



北京大学

硕士研究生学位论文

题目： GLM+：一个高效的广义线性模型求解系统

姓 名： 王 矜 宇

学 号： 1501214408

院 系： 信息科学技术学院

专 业： 计算机软件与理论

研究方向： 多媒体数据库管理技术

导 师： 崔 斌 教授

二〇一八年六月

版权声明

任何收存和保管本论文各种版本的单位和个人，未经本论文作者同意，不得将本论文转借他人，亦不得随意复制、抄录、拍照或以任何方式传播。否则一旦引起有碍作者著作权之问题，将可能承担法律责任。

摘要

广义线性模型因为其简单与高效的特性被广泛地应用于数据挖掘的各个领域中。常见的广义线性模型包括线性回归、套索回归、支持向量机和逻辑回归。求解广义线性模型有多种优化算法，主流的算法包括随机梯度下降、坐标下降和交替方向乘子法三种。随机梯度下降算法每次针对若干样本沿梯度下降的方向求解极小值；坐标下降算法则每次选择一个维度，在保持其他维度的参数值不变的条件下，优化这一维度的参数，并通过反复迭代求解函数最小化。交替方向乘子法是一种求解优化问题的计算框架，通过分解协调过程，将大的全局问题分解为多个较小、较容易求解的局部子问题，并通过协调子问题的解而得到大的全局问题的解。

目前常用的广义线性模型求解系统，都使用单一的优化算法去求解广义线性模型：比如，Shotgun针对坐标下降法进行了大量优化，从而使用坐标下降法去求解线性模型和套索回归；scikit-learn也使用坐标下降法对套索回归、逻辑回归、支持向量机进行求解。然而，在实际任务中我们发现，使用单一的优化算法并不能够对每种广义线性模型都得到较高的性能和较好的拟合效果。实验表明，由于不同模型之间存在差异，只考虑一种优化算法难以对各种训练任务都得到性能高效的解决方案。为了解决训练任务的多样性对模型求解带来的性能问题，本文提出了一种基于规则的优化算法选择器，用于自动地为不同的模型训练任务选择性能较好的优化算法。

本文首先通过详尽的实验对比对不同优化算法的执行效果，结合执行时间与收敛速度两方面的分析，总结出三条启发式规则。接着我们以此为基础设计了一个基于规则的优化算法选择器，可以根据模型类别，自动地选择较优的优化算法。最后，通过加入参数选择、稀疏优化和计算优化三项工程技术，构建了一个优化求解广义线性算法的原型系统GLM+。本文通过真实数据集的实验，与常用广义线性模型求解系统Shotgun和scikit-learn的实验比较，证明了该系统的有效性。

关键词：广义线性模型,优化算法,算法比较,系统实现

GLM+: An Efficient System for Generalized Linear Models

Lingyu Wang (Computer Software and Theory)

Directed by Prof. Bin Cui

ABSTRACT

Generalized linear models are widely used in data analysts and machine learning, especially in large-scale machine learning because of its simplicity and good performance. Generalized linear model includes regression, like linear regression, lasso and classification, support vector machine and logistic regression. We have a few popular optimization methods to solve them, including stochastic gradient descent(SGD), coordinate descent(CD) and alternating direction method of multipliers(ADMM).

For each iteration, SGD randomly samples one or a mini-batch of data examples and updates the model parameter using the estimated gradients. On the contrary, CD solves the optimization problems by successively performing approximate minimization along the directions of coordinates. For each step, CD solves an univariate optimization problem by fixing the value of other coordinates. ADMM solves convex optimization problems by breaking them into smaller subproblems, each of which is then easier to handle. Commonly used systems for generalized linear model use a single optimization algorithm to solve all kinds of problems. However, experiments show that it is impossible to achieve a cost-effective solution for all kinds of generalized linear models due to the differences between different models. In order to resolve the problem, we propose a rule-based optimization algorithm selector to select the best optimization algorithm automatically.

In this paper, we first broadly review the three commonly used optimization methods, stochastic gradient, coordinate descent, alternating direction method of multipliers, perform some experiments, and then propose a rule-based optimizer to guide the solving. We also design a complete system, GLM+, based on the three rules we proposed with the two engineering techniques, parameter selection and optimization for sparse data. We perform some experiments on real datasets and compare it with the commonly used

machine learning systems, Shotgun and scikit-learn, to verify the effectiveness of our system GLM+.

KEYWORDS: generalized linear model, optimization, analysis and comparison, system implementation

目 录

| | |
|----------------------|----|
| 第一章 引言 | 1 |
| 1.1 主要创新点 | 2 |
| 1.2 论文组织结构 | 2 |
| 第二章 研究背景与预备知识 | 3 |
| 2.1 研究背景 | 3 |
| 2.2 常见广义线性模型 | 4 |
| 2.3 常见优化算法 | 7 |
| 第三章 基于规则的算法选择优化器 | 13 |
| 3.1 算法选择规则 | 13 |
| 3.2 优化算法比较 | 15 |
| 3.3 分析与讨论 | 17 |
| 第四章 系统实现 | 21 |
| 4.1 参数的选择 | 21 |
| 4.2 稀疏数据的执行优化 | 21 |
| 4.3 e^x 函数计算优化 | 22 |
| 4.4 并行的更新策略选择 | 25 |
| 第五章 实验与分析 | 27 |
| 5.1 数据集介绍 | 27 |
| 5.2 与其他系统的对比实验 | 27 |
| 5.3 优化策略实验 | 29 |
| 5.4 参数自动调优实验 | 31 |
| 5.5 稀疏数据的优化实验 | 31 |
| 第六章 总结与展望 | 33 |
| 参考文献 | 35 |
| 致谢 | 39 |
| 攻读硕士期间的研究成果 | 41 |
| 北京大学学位论文原创性声明和使用授权说明 | 43 |

插 图

| | | |
|-----|----------------------------|----|
| 2.1 | 逻辑回归函数曲线 | 6 |
| 2.2 | SVM算法示意图 | 7 |
| 2.3 | 随机梯度下降与坐标下降算法示意图 | 8 |
| 3.1 | GLM+系统框架图 | 13 |
| 4.1 | 针对稀疏数据设计的数据存储结构 | 22 |
| 4.2 | 并行执行时无锁更新 | 26 |
| 5.1 | 三种优化算法性能比较 | 30 |

表 格

| | | |
|-----|--------------------------|----|
| 2.1 | 研究模型的优化函数 | 3 |
| 3.1 | 三种优化算法的收敛速度比较 | 14 |
| 3.2 | 三种优化算法求解模型更新函数 | 19 |
| 4.1 | exp()函数的实验比较 | 25 |
| 5.1 | 实验数据集 | 27 |
| 5.2 | GLM+与其他系统的对比实验 | 28 |
| 5.3 | 参数选择实验结果 | 31 |
| 5.4 | 稀疏数据优化实验结果 | 31 |

算 法

| | | |
|-----|--|----|
| 3.1 | 系统优化求解执行流程($model, X, Y, \rho$) | 14 |
| 3.2 | 随机梯度下降算法(SGD)($f, (X, Y)(X \in R^{N \times m}), \rho, \epsilon$) | 15 |
| 3.3 | 循环坐标算法(CD)($f, (X, Y)(X \in R^{N \times m}), \rho, \epsilon$) | 16 |
| 3.4 | 交替方向乘子法(ADMM)($f, g, A, B, c, \rho, \epsilon$) | 16 |
| 4.1 | exp函数实现(x) | 23 |
| 4.2 | exp函数近似算法1(x) | 24 |
| 4.3 | exp函数近似算法2(x) | 25 |

第一章 引言

广义线性模型是线性模型的扩展，其特点是不改变数据的自然度量，建立响应变量Y的数学期望值与线性组合的预测变量P之间的关系 [1]。线性回归(Linear regression)、套索回归(Lasso)、支持向量机(Support Vector Machine)和逻辑回归(Logistic Regression) 算法是常用的广义线性模型，因为其模型简单并且训练高效被广泛地应用在数据挖掘的各个领域。

线性回归是利用数理统计中的回归分析,来确定两种或两种以上变量间相互依赖的定量关系的一种统计分析方法 [2]。线性回归模型与加入了一阶惩罚项的Lasso算法常被用于作为分析数据集，同时，也被广泛地应用于趋势分析等模型的构建中 [2]。逻辑回归本质是一种线性回归模型，一般用来解决二分类问题，适用于大规模稀疏特征优化，广泛应用于广告推荐、欺诈识别等多种分类实际问题 [3, 4, 5, 6, 7]。支持向量机模型在模式识别、回归估计、概率密度函数估计等方面都有应用；例如，在模式识别方面，对于手写数字识别、语音识别、人脸图像识别、文章分类等应用,支持向量机模型都有很好的表现 [8]。

目前求解广义线性模型的常用优化算法有三种，包括随机梯度下降、坐标下降和交替方向乘子法。其中随机梯度下降算法每次针对若干样本沿梯度下降的方向求解极小值；坐标下降算法则每次选择一个维度，在保持其他维度的参数值不变的条件下，优化这一维度的参数，并通过反复迭代求解函数最小化。交替方向乘子法是一种求解优化问题的计算框架，通过分解协调过程，将大的全局问题分解为多个较小、较容易求解的局部子问题，并通过协调子问题的解而得到大的全局问题的解。

目前常用的广义线性模型求解系统，如Shotgun [9]和scikit-learn [10]，都使用单一的优化算法去求解广义线性模型。比如，Shotgun针对坐标下降法进行了大量优化，从而使用坐标下降法去求解线性模型和套索回归；scikit-learn也使用坐标下降法对套索回归、逻辑回归、支持向量机进行求解。然而，在实际任务中我们发现，使用单一的优化算法并不能够对每种广义线性模型都得到较高的性能和较好的拟合效果。比如，用坐标下降法在求解逻辑回归的过程中，会有大量的指数操作，从而导致较长的执行时间。因此，坐标下降法并不适合与逻辑回归算法的求解。为了对多种广义线性模型进行高效的求解，本文提出了一个新的广义线性模型求解系统——GLM+。

1.1 主要创新点

本文以随机梯度下降、坐标下降和交替方向乘子法三种常用的优化算法求解线性回归、套索回归、支持向量机和逻辑回归四种常见的广义线性模型为研究对象，在详尽的实验对比的基础上，结合执行时间与收敛速度两方面的分析，总结出三条启发式规则。并基于此设计了一个基于规则的优化算法选择器，可以根据模型类别，自动地选择最优的优化算法。最后，通过加入参数选择、稀疏优化两项工程技术，构建了一个优化求解广义线性算法的原型系统GLM+。本文的贡献可以总结如下：

1. 在大量真实数据集的实验基础上，结合对随机梯度下降、坐标下降与交替方向乘子法的执行时间与收敛速度两方面的分析，提出了基于规则的算法选择优化器。
2. 在基于规则的算法选择优化器的基础上加入了参数选择、针对稀疏数据的执行优化两项工程技术构建了一个优化求解广义线性算法的完整系统GLM+。
3. 通过与常用广义线性模型求解系统Shotgun和scikit-learn的实验比较，证明了GLM+系统的实用性与高效性。

1.2 论文组织结构

根据前面介绍的研究内容，本文的组织结构如下：

第一章为引言，首先概括了问题背景和研究意义，接下来介绍研究内容和面临的主要挑战，然后指出本文的主要贡献，最后介绍各章节内容安排。

第二章为研究的相关背景与相关工作介绍，首先介绍了我们研究的目标，广义线性模型的定义以及常见的广义线性模型和优化求解算法，然后系统地介绍了与本文相关的研究工作。

第三章介绍优化基于规则的优化算法选择器，通过对随机梯度下降、坐标下降和交替乘子法进行理论分析和实验对比，总结出三条启发式规则，从而构建了一个算法选择优化器，可以根据模型选择较优的优化算法。

第四章介绍广义线性模型求解系统GLM+的系统实现，从超参数的选择、针对稀疏数据的执行优化、对特定函数 e^x 的实现优化和并行更新策略优化四个方面介绍系统实现上的优化工作。

第五章为实验介绍和结果分析，通过多组实验，验证本文实现的广义线性模型求解系统GLM+的实用性和高效性。

第六章总结全文工作，概括本文主要内容并指出存在的问题与不足，对进一步的研究方向进行展望。

第二章 研究背景与预备知识

2.1 研究背景

在实际使用中，形式简单、计算量小的广义线性模型，诸如Lasso 算法、逻辑回归等算法，被广泛地用于数据分析和机器学习等各个领域，尤其是在大规模数据的分类和回归问题中由于其算法的简单和高效被广泛应用。诸如这样的算法包括线性回归、Lasso、逻辑回归和支持向量机等算法。我们研究的模型包括线性回归算法、Lasso两种回归算法和逻辑回归、支持向量机两种分类算法。给定数据集 $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$ 其中 $x_i = (x_{i1}; x_{i2}; \dots; x_{id})$, $y \in R$. 我们的目标是优化函数 $f(w) + g(w)$ ，其中 $f(w)$ 用于优化模型，记录与优化模型的预测值 \hat{y} 与真实值 y 的差异； $g(w)$ 是惩罚函数，用于控制模型的复杂程度，用于防止模型过拟合和降低预测方差。常用的惩罚函数有 l_1 形式的 $g(w) = \|w\|_1$ 和 l_2 形式的 $g(w) = \|w\|_2^2$.我们研究的四种模型的优化函数如表2.1所示：

表 2.1 研究模型的优化函数

| 优化模型 | $f(w)$ | $g(w)$ |
|---------|--|--------------------------|
| 线性回归 | $f(w) = \ y - wx\ _2^2$ | $g(w) = 0$ |
| Lasso回归 | $f(w) = \ y - wx\ _2^2$ | $g(w) = \lambda \ w\ _1$ |
| 逻辑回归算法 | $f(w) = \sum_{i=1}^n \log(1 + e^{-y_i w^T x_i})$ | $g(w) = \lambda \ w\ _1$ |
| 支持向量机 | $f(w) = \sum_{i=1}^n \max\{0, 1 - y_i w^T x_i\}$ | $g(w) = \ w\ _2^2$ |

同时，针对求解这类问题存在若干种优化算法，如随机梯度下降、坐标下降、交替方向乘子算法等各种优化算法。它们虽然原理各有不同，但都可以被用于求解优化模型。随机梯度下降，属于梯度下降的一种，适用于大规模的问题。由于梯度下降法收敛速度慢，其通过顺序地计算对于每一条数据的梯度并以此更新参数值来进行优化，通过梯度的无偏估计代替梯度。坐标下降算法与随机梯度下降算法类似，随机梯度下降算法是逐条数据扫描并优化，而坐标下降算法是逐个特征维度地优化，其在假设其他维度固定的条件下进行单维度的优化。而交替方向乘子算法是一种求解优化问题的计算框架，适用于求解分布式凸优化问题，特别是统计学习问题。ADM通过分解协调（Decomposition-Coordination）过程，将大的全局问题分解为多个较小、较容易求解的局部子问题，并通过协调子问题的解而得到大的全局问题的解。

上述这些方法虽然被广泛使用，但目前并没有关于常见优化算法优化求解模型比较，因此在优化求解模型选择优化算法时往往没有指导，仅凭经验进行选择，并没有系统、完善的比较与研究。而我们的研究就是为了填补这个空白，发现能否针对某个特定模型，是否存在最优的优化算法的选择，使得能在尽可能快的时间能优化求解模型，使其达到一定的收敛水平。

2.2 常见广义线性模型

本节首先介绍广义线性模型的定义，然后简单介绍常见的广义线性模型：线性回归、Lasso回归、逻辑回归算法和SVM算法。

普通线性模型 (ordinary linear model) 可以用下式表示：

$$Y = w_0 + w_1x_1 + w_2x_2 + \dots + w_{p-1}x_{p-1} + \epsilon \quad (2.1)$$

这里 $w_i, i = 1, \dots, p-1$ 为未知参数， w_0 为截距项。其中，需要满足以下假设：

1. 响应变量 Y 和误差项 ϵ 正态性：响应变量 Y 和误差项 ϵ 服从正态分布，且 ϵ 是一个白噪声过程，因而具有零均值，同方差的特性。
2. 预测量 x_i 和未知参数 w_i 的非随机性：预测量 x_i 具有非随机性、可测且不存在测量误差；未知参数 w_i 认为是未知但不具随机性的常数。
3. 研究对象：普通线性模型主要研究响应变量的均值 $E[Y]$ 。
4. 联接方式：在以上三点假设下，对上式两边取数学期望，可得：

$$E[Y] = w_0 + w_1x_1 + w_2x_2 + \dots + w_{p-1}x_{p-1} = \mathbf{wX} \quad (2.2)$$

可以得到，响应变量的均值 $E[Y]$ 与预测量的线性组合 $w_0 + w_1x_1 + w_2x_2 + \dots + w_{p-1}x_{p-1}(\mathbf{wX})$ 通过恒等式 (identity) 联接，即联接函数 $f(x)$ 为恒等函数：

$$E[Y] = f(\mathbf{wX}) = w_0 + w_1x_1 + w_2x_2 + \dots + w_{p-1}x_{p-1} = \mathbf{wX} \quad (2.3)$$

广义线性模型 (generalized linear model) 正是在普通线性模型的基础上，将上述四点模型假设进行推广而得出的应用范围更广，更具实用性的回归模型：

1. 响应变量的分布推广至指数分散族 (exponential dispersion family)：比如正态分布、泊松分布、二项分布、负二项分布、伽玛分布、逆高斯分布。
2. 预测量 x_i 和未知参数 w_i 的非随机性（与普通线性模型相同）：预测量 x_i 具有非随机性、可测且不存在测量误差；未知参数 w_i 认为是未知但不具随机性的常数。

3. 研究对象：广义线性模型的主要研究对象仍然是响应变量的均值 $E[Y]$ 。
4. 联接方式：广义线性模型里采用的联接函数 (link function) 理论上可以是任意的，而不再局限于恒等函数 $f(x) = x$ 。

常见的广义线性模型有线性回归、Lasso回归、逻辑回归算法和SVM算法等，下面对上述算法做简要介绍。

2.2.1 线性回归

给定训练样本 $(x_i, y_i), i = 1, \dots, N$, 其中 $x_i \in R^d, y_i \in R$. 我们的目标是尽可能最小化代价函数 $Q(w)$ ，并将得到的 w 作为参数。不同的模型的优化函数的形式不同。

线性回归是一种回归模型，对一个或多个自变量和因变量之间关系进行建模。模型假设因变量 y 与自变量 x 间的关系为线性，模型可以用下式表示：

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_{p-1} x_{p-1} + \epsilon \quad (2.4)$$

并基于此假设，优化函数：

$$Q_{linear \ regression}(w) = \sum_{i=1}^N (y_i - w^T x_i)^2 \quad (2.5)$$

2.2.2 Lasso回归

对于训练数据特征维度较高、噪声较大的情况，直接应用线性回归模型往往会造成训练模型的方差比较大，因此Lasso算法通过向线性回归代价函数中加入一阶正则项的惩罚项来控制模型复杂度，以得到一个稀疏解。其中 λ 是正则化参数， λ 越大，得到的参数越稀疏。

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_{p-1} x_{p-1} + \epsilon \quad (2.6)$$

$$Q_{lasso \ regression}(w) = \sum_{i=1}^N (y_i - w^T x_i)^2 + \lambda \|w\|_1 (\lambda > 0) \quad (2.7)$$

Lasso方法是一种压缩估计。它通过构造一个惩罚函数得到一个较为精炼的模型，使得它压缩一些系数，同时设定一些系数为零。

2.2.3 逻辑回归算法

对于分类问题，训练样本为 $(x_i, y_i), i = 1, \dots, N$ 其中 $x_i \in R^d, y_i \in 1, 2, \dots, M$. 常用的回归算法有逻辑回归算法和SVM算法。逻辑回归算法常用于二分类问题。逻辑回归假设因变量 y 服从伯努利分布，即联接函数为 $wX = \ln(Y/(1 - Y))$ 。

逻辑回归的拟合函数形式如下：

$$Y = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2 + \dots + w_{p-1} x_{p-1})}} \quad (2.8)$$

即Sigmoid函数，其函数曲线如下：

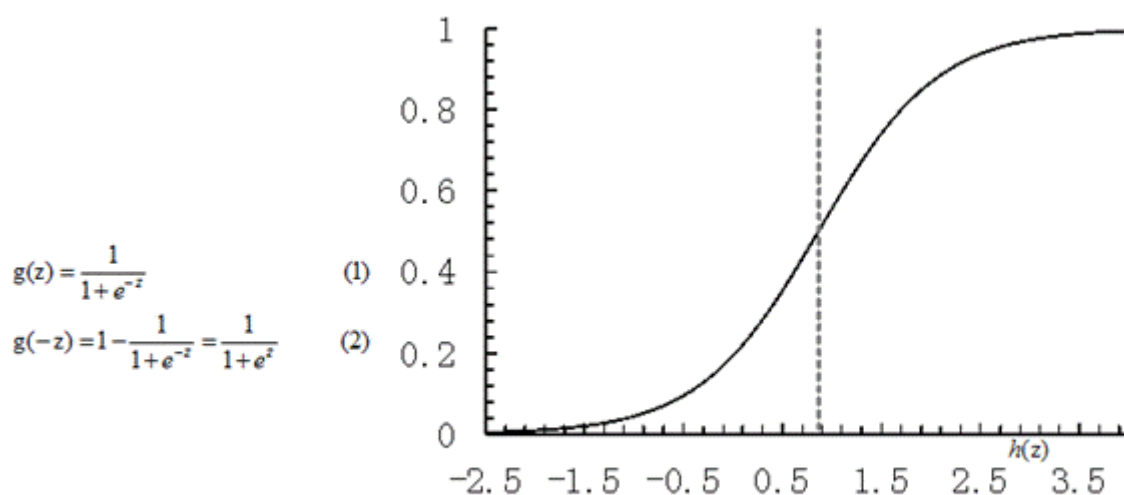


图 2.1 逻辑回归函数曲线

其代价函数为：

$$Q_{logistic}(w) = \sum_{i=1}^N \log(1 + e^{-y_i w^T x_i}) + \lambda \|w\|_1 (\lambda > 0) \quad (2.9)$$

2.2.4 支持向量机算法

支持向量机也是一种分类模型，线性支持向量机基本模型定义为特征空间上的间隔最大的线性分类器。以二分类算法为例，如果定义两种类别的标签为1和-1，那么支持向量机算法的模型为

$$f(x) = \text{sign}(w_0 + w_1 x_1 + w_2 x_2 + \dots + w_{p-1} x_{p-1}) \quad (2.10)$$

代价函数为：

$$Q_{SVM}(w) = \sum_{i=1}^N \max(0, 1 - y_i w^T x_i) + \lambda \|w\|_2^2 (\lambda > 0) \quad (2.11)$$

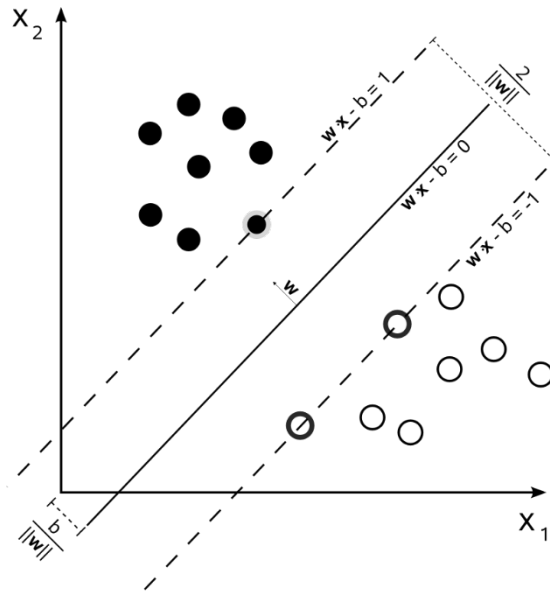


图 2.2 SVM算法示意图

2.3 常见优化算法

由于很多模型在求解优化时并不能得到解析解或解析解的求解过程十分复杂，因此通常使用迭代优化的算法迭代地优化求解。常见的迭代优化求解算法有随机梯度下降算法 [11] (Stochastic Gradient Descent, SGD)、坐标下降算法 [12] (Coordinate Descent, CD) 和交替方向乘子法 [13] (Alternating Direction Method of Multipliers, ADMM)。

本节将简单介绍随机梯度下降、坐标下降以及交替方向乘子算法的原理与更新公式。给定数据集 $D = (x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, 其中 $x_i = (x_{i1}; x_{i2}; \dots; x_{id})$, $y \in R$ 。优化函数为

$$Q(w) = \frac{1}{n} \sum_{i=1}^n Q_i(w) \quad (2.12)$$

其中 w 是 $Q(w)$ 需要优化的参数。 $Q(w)$ 公式中求和中的每一个函数 $Q_i(w)$ 与数据集中每一个样本一一对应。

2.3.1 随机梯度下降

为了使用梯度下降的方法找到函数的最小值，需要沿着沿梯度下降的方向求解极小值。如果实值函数 $Q(x)$ 在点 w 附近可微且有定义，那么我们可以通过梯度下降的公

式来最小化函数值：

$$w := w - \eta \nabla Q(w) = w - \eta \nabla E[Q(w)] = w - \eta \sum_{i=1}^n Q_i(w) \quad (2.13)$$

其中 η 是学习的步长。但梯度下降法收敛速度慢，我们可以使用随机梯度下降来解决这一问题。与梯度下降不同，随机梯度下降根据某个单独样例的误差增量计算权值更新，得到近似的梯度，并根据近似的梯度进行更新。在迭代所有训练样例时，这些权值迭代更新的给出了对于原来误差函数的梯度下降的一个合理近似。

$$w := w - \eta \nabla Q(w; x^{(i)}, y^{(i)}) = w - \eta \nabla Q_i(w) \quad (2.14)$$

已有理论证明 [14]，SGD能够得到很好的结果，且具有很好的收敛速度。同时，在随

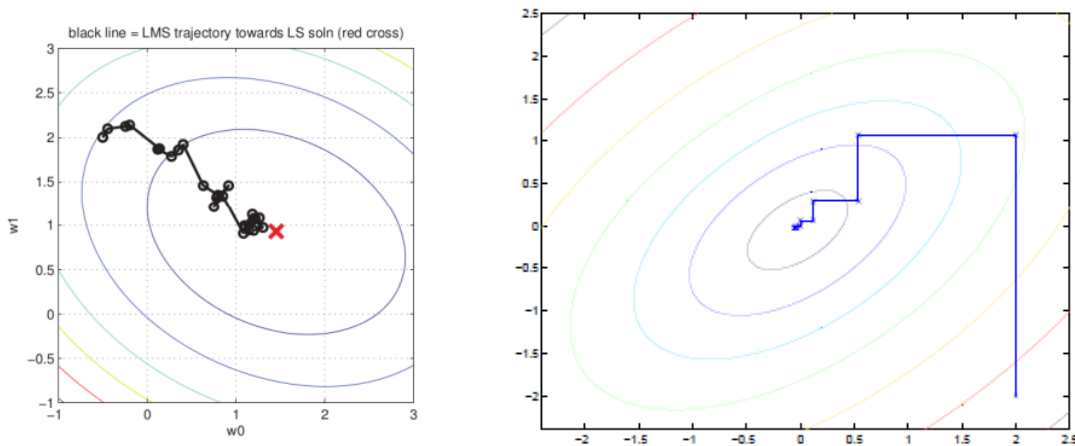


图 2.3 随机梯度下降与坐标下降算法示意图

机梯度算法的基础上，为了选择合适的学习率，出现引入动量的随机梯度下降算法 [15, 16]与Adagrad、Adadelta、adam等自适应学习率算法 [17, 18, 19, 20]。

2.3.2 坐标下降算法

坐标下降优化方法 [21, 22]是一种非梯度优化算法。与随机梯度下降不同，坐标下降法在每次迭代中在当前点处沿一个坐标方向进行一维搜索，固定其他的坐标方向，找到一个函数的局部极小值。在整个过程中依次循环使用不同的坐标方向进行迭代，一个周期的一维搜索迭代过程相当于一个梯度迭代。

（随机）梯度方法是利用目标函数的导数（梯度）来确定搜索方向的，该梯度方向可能不与任何坐标轴平行。而坐标下降法是利用当前坐标方向进行搜索，不需要求

目标函数的导数，只按照某一坐标方向进行搜索最小值。其首先确定一个初始值 w^0 ，重复执行 $k = 1, 2, 3, \dots$ 直到收敛。算法的逻辑如下：

$$\begin{aligned} w_1^k &:= \arg \min_{w_1} Q(w_1, w_2^{k-1}, \dots, w_n^{k-1}) \\ w_2^k &:= \arg \min_{w_2} aQ(w_1^k, w_2, \dots, w_n^{k-1}) \\ &\dots \\ w_n^k &:= \arg \min_{w_n} aQ(w_1^k, w_2^k, \dots, w_n) \end{aligned}$$

2.3.3 交替方向乘子法

交替方向乘子法是一个旨在将对偶上升法的可分解性和乘子法的上界收敛属性融合在一起的算法。它收敛到一个高的精度要求很多次迭代；但几十次迭代就可以达到一个合理的精度，不过可以和其他算法组合来产生一个高的精度。

在介绍ADMM之前我们首先介绍两种优化算法：对偶上升法(Dual Ascent) 和对偶分解法(Dual Decomposition)。

对偶上升法与对偶分解法

有如下优化问题：

$$\min f(x) \quad s.t. Ax = b \quad (2.15)$$

它的拉格朗日形式为：

$$L(x, \lambda) = f(x) + \lambda^T (Ax - b) \quad (2.16)$$

其中 $\lambda > 0$, 即 λ 的每个元素都大于0。对偶形式为：

$$g(\lambda) = \inf_x L(x, \lambda) = -f^*(-A^T \lambda) - b^T \lambda \quad (2.17)$$

其中 f^* 是 f 的共轭函数。

对偶问题为：

$$\max_{\lambda} g(\lambda) \quad (2.18)$$

对偶上升法的迭代更新为：

$$\begin{aligned} x(k+1) &= \arg \min_x L(x, \lambda^k) \quad (\text{x-最小化}) \\ \lambda^{k+1} &= \lambda^k + \alpha^k ((Ax^{k+1} - b)) \quad (\text{对偶变量更新}) \end{aligned} \quad (2.19)$$

其中 α^k 是步长。

假设目标函数是可以分解的，即

$$f(x) = \sum_{i=1}^N f_i(x_i) \quad (2.20)$$

因此，拉格朗日函数可以改写为：

$$L(x, \lambda) = \sum_{i=1}^N L_i(x_i, \lambda) = \sum_{i=1}^N (f_i(x_i) + \lambda^T A_i x_i - \frac{1}{N} \lambda^T b) \quad (2.21)$$

所以对偶分解法的迭代更新为：

$$\begin{aligned} x_i^{k+1} &= \arg \min_{x_i} L_i(x_i, \lambda^k) \quad (\text{x-最小化}) \\ \lambda^{k+1} &= \lambda^k + \alpha^k (Ax^{k+1} - b) \quad (\text{对偶变量更新}) \end{aligned} \quad (2.22)$$

增广拉格朗日和乘子法

为了增加对偶上升法的鲁棒性和放松函数 f 的强凸约束，我们引入增广拉格朗日(Augmented Lagrangians)形式：

$$L_\rho(x, \lambda) = f(x) + \lambda^T (Ax - b) + \frac{\rho}{2} \|Ax - b\|_2^2 \quad (2.23)$$

其中惩罚因子 $\rho > 0$ 。与公式2.16相比，公式2.23只是增加了一个惩罚项，乘子法对应的迭代公式为：

$$x_{k+1} = \arg \min_x L_\rho(x, \lambda^k) \lambda^{k+1} = \lambda^k + \rho(Ax^{k+1} - b) \quad (2.24)$$

将拉格朗日应用于对偶上升法可以极大地增加它的收敛属性，但是它要求一些代价。当 f 可以分解，而拉格朗日 L_ρ 不能分解的，因此 2.24式不能对每个 x_i 并行最小化。这意味着乘子法不能被用来分解。

交替方向算子法

假设有如下优化问题：

$$\min f(x) + g(z) \quad s.t. Ax + Bz = c \quad (2.25)$$

其中 $w \in R^p, z \in R^q$, 同时常数 $A \in R^{r \times p}, B \in R^{r \times p}, c \in R^r$.

如同乘子法中一样, 我们获得它的增广拉格朗日形式为:

$$L_\rho(x, z, \lambda) = f(x) + g(z) + \lambda^T (Ax + Bz - c) + (\rho/2) \|Ax + Bz - c\|_2^2 \quad (2.26)$$

其中, λ 是对偶变量, $\rho > 0$ 是惩罚参数.

那么它的迭代方式为:

$$\begin{aligned} x_{k+1} &= \arg \min_x L_\rho(x, z_k, \lambda_k) \\ z_{k+1} &= \arg \min_z L_\rho(x_{k+1}, z, \lambda_k) \\ \lambda_{k+1} &= \lambda^k + \rho(Ax_{k+1} + Bz_{k+1} - c) \end{aligned} \quad (2.27)$$

其中增广拉格朗日参数 $\rho > 0$. 同时ADMM算法其本身是一个并行算法, 算法步骤见算法3.4, 其中的3-5步可以通过各种优化算法实现。

第三章 基于规则的算法选择优化器

面对优化求解常见广义线性模型难以选择最优优化算法的问题，本文设计并实现了一个集成了多种优化求解方法的广义线性模型求解系统GLM+(Generalized Linear Model+)。系统的框架如图3.1所示，主要分为优化选择器、参数选择和迭代执行三个模块。其中，优化选择器根据规则确定要使用的优化求解方法；参数选择模块负责对算法需要的超参数进行自动选择；迭代执行模块是主体部分，在确定了求解模型与优化方法后，负责迭代求解问题直到收敛，具体算法的执行流程见算法3.1。

GLM+系统运行求解的顺序如下：首先，当给定输入的训练数据及选择需要拟合的广义线性模型后，我们首先得到训练数据集的统计信息，并根据基于规则的优化器得到训练模型使用的优化方法，然后通过采样的方法对训练数据集进行采样，抽取部分训练集数据作为参数选择的数据集，通过参数选择模块从可能的参数值中选择合适的训练所需的参数值。最后使用优化器选择的优化方法及参数选择得到的参数，在迭代执行模块进行模型训练，并在每轮迭代计算结束时判断是否算法已经收敛，如果算法满足收敛条件，则停止优化；否则继续迭代直至收敛。

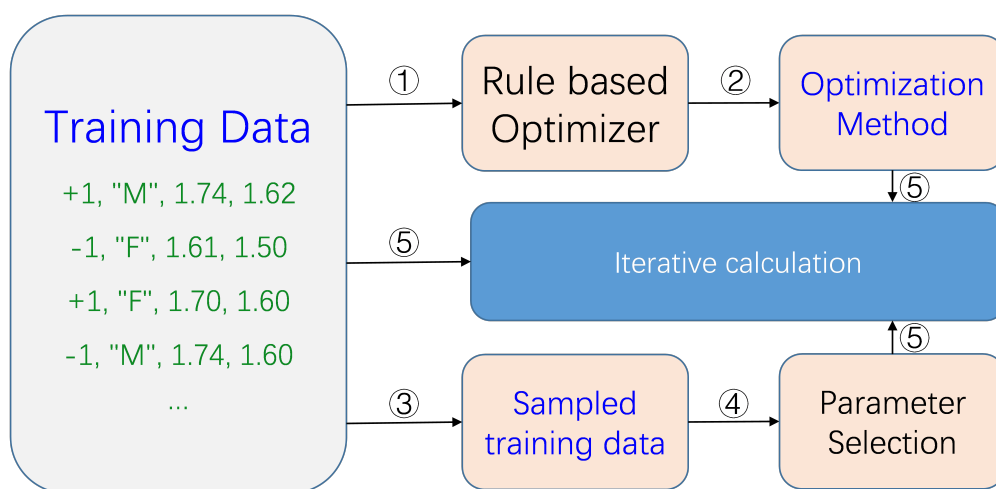


图 3.1 GLM+系统框架图

3.1 算法选择规则

如第1章所介绍和本文实验所验证，随机梯度下降、坐标下降算法及交替方向乘法三种优化算法由于优化求解的原理不同，即使在同一数据集上优化同一模型，三种算法的表现也不相同。我们的目标就是给定输入为某一广义线性模型时，找到求解

算法 3.1 系统优化求解执行流程($model, X, Y, \rho$)

输入: 求解模型 $model \in \{linear, lasso, logistic, SVM\}$, 训练集 (X, Y) , 惩罚参数 ρ , 收敛精度 ϵ

输出: 模型参数 w

```

1: 初始化  $w \leftarrow w_0, k \leftarrow 0$ ; 根据求解模型选择优化方法  $f(model, X, Y, \rho; w)$ 
2: while True do
3:    $w_{k+1} = f(model, X, Y, \rho; w_k)$ 
4:   if  $\|w_{k+1} - w_k\| < \epsilon$  then
5:     break;
6:   end if
7:    $k \leftarrow k + 1$ 
8: end while
9: return  $w_k$ 
    
```

性能最好的优化算法。

三种优化算法求解四种常见的广义线性模型的更新函数如表3.2所示。三种优化算法求解四种常见的广义线性模型的收敛速度有差别, 已有研究工作 [14, 23, 24, 25, 26, 27, 28] 针对三种算法求解不同形式的问题的收敛速度进行了大量研究, 三种优化算法求解广义线性问题的收敛速度比较见表3.1所示。

表 3.1 三种优化算法的收敛速度比较

| | ADMM | CD | SGD |
|---------|------------------------------|--------------------------------|-----------------------------|
| 线性回归 | $O(\ln(\frac{1}{\epsilon}))$ | $O(\ln(\frac{1}{\epsilon}))$ | $O((\frac{1}{\epsilon})^*)$ |
| lasso回归 | $O(\ln(\frac{1}{\epsilon}))$ | $O((\frac{1}{\epsilon}))$ | - |
| 逻辑回归 | $O((\frac{1}{\epsilon}))$ | $O((\frac{1}{\epsilon}))$ | - |
| 支持向量机 | $O((\frac{1}{\epsilon}))$ | $O((\ln(\frac{1}{\epsilon})))$ | - |

* SGD的收敛速度是解在迭代 $O(\frac{1}{\epsilon})$ 样本后达到 ϵ -精度..

SGD跟CD能在相似的迭代轮数后达到收敛 [29]。当然, 理论分析的收敛速度是渐进意义上比较, 对于具体的数据集而言, 数据集的数据倾斜程度、具体的特征数值等都会影响收敛的轮数。优化算法需要迭代到达收敛所需的轮数可能千差万别, 而且无法通过数据集的特征直观的判断出收敛所需轮数。

算法执行的总时间近似等于收敛所需迭代轮数与每轮执行时间的乘积, 由于判断算法收敛速度比较困难; 同时不同优化方法的求解原理不同, 导致不同优化方法的执行时间也有所差别, 难以结合二者在理论上进行分析。因此, 我们通过真实实验的方式比较三种优化方法求解不同模型的求解性能, 发现在求解线性回归、Lasso算法时, CD的性能优于SGD和ADMM; 在求解SVM算法时, SGD的表现最好; 在求解逻辑回

归算法时，ADMM能在最短的时间内收敛。

根据实验结果我们得到了以下三条规则：

规则1 模型输入为线性回归、套索回归时，优化算法选择坐标下降算法。

规则2 模型输入为支持向量机时，优化算法选择随机梯度下降算法。

规则3 模型输入为逻辑回归时，优化算法选择交替方向乘法。

3.2 优化算法比较

本小节将介绍随机梯度下降算法、坐标下降算法和交替方向乘法，并从理论上分析比较它们的异同。

如前述介绍，梯度下降算法是基于一阶梯度的优化方法，其通过计算数据集 X 在 w_k 轮迭代的参数附近的局部梯度信息，以及预设的学习速率 η ，通过随机梯度下降算法计算。为了弥补梯度下降算法每轮迭代需要全量的梯度计算，时间代价比较大，研究人员又提出了随机梯度下降算法，即按照一定的规则，每次选取部分的训练集数据进行梯度的计算和模型参数的更新。与梯度下降算法相比，随机梯度下降算法由于选取的是部分训练集的数据，会引入数据噪声，但执行速度上会比梯度下降算法快很多，算法运行的步骤见算法3.2。

算法 3.2 随机梯度下降算法(SGD)($f, (X, Y)(X \in R^{N \times m}), \rho, \epsilon$)

输入：目标函数定义 f , 训练集 (X, Y) , 惩罚参数 ρ , 收敛精度 ϵ

输出：模型参数 w

```

1: 初始化 $w_0, k \leftarrow 0$ 
2: while True do
3:   for  $i = 0$  to  $N$  do
4:      $w_{k+1} \leftarrow w_k - \eta \nabla f(w_k; x_i, y_i)$ 
5:      $k \leftarrow k + 1$ 
6:     if  $\Delta w < \epsilon$  then
7:       break;
8:     end if
9:   end for
10: end while
11: return  $w_k$ 
```

同样，为了解决梯度下降算法中，对于较大的训练集而言，求全梯度需要占用大量的内存空间，同时时间消耗比较多的问题，坐标下降算法求解目标函数的子函数。子函数问题可以看做选取自变量坐标系中某一个或某一组维度，保持其他的维度值不变的情况下，一次只优化选定维度参数的目标函数。理想的坐标下降算法在选取优化维度时同时考虑迭代消耗的时间与目标函数值下降的相对大小，最理想的选取方

法是每次目标函数值下降的最大方向，但选取优化维度本身可以需要一定的代价。因此，在实际应用中，常用的两种坐标下降的方式是循环坐标下降和随机坐标下降。循环坐标下降方法按照固定顺序对所有维度依次进行迭代计算，算法运行的步骤见算法3.3,而随机坐标下降方法则通过随机的方法选取坐标维度。

算法 3.3 循环坐标算法(CD)($f, (X, Y)(X \in R^{N \times m}), \rho, \epsilon$)

输入: 目标函数定义 f , 训练集 (X, Y) , 惩罚参数 ρ , 收敛精度 ϵ

输出: 模型参数 w

```

1: 初始化 $w_0$ 
2: while True do
3:   for  $i = 0$  to  $M$  do
4:      $w_i \leftarrow \arg \min_{x_i} f(w_1, \dots, w_{i-1}, w_{i+1}, w_M; X, Y)$ 
5:     if  $\Delta w < \epsilon$  then
6:       break;
7:     end if
8:   end for
9: end while
10: return  $w$ 

```

交替方向乘子法可以看做是在增广拉朗格朗日算法基础上发展的算法，混合了对偶上升算法的可分解性和多乘子法的算法优越的收敛性，融合了对偶分解和增广Lagrangian的优点，通过分解协调过程，将大的全局问题分解为多个较小、较容易求解的局部子问题，并通过协调子问题的解而得到大的全局问题的解，算法的运行步骤见算法3.4。

算法 3.4 交替方向乘子法(ADMM)($f, g, A, B, c, \rho, \epsilon$)

输入: 函数定义 f, g , 矩阵 A, B , 向量 c , 惩罚参数 ρ , 收敛精度 ϵ

输出: 向量 x, z, λ

```

1: 初始化 $x_0, z_0, \lambda_0, u \leftarrow \frac{1}{\rho}\lambda, k \leftarrow 0$ 
2: while True do
3:    $x_{k+1} = \arg \min_x (f(x) + \frac{\rho}{2} \|Ax + Bz_k - c + u_k\|_2^2)$ 
4:    $z_{k+1} = \arg \min_z (g(z) + \frac{\rho}{2} \|Ax_{k+1} + Bz - c + u_k\|_2^2)$ 
5:    $u_{k+1} = u_k + Ax_{k+1} + Bz_{k+1} - c$ 
6:    $k \leftarrow k + 1$ 
7:   if  $\Delta x < \epsilon$  then
8:     break;
9:   end if
10: end while
11: return  $x_k, z_k, \rho u$ 

```

可以看到，随机梯度下降算法和坐标下降法都可以通过计算梯度值，更新参数的方法优化求解模型，但二者从两个不同的角度求解，随机梯度下降算法将样本集的各

个样本分裂，每次计算只计算少数样本的梯度值并更新参数；而坐标下降法则是从样本集特征的维度，一次迭代优化更新若干特征维度。同时从算法3.2和3.3比较也可以看出，随机梯度下降算法的求解模式比较固定，计算梯度值后更新参数即可。而针对不同的模型，坐标下降法的求解差别可能很大，这是对于不同的模型而言，将不同的多维度的原问题转换为单一维度的子问题的难度不同，具体的求解算法也有所区别，这也导致了不同模型的求解难以用统一、一致的框架去衡量比较。

交替方向乘子法则与随机梯度下降和坐标下降算法区别较大，它将约束添加到目标函数中，从而转换为一系列子问题进行求解，最终逼近最优解。从在算法3.4中也可以看到，算法执行的3-5步可以由不同的优化算法实现，因此随机梯度下降算法和坐标下降算法也可以与交替方向乘子法结合起来求解模型。

3.3 分析与讨论

优化算法执行的总时间与算法的执行时间和收敛速度都有关。在本小节，我们将结合3.1得到的实验结果，根据优化算法的执行时间与收敛速度进行定性分析。

首先考虑三种优化算法的执行时间。不是一般性地，参与训练的训练集 $D = (x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, 其中 $x_i = (x_{i1}; x_{i2}; \dots; x_{id})$, $y \in R$ 。优化函数为

$$Q(w) = \frac{1}{n} \sum_{i=1}^n Q_i(w) \quad (3.1)$$

其中 w 是 $Q(w)$ 需要优化的参数。 $Q(w)$ 公式中求和中的每一个函数 $Q_i(w)$ 与数据集中每一个样本一一对应。训练数据集的样本数为 n ，每个样本的特征有 d 个维度，训练数据集共有 nnz 个非零特征值。

那么对于随机梯度下降算法而言，算法每次针对若干样本，计算梯度的值并沿着梯度下降的方向求解极小值。因此，对于随机梯度下降算法而言，每次获取样本的特征值 x_i 和标签/真实值 y_i 并计算梯度的值并更新模型的参数值，如果我们将完整遍历样本集一遍作为一次完整的迭代，那么一次完整的随机梯度下降算法迭代所需的时间为：

$$t_{SGD} = \alpha_{SGD} \times nnz \quad (3.2)$$

类似地，坐标下降法是利用当前坐标方向进行搜索，计算在该维度 $x_{:i}$ 上的更新值 δw_i ，将遍历样本集所有特征维度作为一次完整的迭代，那么一次完整的坐标下降算法迭代所需的时间为：

$$t_{CD} = \alpha_{CD} \times nnz + \beta_{CD} \times d \quad (3.3)$$

类似地，对于交替方向乘子法，其算法如下算法3.4。

因此可以得到，交替方向乘子法一轮迭代的时间也可以表示为：

$$t_{ADMM} = \alpha_{ADMM} \times nnz + \beta_{ADMM} \times d \quad (3.4)$$

下面就求解上文提到的四种常见的广义线性模型进行更为具体的分析。

由于交替方向乘子法除了需要更新参数 x 外，还需要在4, 5两步更新 z, u 两个辅助变量。当交替方向乘子法内部使用随机梯度下降更新参数值 w 时，第3步所需时间与随机梯度下降算法类似。因此，交替方向乘子法需要的总的执行时间要高于随机梯度下降算法，因此可以得到 $\alpha_{ADMM} > \alpha_{SGD}$ 。

进一步分析三种优化算法的优化原理可以得到，在求解线性回归、Lasso和SVM算法时，坐标下降算法的更新公式比较简单，执行时间比较短，可以得到 $\alpha_{SGD} > \alpha_{CD}$ ；故有 $\alpha_{ADMM} > \alpha_{SGD} > \alpha_{CD}$ 。

类似地，在求解逻辑回归算法时，由于坐标下降的更新计算过程中指数计算很多，十分耗时，因此坐标下降法的执行时间高于随机梯度下降算法和交替方向乘子法，因此有 $\alpha_{CD} \gg \alpha_{ADMM} > \alpha_{SGD}$ 。

同时结合实际实验的算法收敛效果考虑，发现在求解支持向量机模型时，虽然同样迭代轮数的条件下，坐标下降算法求解的执行时间要少于随机梯度下降和交替方向乘子法，但随机梯度下降算法的收敛效果更好，因此综合考虑求解算法的执行时间和收敛速度，应该选择随机梯度下降算法求解支持向量机。求解逻辑回归算法时交替方向乘子法和随机梯度下降算法的执行时间相差较小，综合考虑，选择收敛效果比较好的交替方向乘子法求解逻辑回归算法。

在大量真实数据集的实验基础上，结合执行时间与收敛速度两方面的分析与说明，我们提出并使用以上三条规则指导广义线性模型的优化求解。

表 3.2 三种优化算法求解模型更新函数

| 模型 | 随机梯度下降 | 坐标下降 | 交替方向乘法 |
|---------|---|--|--|
| 线性回归 | $w \leftarrow w + \gamma_t(y_t - w^T \phi(x_t))\phi(x_t)$ | $w_i = \frac{A_i^T(y - A_{-i}w_{-i})}{A_i^T A_i}$ | $w_i \leftarrow \arg \min_{w_i} (\frac{1}{2} \ w_i \phi(x_i) - y_i\ ^2$ $+ \frac{\rho}{2} \ w_i - z + u_i\ ^2)$ $z \leftarrow \bar{w} + \bar{u}$ $u_i \leftarrow u_i + w_i - z$ |
| Lasso回归 | $u_i \leftarrow [u_i - \gamma_t(\lambda - (y_t - w^T \phi(x_t))\phi_i(x_t))]_+^a$ $v_i \leftarrow [v_i - \gamma_t(\lambda + (y_t - w^T \phi(x_t))\phi_i(x_t))]_+^a$ $w = (u_1 - v_1, \dots, u_d - v_d)$ 其中 $i \in [1, \dots, d]$ | $w_i = S_{\frac{\lambda}{\ A_i\ ^2}}(\frac{A_i^T(y - A_{-i}w_{-i})}{A_i^T A_i})$ | $w_i \leftarrow \arg \min_{w_i} (\frac{1}{2} \ w_i \phi(x_i) - y_i\ ^2$ $+ \frac{\rho}{2} \ w_i - z + u_i\ ^2)$ $z \leftarrow S_{\frac{\lambda}{\rho N}}(\bar{w} + \bar{u})$ $u_i \leftarrow u_i + w_i - z$ |
| 逻辑回归 | $u_i \leftarrow [u_i - \gamma_t(\lambda - y_t \phi_i(x_t)(1 - \frac{1}{1+e^{-y_t w^T \phi(x_t)}}))]_+$ $v_i \leftarrow [v_i - \gamma_t(\lambda + y_t \phi_i(x_t)(1 - \frac{1}{1+e^{-y_t w^T \phi(x_t)}}))]_+$ $w = (u_1 - v_1, \dots, u_d - v_d)$ | 详见 [30] | $w_i \leftarrow \arg \min_{w_i} (l_i(w_i \phi(x_i))$ $+ \frac{\rho}{2} \ w_i - z + u_i\ ^2)$ $z \leftarrow S_{\frac{\lambda}{\rho N}}(\bar{w} + \bar{u})$ $u_i \leftarrow u_i + w_i - z$ |
| 支持向量机 | $w \leftarrow w - \gamma_t \begin{cases} \lambda w, & \text{if } y_t w^T \phi(x_t) \geq 1 \\ \lambda w - y_t \phi(x_t), & \text{otherwise} \end{cases}$ | 详见 [31] | $w_i \leftarrow \arg \min_{w_i} (\frac{1}{2} \ w_i \phi(x_i) - y_i\ ^2$ $+ \frac{\rho}{2} \ w_i - z + u_i\ ^2)$ $z \leftarrow \frac{\rho}{(1/\lambda) + N\rho}(\bar{w} + \bar{u})$ $u_i \leftarrow u_i + w_i - z$ |


^a $[x]_+ = \max\{0, x\}$.^b $S_a(x) = \begin{cases} x + a, & \text{if } x < -a \\ x - a, & \text{if } x > +a \\ 0, & \text{otherwise} \end{cases} \quad (a > 0).$

第四章 系统实现

本章中，我们主要从算法参数的自动选择、对稀疏数据的执行两个优化实现相关的方面介绍GLM+系统实现，其中4.1节介绍参数的自动选择模块，4.2节介绍对稀疏数据存储进行的优化，4.3节介绍对 e^x 求解进行的相关优化，4.4节介绍并行执行更新策略相关的优化。

4.1 参数的选择

SGD与ADMM在运行过程中都需要指定超参数，且超参数对优化效率有很大的影响。SGD中在求得梯度值 $\nabla l(w; x_i, y_i)$ 后，使用 $w - \eta \nabla l(w; x_i, y_i)$ 更新参数值 w ，其中的 η 被称为学习速率。对于不同的数据集，效果最好的 η 值不同，如果 η 选取过大，会导致随机梯度下降算法无法收敛到最优值，出现反复振荡甚至不收敛的现象；而如果选取的 η 值过小，则导致收敛速度非常慢，同样影响算法的性能 [14]。ADMM中同样存在类似的参数 ρ ，如果 ρ 的值过大，会导致振荡出现，过小的 ρ 值则会导致算法效率很差 [32]。在数据集规模很大时，每次迭代都需要很长的时间，大量时间会被浪费在数据集选择参数上。因此，如果优化器选择或用户主动选择了SGD或ADMM，就需要对算法中的参数进行选择。

在选择参数上，我们使用SGD和ADMM的一个性质：参数 ρ 、 η 对算法性能的影响并不受数据集中样本数目的影响。那么对于两个有着相同分布但大小不同的数据集而言，算法的最佳参数 ρ 、 η 是相似的 [24]。因此，选择了通过采样原数据集的方式，得到一个小的、能够代表原数据集的**测试数据集**。通过比较在测试数据集上使用不同参数 ρ 、 η 进行算法迭代的结果选择最佳的参数 ρ^* 、 η^* 。根据上文提到的性质，我们可以使用 ρ^* 、 η^* 对原数据集进行优化训练。当然，通过采样生成小数据集，在小数据集上测试获得最佳参数值的方式并不能保证我们可以得到最佳的参数，通过采样，我们可以得到一个效率可行的参数。

4.2 稀疏数据的执行优化

如果训练数据是稠密的，那么对于SGD而言，每一次根据一条数据更新参数 w 时，首先计算梯度值 $\nabla l(w; x_i, y_i)$ ，需要 $O(d)$ 时间，然后使用 $w - \eta \nabla l(w; x_i, y_i)$ 更新参数值 w ，同样需要 $O(d)$ 时间。而当训练集数据特征的维度很高且数据比较稀疏时，很多计算由于数据特征为0，对优化更新参数没有更新，算法的时间复杂度可以被优化：广义

线性模型具有相同的预测形式： $y^{predict} = f(g(w : x))$, 其中 $g(w : x)$ 是参数 w 与数据特征 x 的线性组合： $g(w : x) = w^T x$ 。那么根据链式求导法则：

$$\frac{\partial f}{\partial w_i} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial w_i} = x_i \frac{\partial f}{\partial g} \quad (4.1)$$

当使用梯度进行更新时，我们只需要计算特征值中非零的部分并更新参数即可。相似的问题也同样存在于CD与ADMM中。由此，通过只计算稀疏数据中非零项部分并更新参数，优化每轮迭代的时间可以被大大缩减。同时，优化的存储还可节省内存使用量。

为了消除对非零数据的遍历与计算的额外代价，我们针对稀疏数据改进了原有存储。因为一般的数组形式的存储没有对非零数据做特别的标记与处理，在访问前无法确定特征值是否为非零值，因此无法达到减少计算的效果。

改进后的存储结构如图4.1所示：对于SGD和ADMM而言，由于在求解过程中，需要逐条地获取数据集中每个样本的特征值，因此采用按样本存储非零特征的方式；而对于CD来说，数据集访问的方式是按特征的每个维度依次访问，因此将同一特征的非零特征存储在一起。

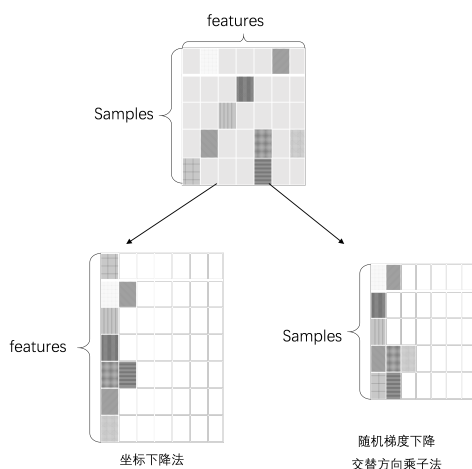


图 4.1 针对稀疏数据设计的数据存储结构

4.3 e^x 函数计算优化

在3.2节中我们已经提到，在坐标下降算法求解逻辑回归的过程中，算法需要大量的 e^x 操作，大大增加了坐标下降算法求解逻辑回归需要的执行时间。因此，在系统实现的过程中，我们对 e^x 的执行过程进行了大量的优化。

为了对 e^x 操作进行优化,我们首先分析了Java语言中的Math.exp()实现,Java的exp()操作使用native方法实现,以保证算法运行的效率。

从算法4.1可以看到,Java的Math.exp()操作较为复杂,其中包括很多的计算步骤,然而对于一些对Math.exp()的精度要求不是特别高但操作数量巨大的应用而言(如坐标下降求解逻辑回归算法),因此在系统实现的过程中,我们尝试了两种近似求解 e^x 的近似算法。

算法 4.1 exp函数实现(x)

```
public static double exp(double x){
    if (x != x)
        return x;
    if (x > EXP_LIMIT_H)
        return Double.POSITIVE_INFINITY;
    if (x < EXP_LIMIT_L)
        return 0;

    // Argument reduction.
    double hi;
    double lo;
    int k;
    double t = abs(x);
    if (t > 0.5 * LN2){
        if (t < 1.5 * LN2){
            hi = t - LN2_H;
            lo = LN2_L;
            k = 1;
        }
        else{
            k = (int) (INV_LN2 * t + 0.5);
            hi = t - k * LN2_H;
            lo = k * LN2_L;
        }
        if (x < 0){
            hi = -hi;
            lo = -lo;
            k = -k;
        }
        x = hi - lo;
    }
    else if (t < 1 / TWO_28)
        return 1;
    else
        lo = hi = k = 0;

    // Now x is in primary range.
    t = x * x;
    double c = x - t * (P1 + t * (P2 +
        t * (P3 + t * (P4 + t * P5))));
    if (k == 0)
        return 1 - (x * c / (c - 2) - x);
    double y = 1 - (lo - x * c / (2 - c) - hi);
    return scale(y, k);
}
```

考虑到 e^x 函数存在

$$e^x = e^{x_1} \times e^{x_2} \times \dots \times e^{x_n} \quad (4.2)$$

的特点, 其中 $\sum_{i=1}^n x_i = x$, 也因此对于 e^x 的计算, 我们可以通过预先计算不同范围内的 e^x , 并将其存储到数组中, 在计算 e^x 函数值的时候, 将其依据上式进行组合即可得到所求函数值, 例如, 如果我们事先计算了 $e^1, e^2, \dots, e^{0.1}, e^{0.2}, \dots, e^{0.9}, e^{0.01}, e^{0.02}, \dots, e^{0.09}$, 那么在求解 $e^{3.45}$ 时, 我们可以由 $e^{3.45} = e^3 \times e^{0.4} \times e^{0.05}$ 将三个预先存储计算的值相乘得到最终的 $e^{3.45}$ 值。因此, 近似算法1 的算法代码如算法4.2所示:

算法 4.2 exp函数近似算法1(x)

```
public void preCompute(){
    for(int i = 0; i < 5; i++){
        double l = Math.pow(10, -2*i);
        for(int j = 0; j < 100; j++){
            expTable[i][j] = Math.exp(j * l);
        }
    }
}

public double exp(double val) {
    double result = 1.0;
    int bitNum = 0;
    for(int i = 0; i < 3; i++){
        bitNum = (int)val % 100;
        if(bitNum >= 0) {
            result *= expTable[i][bitNum];
        }else{
            result /= expTable[i][-bitNum];
        }
        val *= 100;
    }
    return result;
}
```

因此, 通过利用 e^x 的函数性质, 同时将预先计算好的 e^x 进行存储, 在实际计算 e^x 时我们就可以快速而准确的得到所求的函数值。

同时, 根据浮点数存储的格式及 e^x 的性质, e^x 可以通过左右移位结合四则运算近似地得到 e^x 的函数值 [33], 具体的算法如下:

我们也从精确度、算法的执行效率两个方面对两种 e^x 函数的近似算法进行了比较: 理论和实验都表明, 近似算法1的误差率与预先计算存储的 e^x 的大小相关, 可以证明, 如果预先计算存储 $e^{10^{-i} \times j}$, 其中 $0 \leq i \leq a$, $0 \leq j \leq 9$, 共计 $10 \times a$ 项 e^x 值, 那么误差值 $\epsilon \leq 5 \times (e^{10^{-(a+1)}} - 1)$, 如果我们选取 $a = 5$, 那么误差值 $\epsilon \leq 5 \times 10^{-6}$. 同时, 在某些特殊的值下, 近似算法2最差的误差率在4%。

在算法的执行效率上, 我们对 e^x 精确算法、近似算法1、近似算法2进行了对比, 依次运行100,000,000次三种算法, 可以得到:

算法 4.3 exp函数近似算法2(x)

```

static double[] ExpAdjustment = new double[256] {
    1.040389835,
    //...
    1.040618648
};
static double FastExp(double x)
{
    var tmp = (long)(1512775 * x + 1072632447);
    int index = (int)(tmp >> 12) & 0xFF;
    return BitConverter.Int64BitsToDouble(tmp << 32)
        * ExpAdjustment[index];
}

```

表 4.1 exp()函数的实验比较

| 求解算法 | e^x 精确算法 | 近似算法1 | 近似算法2 |
|----------|------------|-------|-------|
| 执行时间(ms) | 15756 | 6656 | 5623 |

从实验结果可以看出，与精确算法相比，近似算法1的加速比为2.367，近似算法2的加速比为2.802。求解的时间上大大缩短,可以有2-3倍的效率提升。

综合近似算法的精确度和执行效率两个方面考虑，在系统实现中，我们选择使用了预先计算好的 e^x 进行存储，计算时将对应的 e^x 相乘得到最终结果的近似算法1。

4.4 并行的更新策略选择

当参与计算的数据集太大，单线程执行优化求解的时间无法忍受时，我们可以选择在多线程或者分布式的环境下运行算法。随机梯度下降算法与坐标下降算法的本质是每次选取一条样本/一个维度，计算针对该样本/维度的代价函数的最小化。二者区别在于随机梯度下降每次选取一个样本，对该样本涉及到的特征进行更新；而坐标下降则每次选取一个特征维度，使用整个训练集对该特征维度进行更新。从本质上讲，二者是顺序执行的串行算法。并行运行需要进行调整。并行执行随机梯度下降算法和坐标下降算法需要在多个处理器上同时计算更新值，并将更新值提交到全局的参数。问题在于更新提交到全局参数时可能发生冲突。一种方法是使用锁等同步机制维护全局解，避免冲突的发生。但由于对广义线性模型的优化求解更新频繁，计算简单，多处理器并行计算获得的加速比会被处理器间的同步操作抵消，导致并行无法获得很好的加速比。

因此我们选择了图4.2所示的无锁的更新方法。在单机多线程的环境下，处理器通过共享内存来访问参数，并且这些参数不进行加锁。每个处理器可以自由无锁地访问全局参数。无锁的更新方法必然会导致部分更新被覆盖，但如果数据特征是稀疏的，

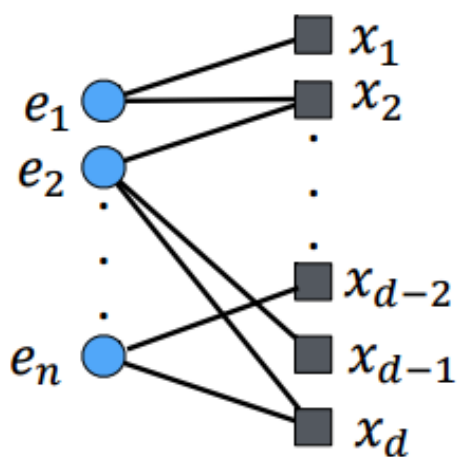


图 4.2 并行执行时无锁更新

那么发生冲突的概率就会很低，处理器之间几乎不会进行更新信息重写，该方法几乎可以达到一个最优的收敛速度 [34]。

与此同时，交替方向乘子法则易于并行地运行，因为其本身就是并行算法。在并行环境下，每个处理器处理部分数据。我们先将数据集按行划分为 n 部分,其中 x_i, y_i 分别表示 x, y 数据的第 i 个分块，被第 i 个处理器处理。分别进行求解子问题，并将各个处理器得到的 w, u, z 汇总后更新。

第五章 实验与分析

本章首先介绍实验相关的数据集，然后说明具体的实验方案以及相应的结果分析。

5.1 数据集介绍

在本章中，我们首先通过实验对比了GLM+与常见的机器学习系统Shotgun和scikit-learn^①的性能，然后测试了SGD、CD和ADMM求解线性回归、逻辑回归、Lasso和SVM模型的求解性能，验证了GLM+中的算法优化器的有效性，最后对GLM+系统的参数自动调优的效果、稀疏数据的优化进行了实验测试。

本章将在2个人工生成数据集和2个真实的回归数据集上进行了线性回归与Lasso算法的实验，在3个真实分类数据集上进行了逻辑回归与SVM算法的实验，数据集见表5.1。其中(1)-(4)为回归数据集，(5)-(7)为分类数据集，(2)、(3)数据集是人工生成的数据集。其他数据集来源于libSVM数据集 [35]。人工生成的数据集仿照Single-Pixel Camera datasets生成 [9]。

表 5.1 实验数据集

| 数据集 | 样本数 | 特征数 | 非零特征数 |
|----------------------|--------|-----------|------------|
| (1)log1p.E2006.train | 16,087 | 4,272,227 | 96,218,242 |
| (2)pepers_05_6_6 | 32,768 | 65,536 | 1,181,116 |
| (3)pepers_05_12_12 | 32,768 | 65,536 | 2,362,232 |
| (4)E2006.train | 16,087 | 150,360 | 19,834,498 |
| (5)rcv1.binary | 20,242 | 47,236 | 1,501,157 |
| (6)gisette | 6,000 | 5,000 | 29,730,000 |
| (7)news20.binary | 19,996 | 1,355,191 | 8,942,471 |

5.2 与其他系统的对比实验

为了验证我们所提出的优化选择器的有效性和系统实现的高效性，我们与目前常见的广义线性模型求解系统Shotgun与scikit-learn进行了实验对比。其中Shotgun使

^① scikit-learn求解SVM使用了封装的libsvm/liblinear实现。

用CD求解线性模型和Lasso模型；scikit-learn使用最小二乘法求解线性模型，使用CD求解Lasso模型、支持向量机和逻辑回归模型。实验选取了(1)、(4)、(5)、(6)、(7)共5个真实数据集，在相同条件下分别求解线性回归、Lasso、逻辑回归和支持向量机模型，记录不同系统收敛到相同loss值时的执行时间(单位为s)，实验结果见表5.2。

表 5.2 GLM+与其他系统的对比实验

| | Linear Regression | | | Lasso | | |
|---------|---------------------|--------------|-------------|-------------|-------------|-------------|
| | (1) | (4) | | (1) | (4) | |
| GLM+ | 26.69 | 27.24 | | 8.38 | 0.23 | |
| shotgun | - | - | | 40.55 | 3.30 | |
| sklearn | >500 | 101.15 | | 27.09 | 2.04 | |
| | Logistic Regression | | | SVM | | |
| | (5) | (6) | (7) | (5) | (6) | (7) |
| GLM+ | 0.18 | 2.74 | 0.52 | 1.45 | 6.34 | 1.78 |
| shotgun | 0.36 | 6.14 | 1.19 | - | - | - |
| sklearn | 0.68 | 14.51 | 33.90 | 2.46 | 7.34 | 5.73 |

从表5.2的实验结果可以看出，GLM+系统在四种广义线性模型的求解上都优于Shotgun与scikit-learn两个系统。

在求解线性回归模型时，我们选取的CD的执行效率要高于scikit-learn的最小二乘法, GLM+的性能是scikit-learn的4倍到20倍以上，主要原因在于GLM+选择了正确优化算法。在求解Lasso模型时，三个系统都选择使用CD求解，但GLM+系统在执行时间上都优于其他两个系统。GLM+的性能是Shotgun的5倍到15倍，是scikit-learn的3倍到10倍。在逻辑回归模型的求解中，GLM+系统选择的ADMM在性能上要优于选择CD的Shotgun与scikit-learn系统，性能是它们2倍到60倍。在SVM算法的实验中，我们选取的SGD的性能也优于scikit-learn系统使用的坐标下降法性能，最多能提升3倍的性能。

从上述实验可以看到，与Shotgun和scikit-learn两个常见的广义线性模型求解系统相比，GLM+系统会选择更优的优化算法，配合上在系统实现上的优化，GLM+在实际求解执行过程中求解性能优于常见的广义线性模型求解系统。因此，我们所提出的优化选择器和系统实现是高效的。

5.3 优化策略实验

为了衡量不同优化算法在求解不同模型的求解优劣，我们需要比较不同优化算法的收敛性能。由于不同的数据集的数据有所差别，且不同优化算法求解的原理不同，能够优化收敛到的最优值和每轮迭代的执行时间也略有差别，所以我们比较每个优化算法执行到收敛时的所需时间。收敛的标准为两次相邻的迭代的代价函数值的差小于1%：

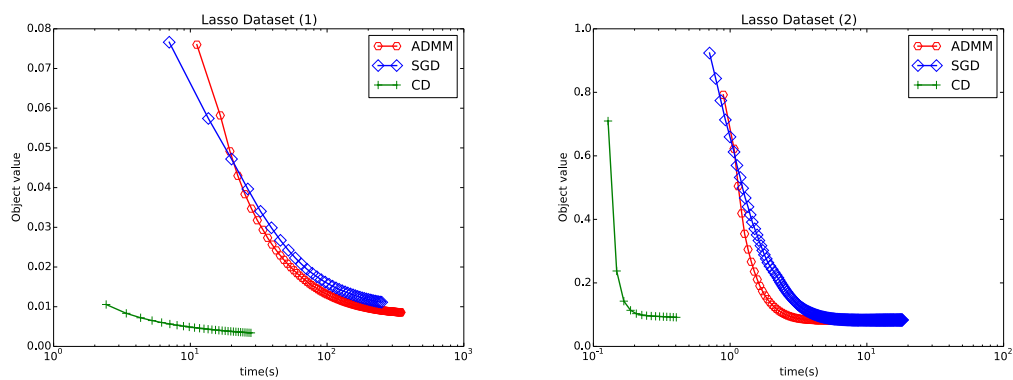
$$\frac{Q(w_{t+1}) - Q(w_t)}{Q(w_t)} < 1\% \quad (5.1)$$

在选择SGD与ADMM的参数上，我们使用了前文提到的算法性质：即参数 η 对算法性能的影响并不受数据集中样本数目的影响。因此，我们选择通过采样原数据集的方式，得到一个小的、能够代表原数据集的测试数据集，并在参数集列表 $[1, 0.33, 0.1, 0.033, 0.01, 0.0033, 0.001, 3.3 \times 10^{-3}, 1 \times 10^{-3}, 3.3 \times 10^{-4}, 1 \times 10^{-4}, 3.3 \times 10^{-5}, 1 \times 10^{-5}, 3.3 \times 10^{-6}, 1 \times 10^{-6}]$ 中选择一个效果最好的参数 η^* ，使用 η^* 对原数据集进行优化训练。我们针对每个算法模型，进行了一组实验。在每组实验中，使用SGD、CD 和ADMM 三种优化算法进行了算法优化的测试。实验数据见图5.1。

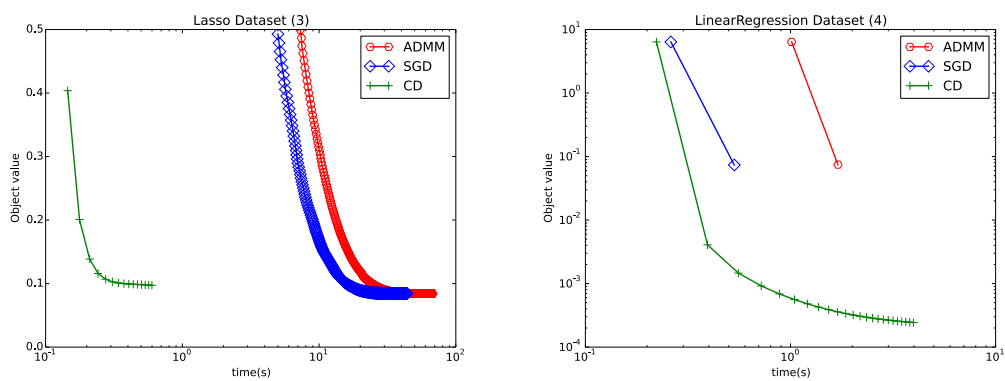
图5.1(a),5.1(b)中展示的是三种优化算法优化求解线性回归和Lasso算法的效果示意。可以看出，在三种优化算法求解线性回归和Lasso算法时，使用CD优化方法能够得到最好的效果。与SGD法跟ADMM相比，CD能够收敛到相同甚至更低的代价值，同时迭代至收敛所需的时间更少。原因正如第3节中对规则1的分析，CD每轮迭代的执行时间要少于SGD和ADMM。在三种优化算法的收敛速度类似的条件下，迭代到相同代价值的时间自然更少。因此，线性回归算法与Lasso算法应该使用CD方法优化。

图5.1(c)展示了SVM模型在三种优化算法下的收敛速度对比。通过观察可以看到，在其中的两个数据集中，CD优化至收敛所需的时间最少，其次是SGD法和ADMM。这与第3节的分析是一致的。然而，从图中也可以看出，CD在收敛的数值上并没有达到SGD跟ADMM所达到的收敛数值，这在数据稠密的数据集上表现尤为明显，CD收敛时的数值与SGD收敛的数值相差12倍。从图5.1(c)实验结果也可以看出，CD可以很快地迭代运行到收敛，但并不能收敛到其他两个算法的最优值。因此，SGD是最好的选择。

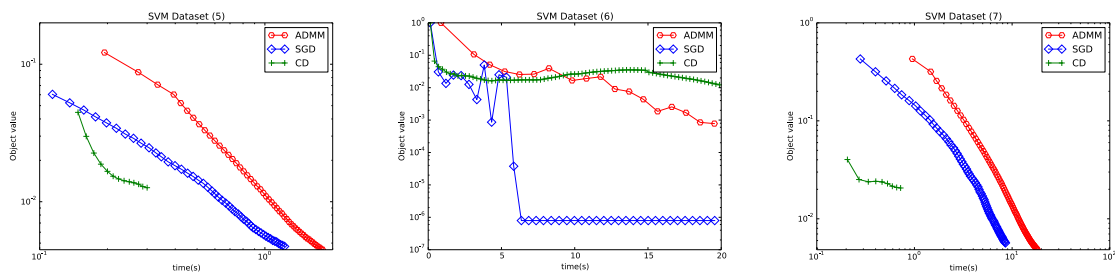
在图5.1(d)逻辑回归实验的实验结果中，我们可以看出ADMM的效果最好。因此，在GLM+中，我们使用ADMM算法对逻辑回归模型进行求解。



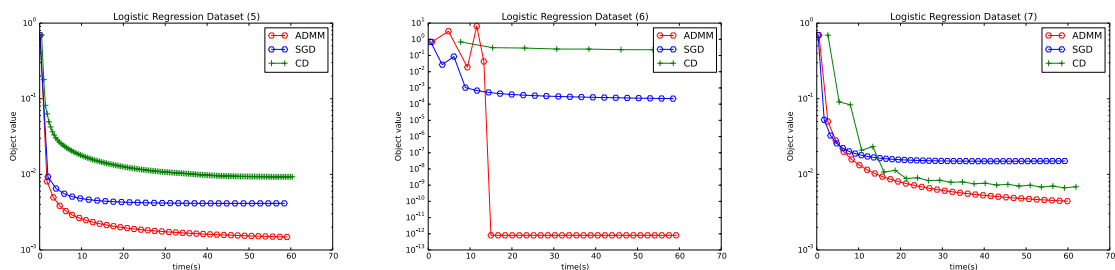
(a) Linear regression experiment



(b) Lasso experiment



(c) SVM experiment



(d) Logistic regression experiment

图 5.1 三种优化算法性能比较

5.4 参数自动调优实验

为了验证参数自动调优的有效性，我们对需要参数调节的SGD与ADMM进行了实验验证。实验选取了(3)、(4)、(5)、(7)四个数据集进行实验测试，将自动调优选择的参数与其他参数进行比较执行20轮迭代后的loss数值。实验结果如表5.3。

表 5.3 参数选择实验结果

| | Lasso(3) | Linear(4) | LR (5) | SVM (7) |
|---------|---------------|----------------|---------------|---------------|
| | ADMM | SGD | SGD | ADMM |
| 1 | - | - | 0.119 | 0.084 |
| 0.1 | - | 0.17093 | 0.108* | 0.040* |
| 0.01 | 0.085* | 0.07400 | 0.203 | 0.162 |
| 0.001 | 1.754 | 0.07327 | 0.466 | 0.511 |
| 0.0001 | 7.463 | 0.07328* | 0.656 | 0.935 |
| 0.00001 | 9.566 | 0.07329 | 0.689 | 0.993 |

表 5.4 稀疏数据优化实验结果

| | | Linear(2) | Lasso (3) | SVM (6) |
|------|--------|-----------|-----------|---------|
| ADMM | sparse | 974 | 685 | 3,981 |
| | dense | 132,168 | 164,524 | 4,112 |
| CD | sparse | 199 | 294 | 4,267 |
| | dense | - | - | 4,713 |
| SGD | sparse | 269 | 383 | 56,684 |
| | dense | 399,569 | 942,579 | 54,406 |

通过分析表5.3的实验结果发现，使用小数据集自动调优选择的参数与真实最优的学习速率大多数时候是相同或非常接近的，这说明我们的参数自动选择调优是有效的。

5.5 稀疏数据的优化实验

为了避免对非零数据的遍历与计算的额外代价，我们针对稀疏数据改进了原有存储。为了验证我们对稀疏数据所做的优化是有效的，我们对原有实现与稀疏优化存储

的实现进行了实验对比。实验选取了(2)、(3)、(6)三个数据集进行实验, 比较两种实现分别执行5轮迭代所需的时间。实验结果如表5.4(单位为ms), 其中dense为原有的数组存储方法, sparse为GLM+提出的优化存储方法, 其中dense实现中部分数据集由于尚未优化时所需内存过大无法执行。

从实验数据可以看出, 对于稠密数据集(6)而言, 稀疏优化存储所需时间与原有实现类似, 而对于稀疏数据集(2)、(3)而言, 稀疏优化存储的实现迭代所需时间远小于原有实现的时间。GLM+系统实现中提出的针对稀疏数据的优化是十分高效的。

通过以上GLM+与Shotgun、scikit-learn系统的对比、GLM+系统的选择与优化实验和两项工程优化共四个实验, 证明我们的优化选择器与基于此优化器实现的GLM+系统是实用且高效的。

第六章 总结与展望

广义线性模型是线性模型的扩展，常见的广义线性模型包括线性回归、套索回归、支持向量机和逻辑回归等算法。因为其模型简单并且训练高效被广泛地应用在数据挖掘的各个领域。然而对于广义线性模型的求解而言，通常有随机梯度下降、坐标下降和交替方向乘子法等很多优化算法。常用的广义线性模型求解系统，都使用单一的优化算法去求解广义线性模型。然而在实际实验中我们发现，使用单一的优化算法并不能够对每种广义线性模型都得到较高的性能和较好的拟合效果。本文旨在解决寻找求解常见广义线性模型的最优化的算法。

以随机梯度下降、坐标下降和交替方向乘子法三种常用的优化算法求解线性回归、套索回归、支持向量机和逻辑回归四种常见的广义线性模型为研究对象，在详尽的实验对比的基础上，结合执行时间与收敛速度两方面的分析，总结出三条启发式规则。并基于此设计了一个基于规则的优化算法选择器，可以根据模型类别，自动地选择最优的优化算法。最后，通过加入参数选择、稀疏优化两项工程技术，构建了一个优化求解广义线性算法的原型系统GLM+。

分析与实验都表明，坐标下降法求解线性回归、Lasso算法，随机梯度下降法求解SVM算法，交替方向乘子法求解逻辑回归算法具有很好的表现。本文也通过与常见求解系统的实验对比，证明了我们的优化选择器与基于此优化器实现的GLM+系统是实用且高效的。

综上所述，本文的主要贡献在于：

1. 在大量真实数据集的实验基础上，结合对随机梯度下降、坐标下降与交替方向乘子法的执行时间与收敛速度两方面的分析，提出了基于规则的算法选择优化器。
2. 在基于规则的算法选择优化器的基础上加入了参数选择、针对稀疏数据的执行优化两项工程技术构建了一个优化求解广义线性算法的完整系统GLM+。
3. 通过与常用广义线性模型求解系统Shotgun和scikit-learn的实验比较，证明了GLM+系统的实用性与高效性。

当然，本文得到的结论主要基于不同优化算法求解的效果实验，同时结合定性的理论分析得出了优化算法选择器的启发式规则；受限于随机算法的求解理论性不足和个人对优化算法求解理论的理解不足，并没有挖掘出不同优化求解算法对不同模型求解效果差异的深层原因，对优化算法的启发式规则的阐释与理解欠缺，也是本文的缺点之一。

关于进一步的研究方向,可以尝试将求解的模型扩展到更多的广义线性模型,甚至更多的浅层模型;同时优化算法中我们主要的关注目标是随机梯度下降、坐标下降和交替方向乘子法三种优化算法,对于广义线性模型来说,其他的优化算法,如L-bfgs或者二阶优化算法的比较与分析,以及不同优化算法之间的区别与联系也是值得研究的问题。

参考文献

- [1] George AF Seber, Alan J Lee. Linear regression analysis[M], vol. 936. John Wiley & Sons, 2012
- [2] Robert Tibshirani. Regression shrinkage and selection via the lasso[J]. Journal of the Royal Statistical Society Series B (Methodological). 1996:267–288
- [3] David W Hosmer Jr, Stanley Lemeshow, Rodney X Sturdivant. Applied logistic regression[M], vol. 398. John Wiley & Sons, 2013
- [4] Scott Menard. Applied logistic regression analysis[M], vol. 106. Sage, 2002
- [5] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, Joaquin Quiñonero Candela. Practical Lessons from Predicting Clicks on Ads at Facebook[C]. ADKDD. 2014, 5:1–5:9. URL <http://doi.acm.org/10.1145/2648584.2648589>
- [6] Jiawei Jiang, Bin Cui, Ce Zhang, Lele Yu. Heterogeneity-aware Distributed Parameter Servers[C]. SIGMOD. 2017, 463–478. URL <http://doi.acm.org/10.1145/3035918.3035933>
- [7] H. Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, Jeremy Kubica. Ad click prediction: a view from the trenches[C]. KDD. 2013, 1222–1230. URL <http://doi.acm.org/10.1145/2487575.2488200>
- [8] Johan AK Suykens, Joos Vandewalle. Least squares support vector machine classifiers[J]. Neural processing letters. 1999, **9**(3):293–300
- [9] Joseph K Bradley, Aapo Kyrola, Danny Bickson, Carlos Guestrin. Parallel coordinate descent for l1-regularized loss minimization[J]. arXiv preprint arXiv:11055379. 2011
- [10] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python[J]. Journal of Machine Learning Research. 2011, **12**(Oct):2825–2830
- [11] Léon Bottou. Large-scale machine learning with stochastic gradient descent. Proceedings of COMPSTAT’2010, Springer, 2010. 177–186

- [12] Stephen J Wright. Coordinate descent algorithms[J]. Mathematical Programming. 2015, **151**(1):3–34
- [13] Stephen P. Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers[J]. Foundations and Trends in Machine Learning. 2011, **3**(1):1–122
- [14] Léon Bottou. Stochastic gradient descent tricks. Neural networks: Tricks of the trade, Springer, 2012. 421–436
- [15] Rie Johnson, Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction[C]. Advances in neural information processing systems. 2013, 315–323
- [16] Ilya Sutskever, James Martens, George Dahl, Geoffrey Hinton. On the importance of initialization and momentum in deep learning[C]. International conference on machine learning. 2013, 1139–1147
- [17] Matthew D Zeiler. ADADELTA: an adaptive learning rate method[J]. arXiv preprint arXiv:12125701. 2012
- [18] Diederik P Kingma, Jimmy Ba. Adam: A method for stochastic optimization[J]. arXiv preprint arXiv:14126980. 2014
- [19] John Duchi, Elad Hazan, Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization[J]. Journal of Machine Learning Research. 2011, **12**(Jul):2121–2159
- [20] Tijmen Tieleman, Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude[J]. COURSE: Neural networks for machine learning. 2012, **4**(2):26–31
- [21] Jerome Friedman, Trevor Hastie, Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent[J]. Journal of statistical software. 2010, **33**(1):1
- [22] Paul Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization[J]. Journal of optimization theory and applications. 2001, **109**(3):475–494
- [23] Stephen Boyd, Lieven Vandenberghe. Convex optimization[M]. Cambridge university press, 2004
- [24] Mingyi Hong, Zhi-Quan Luo. On the linear convergence of the alternating direction

- method of multipliers[J]. Mathematical Programming. 2017, **162**(1-2):165–199
- [25] Ankan Saha, Ambuj Tewari. On the finite time convergence of cyclic coordinate descent methods[J]. arXiv preprint arXiv:10052146. 2010
- [26] Kai-Wei Chang, Cho-Jui Hsieh, Chih-Jen Lin. Coordinate descent method for large-scale l2-loss linear support vector machines[J]. Journal of Machine Learning Research. 2008, **9**(Jul):1369–1398
- [27] Shai Shalev-Shwartz, Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization[J]. Journal of Machine Learning Research. 2013, **14**(Feb):567–599
- [28] Bingsheng He, Xiaoming Yuan. On the $O(1/n)$ Convergence Rate of the Douglas–Rachford Alternating Direction Method[J]. SIAM Journal on Numerical Analysis. 2012, **50**(2):700–709
- [29] Ce Zhang, Christopher Ré. Dimmwitter: A study of main-memory statistical analytics[J]. Proceedings of the VLDB Endowment. 2014, **7**(12):1283–1294
- [30] Guo-Xun Yuan, Kai-Wei Chang, Cho-Jui Hsieh, Chih-Jen Lin. A comparison of optimization methods and software for large-scale l1-regularized linear classification[J]. Journal of Machine Learning Research. 2010, **11**(Nov):3183–3234
- [31] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S Sathya Keerthi, Sellamanickam Sundararajan. A dual coordinate descent method for large-scale linear SVM[C]. Proceedings of the 25th international conference on Machine learning. ACM, 2008, 408–415
- [32] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers[J]. Foundations and Trends in Machine Learning. 2011, **3**(1):1–122
- [33] Nicol N Schraudolph. A fast, compact approximation of the exponential function[J]. Neural Computation. 1999, **11**(4):853–862
- [34] Benjamin Recht, Christopher Re, Stephen Wright, Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent[C]. Advances in neural information processing systems. 2011, 693–701
- [35] Chih-Chung Chang, Chih-Jen Lin. LIBSVM: a library for support vector machines[J]. ACM transactions on intelligent systems and technology (TIST). 2011, **2**(3):27

致谢

写到致谢，已是毕业论文的最后一步了。同样，我的求学生涯也要接近尾声了。回想三年生活，仿佛弹指一挥间，仿佛写本科毕业的致谢就发生在昨天，一切似乎没什么大的不同，只是本科待过的1631现在已经变成了会议室，1601也旧貌换新颜。更不同的是，这次，是真的要离开北大了。

感谢燕园赠与我七年的美好时光。曾很多次畅想过毕业的场景，却没曾想过还未毕业却已开始怀念它。

想感谢的人很多。最想感谢的人，是我的导师崔斌教授。崔老师治学严谨，在学术上有很强的洞察力，总能捕捉到学术前沿的研究方向，能在崔老师的指导下做研究是我们的荣幸。从本科二年级下半学期进入实验室师兄开始，到三年的硕士求学，崔老师都始终给予我耐心的指导和支持。感谢老师的悉心指导，这些年松懈过也努力过，自知与您的期望相距甚远，感谢老师的宽容与期待。崔老师在学术上的孜孜不倦，将一直是我学习的榜样。

感谢实验室一起并肩作战，共同成长的同学们。感谢施晓罡师兄，在我刚刚加入实验室时带我融入实验室，带我克服开始面对未知领域的恐惧，在科研工作上给我的诸多指点。感谢余乐乐师兄，感谢你在自己繁重的博士科研任务之外，悉心带领不靠谱的我完成科研任务。感谢在学术上孜孜不倦，辛苦耕耘的邵莹侠师兄、江佳伟师兄，从你们身上学到了很多做人做事的道理。感谢提携后辈的徐宁师兄，在MAX+的时光学到了很多。感谢土豪谢怡然师姐和黄艳香师姐，是你们的存在让实验室的生活变得精致而温暖。感谢认真靠谱的“大师兄”张智鹏，看到你科研认真，生活开心，由衷的为你高兴。感谢实验室的闫学灿、李旭鹏、苑斌、黎洋、苗旭鹏、符芳诚、谢旭、李恬、陈一茹，王子辰等各位师弟师妹，是你们让实验室有了朝气，祝愿你们学业顺利，前程似锦。也感谢1602的幽默乐观的DK师兄，朱纪乐、宋卫平、宋伊萍等等很多很多的同学，无法想象没有你们的日子。

感谢我的三位可爱的室友，陈修司，郑淇木和江忻玺同学。我们一起努力过，拼搏过，也一起疯狂过。我们共同欢笑，也在难过的时候做彼此的依靠。和你们一起的宿舍生活虽然只有三年，却值得我永久回忆。如果不是你们，生活将远不及现在缤纷多彩。感谢我的本科同学，吕彬彬，杨岳，李炜，马辉。与你们一起玩耍的时光十分快乐，时常让我有还是青涩本科的错觉，希望与你们做一辈子的好朋友。

感谢在实习中遇到的各位同事。感谢我的mentor周珣，耐心的指导我，带我养成了很多好习惯，也带领我一窥推荐算法神秘面纱。感谢王鹏的耐心帮助与指导，

和nice的乔木老师，感谢李磊，丁昊，肖文之，王浩，周义，薛昊，以及推荐组的各位同事，感谢你们对我的鼓励与帮助。暑期实习的两个月让我受益良多。希望还能再有机会与你们做有意义、有挑战的事。

感谢我的父母。感谢你们在我求学生涯里一如既往的关心与支持。我的父母一直在背后默默的支持我，为我牺牲了很多。自从上大学开始，跟他们在一起的日子就变得很少。除了看着他们慢慢变老，并没有为他们做过什么。希望以后我能做一个称职的儿子，也希望不辜负你们的期望，成为一个更好的人。

想要感谢的人太多，无法枚举，谨在此一并感谢。感谢所有与我度过七年大学快乐时光的所有人，对于曾经伤害过的人，诚挚的说一声抱歉。他们中有的人还出现在我的生活中，而有的已经或即将渐行远，淡出彼此的生活。感谢你们曾出现在我的人生中，感谢你们教给我的道理，感谢你们陪我成长，感谢你们对我的包容，感谢你们带来的美好时光，没有你们我的生活将远不会如此精彩。无论以后是否还有交集，和你、你们在一起的快乐时光我一直不曾忘记，也永远不会忘记。

每一段时光终会划上一段句号。再次感谢所有曾经帮助过我的人们，感谢你们带来的温暖。我也会努力工作，希望成为更好的人，将我的温暖带给你们。

攻读硕士期间的研究成果

发表论文

1. 王羚宇, 余乐乐, 江佳伟, 崔斌: GLM+: 一个高效的广义线性模型求解系统. 第30届中国数据库学术会议 NDBC 2017
2. Lele Yu, **Lingyu Wang**, Yingxia Shao, Long Guo, Bin Cui. GLM+: An Efficient System for Generalized Linear Models, IEEE BigComp 2018

参与的科研项目

1. 国家”九七三”重点基础研究发展规划项目基金, 2014CB340405, 网络大数据计算的基础理论及其应用研究, 2014.01-2018.12
2. 国家自然科学基金面上项目, 61272155, 支持互联网级的大规模数据库系统研究, 2013.01-2016.12

北京大学学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名： 日期： 年 月 日

学位论文使用授权说明

（必须装订在提交学校图书馆的印刷本）

本人完全了解北京大学关于收集、保存、使用学位论文的规定，即：

- 按照学校要求提交学位论文的印刷本和电子版本；
- 学校有权保存学位论文的印刷本和电子版，并提供目录检索与阅览服务，在校园网上提供服务；
- 学校可以采用影印、缩印、数字化或其它复制手段保存论文；
- 因某种特殊原因需要延迟发布学位论文电子版，授权学校在 ☐ 一年 / ☐ 两年 / ☐ 三年以后在校园网上全文发布。

（保密论文在解密后遵守此规定）

论文作者签名： 导师签名： 日期： 年 月 日