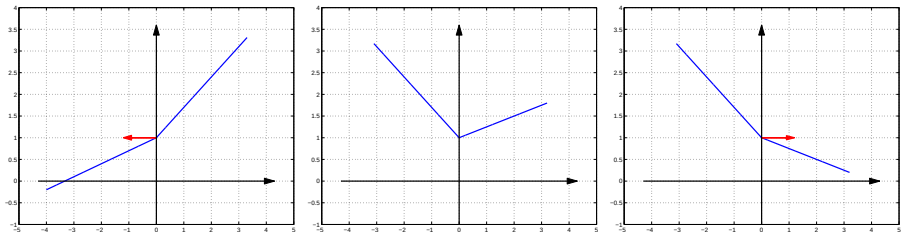
在这部分我们着重讲解如何利用L-BFGS的框架来解决L1-LR问题，这个算法叫Orthant-Wise Limited memory Quasi-Newton method (OWL-QN)，由Andrew Galen和Jianfeng Gao于2007年提出[1]。OWL-QN的主要思想是，给定当前点$\mathbf{x}_k$，当我们把搜索限定在$\mathbf{x}_k$所在的某个特定象限的时候（注意这里说的是“某个”因为当$\mathbf{x}_k$的某几项是0的情况下，会有多个象限包含$\mathbf{x}_k$），$\mathbf{x}_k$各个维度的正负已经确定，所以L1项$|\mathbf{x}_k|$就完全变成了一个线性项。更重要的是，由于L1项变成了线性项，它对二阶导数的计算没有任何影响，所以（拟）牛顿方法中需要的海森矩阵就和L1项没有任何关系了，从而用L-BFGS计算$H_k \nabla f_k$的方法就完全适用。但同时也应该注意到，由于我们不能总在一个象限里面搜索，所以当$\mathbf{x}_k(i) = 0$的时候还得考虑到底是往小于0还是大于0的方向搜索。技术上讲，这个问题是通过所谓的伪梯度(pseudo-gradient)解决的。由于论文[1]在这个方面也没有给出比较详细和直观的论述，我们就在这方面多花费一些笔墨。

对于目标函数

$$f(\mathbf{x}) = l(\mathbf{x}) + \lambda \|\mathbf{x}\|_1, \qquad \mathbf{x} \in \mathbb{R}^n$$

虽然它在任何$\mathbf{x}(i) = 0$的地方都不可微，这并不妨碍我们定义$f(\mathbf{x})$对$\mathbf{x}(i)$的左右微分。具体来讲，$f(\mathbf{x})$关于$\mathbf{x}(i)$的左右微分（分别记为$\partial_i^- f(\mathbf{x}), \partial_i^+ f(\mathbf{x})$）如下

$$\partial_i^{\pm} f(\mathbf{x}) = \frac{\partial}{\partial \mathbf{x}(i)} l(\mathbf{x}) + \begin{cases} \lambda \delta(\mathbf{x}(i)) & if\ \mathbf{x}(i) \neq 0 \\ \pm\lambda & if\ \mathbf{x}(i) = 0 \end{cases},$$



(a) 第一种情况　(b) 第二种情况　(c) 第三种情况

Figure 7: OWL-QN中的伪梯度定义

其实就是把$|x|$的左右微分给代进去而已：不等于0时就是符号函数，等于0时左微分为-1，右微分为+1。论文[1]中直接给出了基于左右微分定义的伪梯度

$$\diamond_i f(\mathbf{x}) = \begin{cases} \partial_i^- f(\mathbf{x}) & if\ \partial_i^- f(\mathbf{x}) > 0 \\ \partial_i^+ f(\mathbf{x}) & if\ \partial_i^+ f(\mathbf{x}) < 0 \\ 0 & otherwise \end{cases}$$

看起来有点不知其所以然的感觉，并且好像前两种情况好像并不是完全互斥的。那么我们下面分析一下这个伪梯度到底定义了什么？首先，我们注意到$\partial_i^- f(\mathbf{x}) \leq \partial_i^+ f(\mathbf{x})$，那么根据左右微分和0的关系，我们可以将其划分为以下三种情况

1. $0 \leq \partial_i^- f(\mathbf{x}) \leq \partial_i^+ f(\mathbf{x})$: 这种情况如图7(a)所示，由于我们要最小化目标函数，所以我们应该往$\mathbf{x}(i) < 0$的方向搜索，所以就应该采用此处的左微分。

2. $\partial_i^- f(\mathbf{x}) \leq 0 \leq \partial_i^+ f(\mathbf{x})$: 这种情况如图7(b)所示，很显然由于$\mathbf{x}(i) = 0$已经是这个维度的最小点了，那么我们就没有必要调整了，所以就定义它的伪梯度为0。

3. $\partial_i^- f(\mathbf{x}) \leq \partial_i^+ f(\mathbf{x}) \leq 0$: 这种情况如图7(c)所示，同样由于我们需要最小化，所以应该往$x(i) > 0$的方向搜索，应该采用此处的右微分。

总结这上面三种情况正是式子 5给出的定义。有了伪梯度作为在每个点的搜索方向，并且知道在给定象限内，二阶导数海森矩阵和L1项无关，我们不难给出如何用L-BFGS来解决L1-LR问题，这就是OWL-QN方法，具体如算法2所示。

关于算法2，有如下几个细节需要注意一下

1. 第4行：由于伪梯度就是在将要搜索象限内的合法梯度，我们就完全用$\diamond f_k$代替L-BFGS中的∇f_k。这个负号就是为了直接算出来牛顿方向$-H_k \diamond f_k$。L-BFGS中算出的$H_k \nabla f_k$是牛顿方向的反方向。这里的这个负号使得最终的搜索方向能以这个负伪梯度方向做投影。

2. 第5行：完全用L-BFGS来寻找牛顿方向。

Algorithm 2 Orthant Wise Limited-memory Quasi-Newton (OWL-QN)

01: choose initial point x_0
02: $S \Leftarrow \{\}, Y \Leftarrow \{\}$
03: **for** $k = 0$ to MaxIters **do**
04: Compute $v_k = -\diamond f(x_k)$
05: Compute $d_k \Leftarrow H_k v_k$ using S and Y
06: $p_k \Leftarrow \pi(d_k; v_k)$
07: Find x_{k+1} with constrained line search
08: **if** termination condition satisfied **then**
09: Stop and return x_{k+1}
10: **end if**
11: Update S with $s_k = x_{k+1} - x_k$
12: Update Y with $y_k = \nabla l(x_{k+1}) - \nabla l(x_k)$
13: **end for**

3. 第6行：由于搜索需要限制在这个特定的象限内，所以牛顿方向d_k被投影在伪(负）梯度v_k方向从而保证搜索不会使得某一维从负变正或相反。

4. 第7行：在线搜索的时候，每一个被搜索到的点也被投影在$\mathbf{x}_k$所在的象限。

5. 第12行：就像前面所讲的，海森矩阵的近似和L1项无关，所以，y只需要记录$l(\mathbf{x})$梯度的变化

如果我们前面已经实现了大规模并行的L-BFGS，实现OWL-QN只需要做上面列出的这些调整。

6 相关工作

在这一部分，我们简要介绍一下和求解LR相关的工作与文献。关于求解L2-LR，一致推荐的读物是Tom Minka（发明Expectation Propagation算法的牛人）在2003年写的一篇并不屑发表的文章[12]，其中对比了当时已有的7种求解L2-LR问题的方法，包括BFGS, FixedHessian, NewtonMethod, ConjugateGradient, CoordinateDescent, ModifiedIterativeScaling 和DualMethod。在Tom所实验的数据集上（几百维的特征，几千个样本），BFGS是表现最好的。由于这些数据集很小，存储海森矩阵的逆根本就不是个事，所以也就不需要L-BFGS了。最近，Chih-Jen Lin研究组的Guo-Xun Yuan等人又做了一个更全面的关于L1规范化的线性分类器（包括LR和SVM）的比较报告[18]，有兴趣的同学可以研读一下。

但是，我们在平时工作中使用的L2-LR通常不是用BFGS或者L-BFGS求解的，为什么呢？因为大神Chih-Jen Lin已经把它写在了估计大家都听说过或用过的LIBLINEAR库里了。LIBLINEAR中的LR采用的是一个叫做Trusted Region Newton (TRON)的方法[8][2]。TRON是Chih-Jen Lin于1999年和另外两

[2] 下载地址：`http://www.mcs.anl.gov/~more/tron/`

位研究人员发明的一种牛顿方法，而非拟牛顿法。Lin在[9]展示了TRON能在单机的情况下比L-BFGS更快的解决L2-LR问题，而这种加速可以归结为它在每一步采用的都是更真实的牛顿方向。它的缺点，虽然作者本身没有明确说明，笔者私下认为（1）TRON有太多的参数需要设定而L-BFGS没有任何参数需要设定。（2）尽管TRON也可以并行化（因为其中需要的只是矩阵和向量的乘法和向量之间的加法），其实现要比L-BFGS复杂并且难维护。（3）TRON很难通过简单扩展来解决L1-LR问题。事实上，[18]中实现了通过$w = w^+ - w^-,\ \ w^+ \geq 0, w^- \geq 0$翻倍变量的方式来使用TRON解决L1-LR，但其运行效率大大低于OWL-QN。效率降低的原因在于它将无约束问题转化为了带两倍于原来变量数的约束问题。

在求解L1-LR方面，由于L1项的不可微性，前期提出了很多特殊的方法用于处理在$x = 0$的情况，其中包括Perkins等人在2003年提出的*grafting*方法[14]，2004年Goodman提出的基于generalized iterative scaling (GIS)的求解方法[4]，和2006年Lee等人所提出的将L1-LR转化为一个re-weighted least square问题的方法[6]。尽管这些方法都可以解决一定规模的L1-LR问题，但是由于其设计的局限性，无法应用到大规模的LR问题上。2007年，斯坦福Stephen Boyd教授及其团队将适用于大规模优化问题的内点法（interior point method）应用于L1-LR，并且用实验结果表明其速度超出了其他凸优化问题解决方法，例如MOSEK[3]。但是，其缺点在于并行化不是很方便，所以2007年Andrew Galen和Jianfeng Gao提出的OWL-QN方法[1]就变成了通常采用的方法。OWL-QN的优点在于（1）它充分考虑了L1的特殊性并且（2）借助L-BFGS到达了很高的通用性与并行性。LIBLINEAR库中的L1-LR是一种基于GLMNET[3]的改进算法[19]，需要的前期知识准备比较多，这里就不再详细介绍了。

对于在单机上试用和使用逻辑回归来说，LIBLINEAR库是不二的选择。但对实际的工程应用，通常数据量太大不能放到一台机器上时，考虑并行化的话，就不要费力想怎么并行化LIBLINEAR库（虽然也是可行的）。直接实现一个通用的并行L-BFGS算法，然后把L2-LR相应的梯度代进去，就是一个并行的L2规则化的逻辑回归。后期如果需要的话，还可以做稍许修改，将其变为并行的OWL-QN，从而用来解决L1规则化的逻辑回归。另外，作为一个副产品，此处实现的并行L-BFGS算法还可以用来解决其他任何可微无约束的优化问题。

7 前沿研究

最后,讨论一下有关LR的扩展内容，主要是一些由于篇幅问题无法展开讨论，但同时又是实际中非常重要的内容。第一是线上算法(online algorithm)。本文作为LR的进阶教程，在算法上主要讲解了批处理模式(batch mode)的学习算法，意思是所有的训练样本在初始时就是给定的，而每一次迭代都是把这些样本当做一批处理后而进行的参数更新。但在现实中的大规模LR应用，批处理模式通常不是最好的选择。拿最常见的广告投放系统来说，由于每天要投放几十亿的广告并能知道哪些投放真正被用户点击了，这相当于每天我们都能收集数十亿的训练样本。虽然我们可以每隔一个特定的时间（例如每隔几周、几天或者几小时之类的）就通过批处理模式更新一下参数，更有效地方式通常是线上模式(online mode)：即每个样本来的时候我

[3] `http://www.mosek.com`

们都相应的更新参数，并且每个样本只处理一次。在这种情况下，L2-LR的训练模式就需要从拟牛顿方法转变为线上梯度下降(Online Gradient Descent (OGD))算法。其基本原理如本文式子10所示，只不过其中的$\nabla f(\mathbf{w}_k)$不再是通过所有样本估计得出，而是通过一个或多个当前所见的样本。但是，直接应用OGD于L1-LR，即便加上了关于L1的伪梯度，也不能产生稀疏的模型（因为步长设定的原因），所以一系列的新的应用线上算法产生稀疏LR模型的算法被提了出来，代表的有FOBOS[2], Truncated Gradient[7], Regularized Dual Averaging (RDA)[17] 和Follow The (Proximally) Regularized Leader (FTRL-Proximal)[10]，而RDA和FTRL-Proximal看起来是现在最好的稀疏模型线上训练算法。

第二，对应线上算法，一个非常重要的trick就是如何设置每一次根据当前看到的训练样本做更新时所用的步长。通常我们要求这个步长是慢慢减小的，从而可以保证模型的最终收敛性。但是，在线上算法的背景下，由于广告的关键词及用户的行为是有其固有的波动性（例如随时间与地点），我们并不一定非得要求一个收敛固定的模型。这样就为如何设置步长提出了新的挑战。在另一方面，由于我们有上十亿的特征维度，这些上亿的维度是否应该应用同样的步长也是一个值得商榷的事情。直觉上来说，由于每个特征取0的概率大不相同，每次参数更新都递减步长对那些常取0值得特征不公平，所以就产生了不同特征应用不同递减步长的做法，并且取得了相当好的效果[16]。

最后，我们在实际应用LR的系统中还要密切关注预测的概率与实际概率的符合程度，即学术上讲的校正（Calibration）。校正在有些只关注排序与分类的系统中一点都不重要，但在有些系统中，例如广告竞价投放系统中，它确是非常的重要：因为我们需要根据LR预估出的CTR来计算一个广告被点击后应该向广告主收多少钱。我们预测出的概率，例如LR中通过逻辑函数给出的0到1之间的值，和实际中观测到的概率之间的差别可能来源于很多我们无从知晓或无法弥补的因子，例如不够全面的特征和与实际不符的数学模型。所以，校正通常是通过学习一个新的回归函数将我们预测值回归到观测到的值。其实，并不是只有LR需要校正，我们常用的支持向量机(SVM)其实是更需要校正的机器学习算法。有兴趣的同学可以从John Platt的关于校正的经典文章[15]读起吧。

作为结尾，笔者推荐Google在今年国际顶级数据挖掘会议SIGKDD2013上发表的一篇关于广告点击率预估系统的文章：Ad Click Prediction: a View from the Trenches [11]。其中对上面提到的三点都有相关阐述，可以作为相关方向入口的指引吧。但是请注意这篇文章中，不知是有意还是无意，其中的公式有相当多的笔误。虽然这些笔误还不至于影响大家了解其主要思想，但还是请大家阅读时注意。最后，请各位领导、大拿、冲在第一线的研究猿与攻城狮们不吝赐教、切磋琢磨、一同进步。谢谢！

References

[1] G. Andrew and J. Gao. Scalable training of l1-regularized log-linear models. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, pages 33–40, New York, NY, USA, 2007. ACM.

[2] J. C. Duchi and Y. Singer. Efficient learning using forward-backward splitting. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *NIPS*, pages 495–503. Curran Associates, Inc., 2009.

[3] J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

[4] J. Goodman. Exponential priors for maximum entropy models. In D. M. Susan Dumais and S. Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 305–312, Boston, Massachusetts, USA, May 2 - May 7 2004. Association for Computational Linguistics.

[5] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning, Second Edition: Data Mining, Inference, and Prediction.* Springer Series in Statistics. Springer, 0002-2009. corr. 3rd edition, Feb. 2009.

[6] S. in Lee, H. Lee, P. Abbeel, and A. Y. Ng. Efficient l1 regularized logistic regression. In *In AAAI*, 2006.

[7] J. Langford, L. Li, and T. Zhang. Sparse online learning via truncated gradient. *J. Mach. Learn. Res.*, 10:777–801, June 2009.

[8] C.-J. Lin and J. J. Moré. Newton's method for large bound-constrained optimization problems. *SIAM J. on Optimization*, 9(4):1100–1127, Apr. 1999.

[9] C.-J. Lin, R. C. Weng, and S. S. Keerthi. Trust region newton method for logistic regression. *Journal of Machine Learning Research*, 9:627–650, 2008.

[10] H. B. McMahan. Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

[11] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, S. Chikkerur, D. Liu, M. Wattenberg, A. M. Hrafnkelsson, T. Boulos, and J. Kubica. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2013.

[12] T. P. Minka. A comparison of numerical optimizers for logistic regression. Technical report, Microsoft Research, 2003.

[13] A. Y. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, pages 78–, New York, NY, USA, 2004. ACM.

[14] S. Perkins and J. Theiler. Online feature selection using grafting. In *In International Conference on Machine Learning*, pages 592–599. ACM Press, 2003.

[15] J. C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *ADVANCES IN LARGE MARGIN CLASSIFIERS*, pages 61–74. MIT Press, 1999.

[16] M. J. Streeter and H. B. McMahan. Less regret via online conditioning. *CoRR*, abs/1002.4862, 2010.

[17] L. Xiao. Dual averaging methods for regularized stochastic learning and online optimization. *J. Mach. Learn. Res.*, 11:2543–2596, Dec. 2010.

[18] G.-X. Yuan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. A comparison of optimization methods and software for large-scale l1-regularized linear classification. *J. Mach. Learn. Res.*, 11:3183–3234, Dec. 2010.

[19] G.-X. Yuan, C.-H. Ho, and C.-J. Lin. An improved glmnet for l1-regularized logistic regression. *Journal of Machine Learning Research*, 13:1999–2030, 2012.