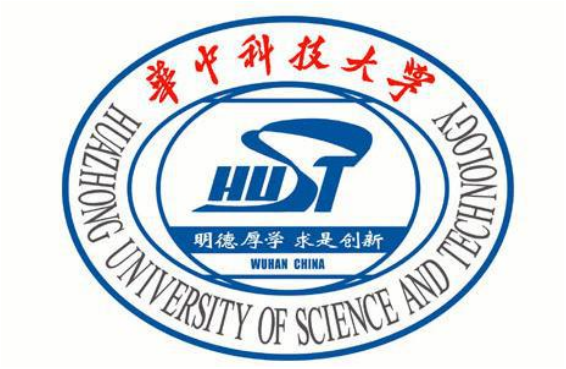
华中科技大学计算机科学与技术学院

机器学习结课报告



专　　业：计算机科学与技术

班　　级：计算机 ACM1801 班

学　　号：U201814741

姓　　名：宋奕欣

成　　绩：

指导教师：何琨

完成日期：　2020 年　 6 月 25 日

目 录

1 逻辑回归算法综述

1.1 概述

逻辑回归模型由统计学家 David Cox 于 1958 年发明，该模型本身只是根据输入对输出的概率进行建模，不进行统计分类，并不是分类器，但它可以用于构建分类器：选择某个数值作为阈值，将输出大于阈值的那个输入划分为一个类别，低于阈值则划分为另一个类别，这就构成了一种常见的二值分类器。

逻辑回归是一种有监督的统计学习方法，其本质上是一种广义线性回归模型，下面从线性模型开始介绍。

1.2 线性模型

给定由 d 个属性描述的示例 $\boldsymbol{x} = (x_1;x_2;...;x_d)$，其中$x_i$是 x 在第 i 个属性上的取值，线性模型试图学得一个通过属性的线性组合来进行预测的函数，即

$$f(x) = w_1x_1 + w_2x_2 + ... + w_dx_d + b$$

一般用向量形式写成

$$f(x) = w^Tx + b$$

其中 $\boldsymbol{w} = (w_1;w_2;...;w_d)$. W 和 b 需要经过训练后得到。

线性模型形式简单、易于建模，但却蕴含着机器学习中一些重要的基本思想。许多功能更为强大的非线性模型可在线性模型的基础上通过引入层级结构或高维映射而得。此外，由于参数 w 直观表达了各属性在预测中的重要性，因而线性模型具有很好的可解释性。

1.3 广义线性模型

线性模型具有一定的局限性，只能用于线性可分的情况下，但却有丰富的变化。比如说我们常见的线性模型为 $y = w^Tx + b$ ，但我们可以通过加入指数函数得到 $y = e^{w^Tx+b}$，这样即将输出标记的对数作为线性模型的目标，就得到了对数线性回归模型。当然我们还可以用其它的一些单调可微函数 f(x)，即“广义线性回归模型”，即：

$$y = f(w^T x + b)$$

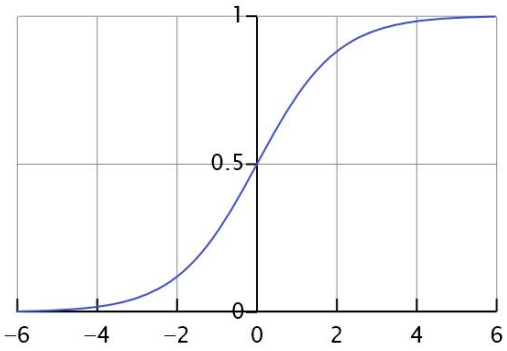
广义线性模型的数学公式最早由 John Nelder 和 Robert Wedderburn 建立。它对包括线性回归、逻辑回归和泊松回归等在内的多个模型进行了统一。他们还提出了一种迭代加权最小二乘法来对该模型参数进行最大似然估计。

1.4 逻辑回归模型

线性回归模型用于回归学习，如果能够找到一个单调可微的联系函数 $g(\cdot)$，将分类任务的真实标记 y 与线性回归模型的预测值 $w^T x + b$ 联系起来，就可以实现分类任务。逻辑函数即是这样的一个常用函数：

$$y = \frac{1}{1 + e^{-z}}$$

逻辑函数或逻辑曲线是一种常见的 S 形函数（即 Sigmoid 函数），它是皮埃尔·弗朗索瓦·韦吕勒在 1844 年（或 1845）在研究它与人口增长的关系时命名的。其图像如下：



从上图可以看出，逻辑函数将 z 值转换为一个接近 0 或 1 的 y 值，并且其输出值在 z=0 附近变化很陡。将逻辑函数作为$g^{-}(\cdot)$代入上式，得到

$$y = \frac{1}{1 + e^{-(w^T x + b)}}$$

上式又可以变化为

$$\ln \frac{y}{1 - y} = w^T x + b$$

逻辑回归模型的有许多优点：无需实现假设数据分布，直接对分类可能性进

行建模，避免了假设分布不准确的问题。且它不是仅预测出类别，而是可得到近似概率预测。

1.5 逻辑回归模型参数估计

逻辑回归模型学习时，对于给定的训练数据集

$$\boldsymbol{T} = \{(x_1,y_1), (x_2,y_2),...,(x_N,y_N)\}$$

其中$x_i \in R^n, y_i \in \{0,1\}$，可以运用极大似然估计法估计模型参数。设：

$$P(Y = 1|x) = \pi(x), P(Y = 0|x) = 1 - \pi(x)$$

似然函数为：

$$\prod_{i=1}^{N} [\pi(x_i)]^{y_i}[1 - \pi(x_i)]^{1-y_i}$$

对数似然函数为：

$$L(w) = \sum_{i=1}^{N} [y_i \log \pi(x_i) + (1 - y_i) \log (1 - \pi(x_i))]$$

对 $L(w)$求极大值，得到 w 的估计值。

这样，模型的学习过程就变成了以对数似然函数为目标函数的最优化问题了。可以看出这个目标函数是任意阶可导的凸函数，有很好的数学性质，有许多数值优化算法都可以用于求取最优解，如梯度下降法、牛顿法等。

1.6 逻辑回归核方法应用

核方法对于无线性决策边界的数据集有很好处理模式，课堂上老师讲授过核方法的威力是很大的。由于当低维度数据通过映射函数映射到高维度时是需要花费很多计算的，而且效果有时候也并不是很好，但是核方法的技巧就很灵活。

在课堂上老师已经讲过对于线性分类器都可以使用核方法来进行处理。即有

$$w_* = \sum \alpha x_i$$

通过对损失函数进行处理可将问题转化为对 α 的优化问题。通过计算可得对于 α 的梯度下降公式如下：

$$\alpha_{\mathrm{n+1}} = \alpha_n + \sum_{i=1}^{n} lr * (y - p) * K[i]$$

$$\mathrm{bias+} = \mathrm{lr} * (y - p)$$

1.7 梯度下降之优化

1.7.1 一个框架回顾优化算法

首先我们来回顾一下各类优化算法。

深度学习优化算法经历了 SGD -> SGDM -> NAG ->AdaGrad -> AdaDelta -> Adam -> Nadam 这样的发展历程。Google 一下就可以看到很多的教程文章，详细告诉你这些算法是如何一步一步演变而来的。在这里，我们换一个思路，用一个框架来梳理所有的优化算法，做一个更加高屋建瓴的对比。

首先定义：待优化参数：$\vec{\mathrm{w}}$，目标函数：$\mathrm{f}(\vec{w})$，初始学习率为 α 。

而后，开始进行迭代优化，在每个 epoch t：

1. 计算目标函数关于当前参数的梯度：$g_t = \nabla f(w_t)$

2. 根据历史梯度计算一阶动量和二阶动量：

$m_t = \phi(g1, g2, \cdots, g_t); V_t = \varphi(g_1, g_2, \cdots, g_t)$

3. 计算当前时刻的下降梯度：$\eta_{\mathrm{t}} = \alpha \bullet m_t / \sqrt{V_t}$

4. 根据下降梯度进行更新：$w_{t+1} = w_t - \eta_t$

掌握这个框架，优化算法的设计就变得很容易。

1.7.2 SGD

SGD 即随机选择一部分数据来做梯度下降。

1.7.3 SGDM

为了抑制 SGD 的震荡，SGDM 认为梯度下降过程可以加入惯性。下坡的时候，如果发现是陡坡，那就利用惯性跑的快一些。SGDM 全称是 SGD with momentum，在 SGD 基础上引入了一阶动量：

$$m_t = \beta_1 * m_{t-1} + (1-\beta_1) * g_t$$

一阶动量是各个时刻梯度方向的指数移动平均值，约等于最近（$1/1-\beta_1$）个时刻的梯度向量和的平均值。

也就是说，t 时刻的下降方向，不仅由当前点的梯度方向决定，而且由此前累积的下降方向决定。β 的经验值为 0.9，这就意味着下降方向主要是此前累积的下降方向，并略微偏向当前时刻的下降方向。想象高速公路上汽车转弯，在高速向前的同时略微偏向，急转弯可是要出事的。

1.7.4 NAG

SGD 还有一个问题是困在局部最优的沟壑里面震荡。想象一下你走到一个盆地，四周都是略高的小山，你觉得没有下坡的方向，那就只能待在这里了。可是如果你爬上高地，就会发现外面的世界还很广阔。因此，我们不能停留在当前位置去观察未来的方向，而要向前一步、多看一步、看远一些。

NAG 全称 Nesterov Accelerated Gradient，是在 SGD、SGD-M 的基础上的进一步改进，改进点在于步骤 1。我们知道在时刻 t 的主要下降方向是由累积动量决定的，动量走了一步，那个时候再怎么走。因此，NAG 在步骤 1，不计算当前位置的梯度方向，而是计算如果按照累积动量走了一步，那个时候的下降方向：

$$g_t = \nabla f(w_t - \alpha \cdot m_{t-1} / \sqrt{V_{t-1}})$$

然后用下一个点的梯度方向，与历史累积动量相结合，计算步骤 2 中当前时刻的累积动量。

1.7.5 AdaGrad

此前我们都没有用到二阶动量。二阶动量的出现，才意味着“自适应学习率”优化算法时代的到来。SGD 及其变种以同样的学习率更新每个参数，但深度神经网络往往包含大量的参数，这些参数并不是总会用得到（想想大规模的 embedding）。对于经常更新的参数，我们已经积累了大量关于它的知识，不希望被单个样本影响太大，希望学习速率慢一些；对于偶尔更新的参数，我们了解的信息太少，希望能从每个偶然出现的样本身上多学一些，即学习速率大一些。

怎么样去度量历史更新频率呢？那就是二阶动量——该维度上，迄今为止所有梯度值的平方和：

$$\mathrm{V_t} = \sum_{i=1}^{t} g_i^{\,2}$$

1.7.6 Adam

谈到这里，Adam 和 Nadam 的出现就很自然而然了——它们是前述方法的集大成者。我们看到，SGD-M 在 SGD 基础上增加了一阶动量，AdaGrad 和 AdaDelta 在 SGD 基础上增加了二阶动量。把一阶动量和二阶动量都用起来，就是 Adam 了——Adaptive + Momentum。

SGD 一阶动量：

$$\mathrm{m_t} = \beta_1 * m_{t-1} + (1-\beta_1) * g_t$$

加上 Adadelta 的二阶动量：

$$V_t = \beta_2 * V_{t-1} + (1-\beta_2) g_t^{\,2}$$

2 实验

2.1 实验任务

用二分类判断一个人的收入是否会超过 50K。

本次实验将采用上文介绍的逻辑回归算法来做一个分类器，通过每个人的特征来预测其收入是否会超过 50K。

2.2 数据集

1.数据集的大小为 4000 行×59 列；

2.4000 行数据对应 4000 个人,ID 编号从 1 到 4000；

3.59 列数据中，第一行为 ID，最后一行为 label(1 或 0)表示年收入是否大于

50K，中间的 57 行为 57 种属性值。

2.3 试验设置

在本次实验中，我们将数据集 4000 条样例中的 3000 条作为训练集，剩下 1000 条作为测试集。在训练集上对学习性能进行评估，试验评估指标为模型准确率：

准确率 = 分对样本数 ÷ 总样本数

实验环境：Windows10

编程语言：Python

2.4 算法描述

首先对数据进行预处理，即将每个属性进行归一化操作，并将数据集前 3000 行作为训练集，后 1000 行作为测试集分开。

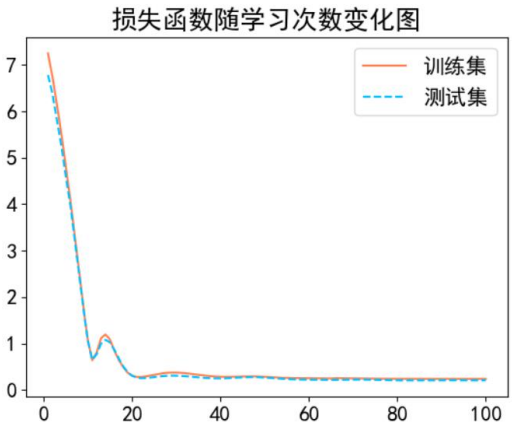
接下来用逻辑回归算法对训练集进行学习，通过逻辑回归算法拟合数据，并测试准确率。

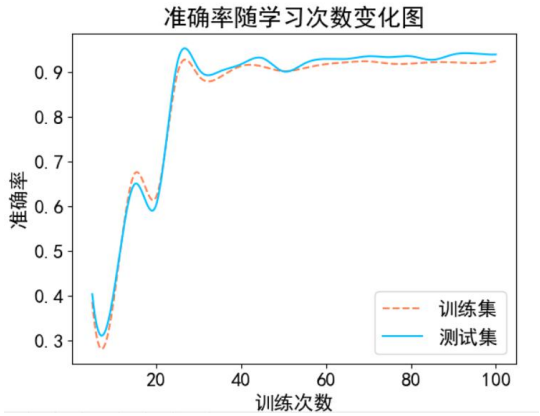
最后对结果进行分析。

2.5 试验步骤

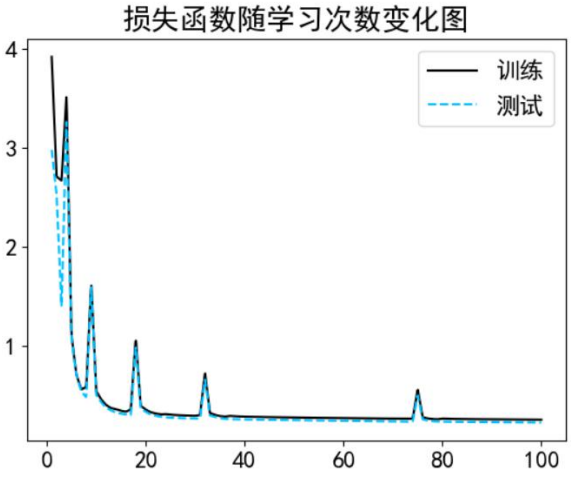
1) 将数据集进行预处理。
2) 用逻辑回归算法在训练集上进行训练，建立逻辑回归模型，并计算模型在训练集上的准确率。
3) 对测试集进行预测，统计预测结果。
4) 对预测偏差进行分析，完成试验。
5) 对实验进行进一步分析。

2.6 实验结果





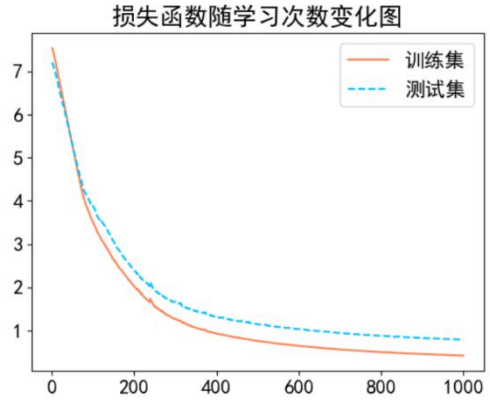
此图给出了目前我能训练出的最好的效果，即准确率可以达到 94.6%，这是利用了 adgrad 优化，最初学习率取 0.1 的结果，这也是损失函数在第一步迭代的过程中会上升的原因。我们可以看到在使用 adgrad 优化时，损失函数非常的平滑，而且收敛的非常快，我们与不用优化的损失函数图进行对比。下图为采用固定学习率的损失函数图。可以看到，损失函数在第 20 轮时仍然没有达到收敛，甚至由于学习率太大而会出现较大的波动。

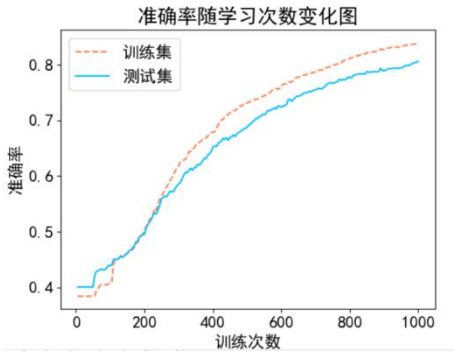


这很明显的体现了一个 adgrad 优化的一个特点，即前期激励，即当梯度较小时放大梯度；后期惩罚，即缩小梯度，这会导致提前结束训练过程。

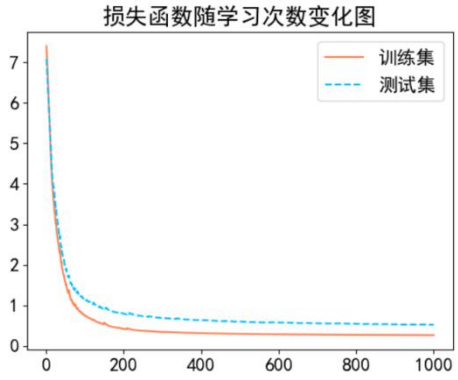
现采用固定学习率，随机梯度下降的方式，对比不同学习率对实验准确率以及损失函数的影响。

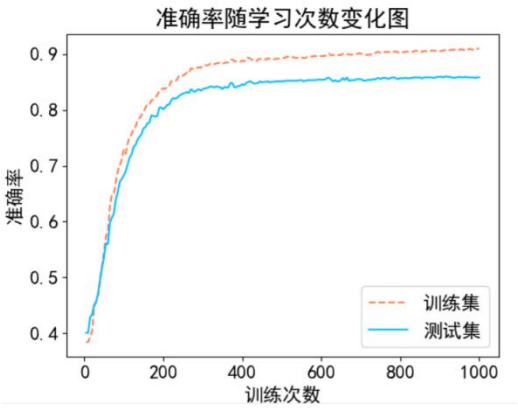
当学习率为 0.01 时，



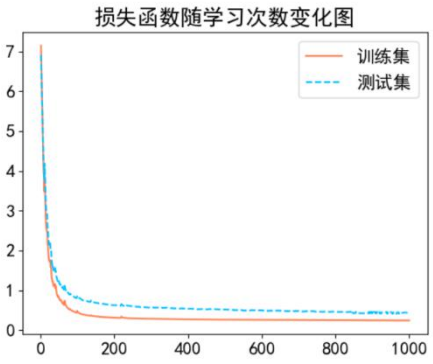


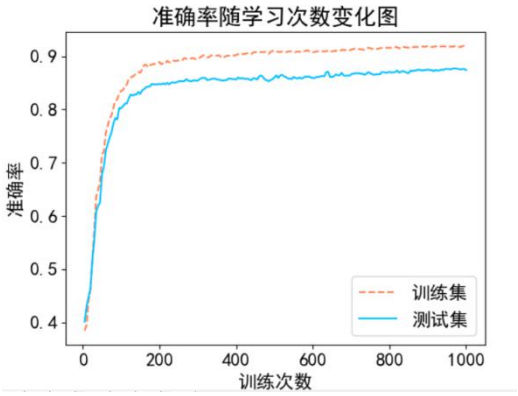
当学习率为 0.05 时，



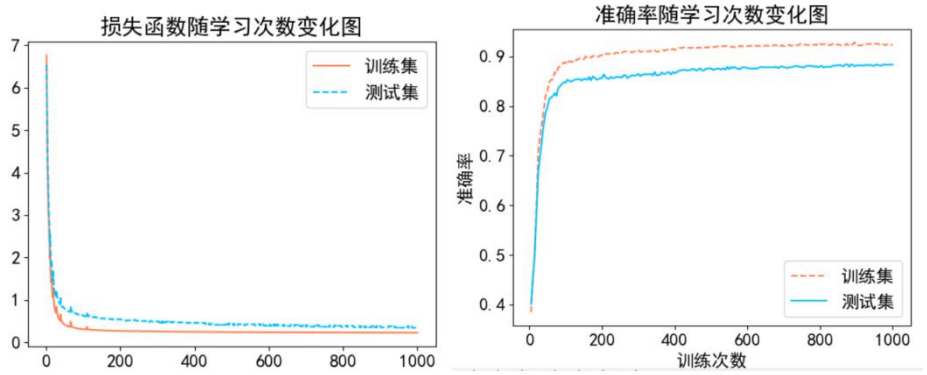


当学习率为 0.1 时，

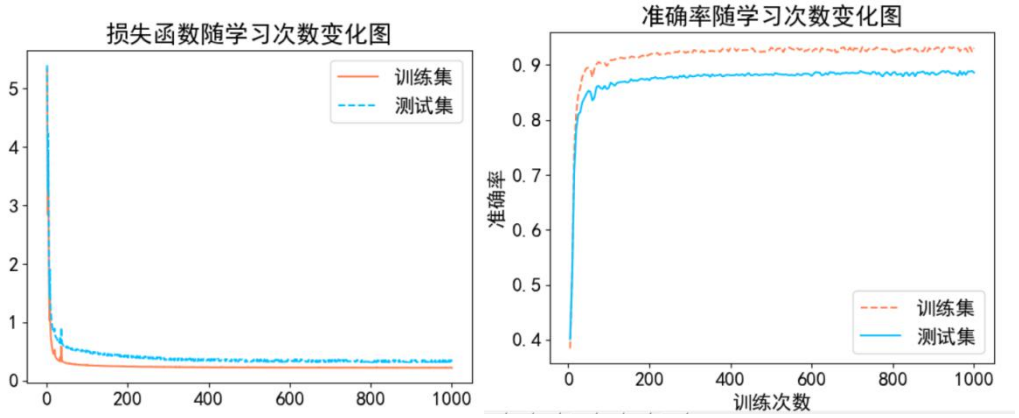




当学习率为 0.2 时，



当学习率为 0.5 时，

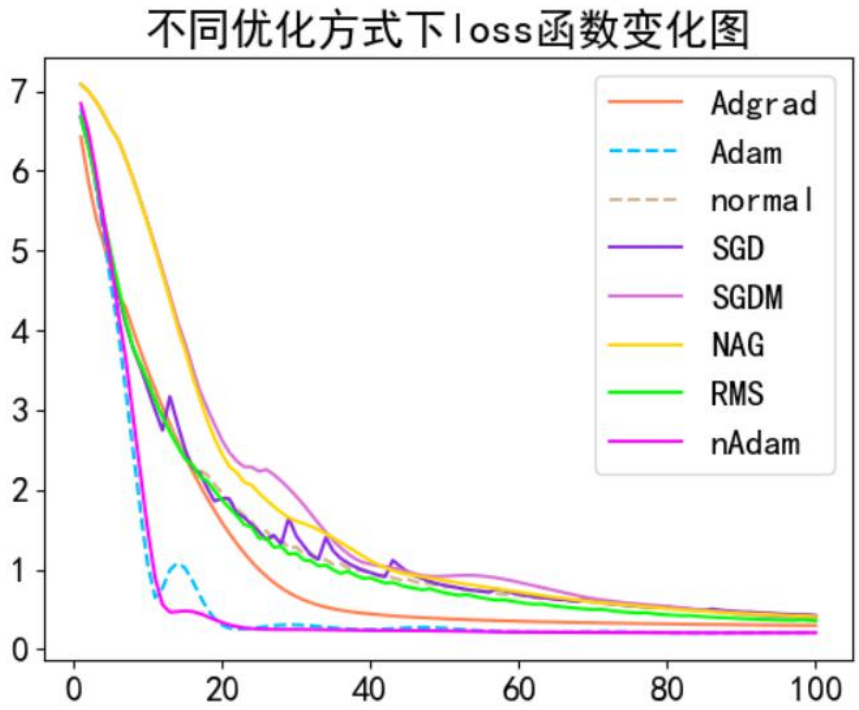


可见不同的学习率会导致准确率出现较大的差别，一方面是因为如果学习率太小则会导致收敛速度过于慢从而导致迭代次数到了却还没有收敛，也有可能陷入局部最优。另一方面是如果学习率太大则会导致收敛难以到达使得 loss 到最小的点，从而导致无法收敛，训练的参数不停的摇摆不定。

对于以上的处理方法则是引入动态学习率，比如说 adagrad、adam 优化。这样即使最初时学习率调整的不合适，也能够根据训练的增多而使得学习率能够进行适当的调整。

我们不妨记录一下迭代过程中利用不同优化算法时学习率的变化。

这是取初始学习率为 0.1 时的记录。



对比起来虽然普通的梯度下降和 SGD 以及 SGDM 在算法上是比较简单的，但是其收敛速度并不算慢，与 SGD 比起来，普通的梯度下降算法更稳定一些，而 SGD 收敛的更快一些。

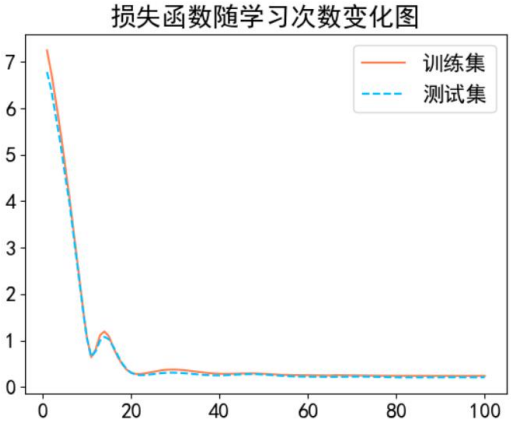
当 SGD 与 Adgrad、Adam 比起来时则有比较明显的不同。显然 Adgrad 和 Adam 的收敛速度要快得多，尤其是 Adam，几乎在第 20 轮就已经收敛了。

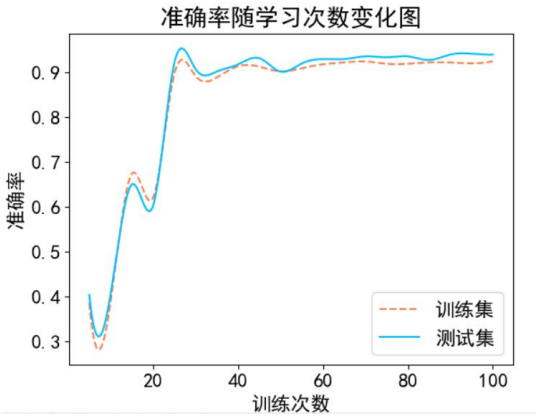
但这并不意味着 Adam 永远是最好的训练方法，因为已经有人指出了 Adam 的弊端，它并不一定收敛，也有可能错过全局最优解。但这些都是在比较极端的情况下，通常情况下对于初学者来说，直接使用 Adam 的优化是比较合适的。但对于高级的专业的人工智能研究员来说，他们可以使用各种各样灵活的选择。这需要我们长期的积累才可以做到。

下面研究正则化系数对于学习过程的影响。

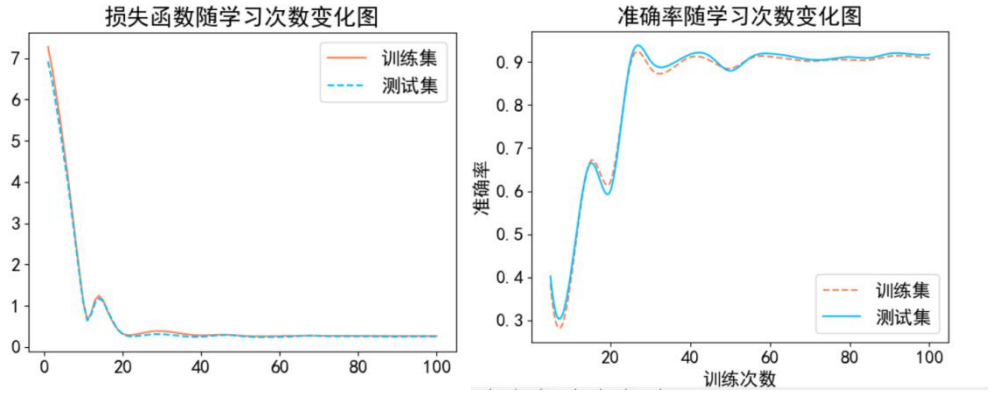
取学习率为 0.1 时进行比较。

当正则化系数为 0.001 时，损失函数和准确率如图。

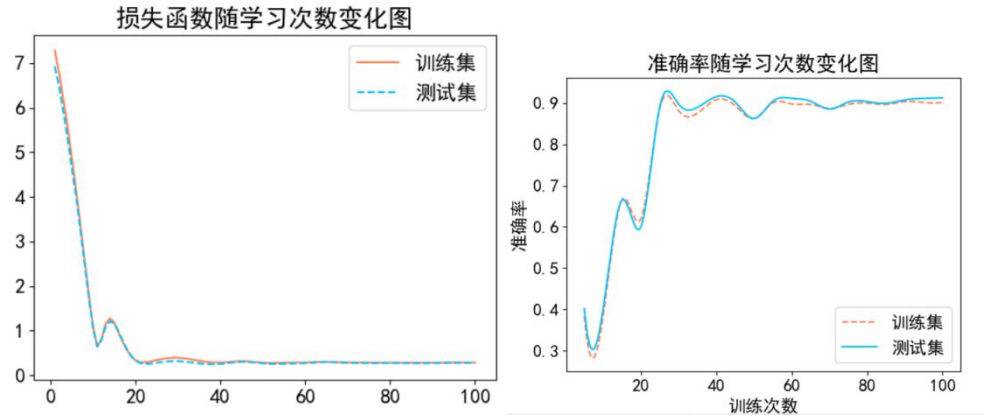




当正则化系数为 0.005 时，

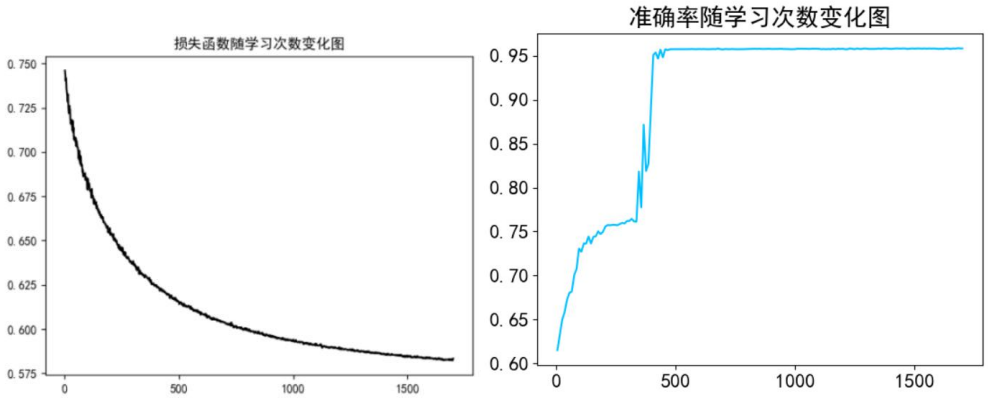


当正则化系数为 0.01 时，



随着正则化系数的增大，损失函数以及准确率的图趋势没有大体变化，而随着正则化系数的增大，准确率的最大值在逐渐的下降，这其实应该是数据没有过拟合的正常情况，因为正则化是为了防止过拟合导致训练集的准确率太高的时候而测试集准确率太低。而我们实验过程中数据并没有过拟合，所以也在情理之中。

使用核方法 loss 曲线的变化如下图。



3 实验总结

通过这次实验，对逻辑回归、核方法以及梯度下降算法的各种优化有了比较深入的理解。

在结课项目之前，我对逻辑回归的认识仅局限于课堂上的讲授，但对于实际的操作还没有清晰的认识。在这次实验最开始的时候，我通过查阅网上的资料，现在能够手写出一份还算不错的逻辑回归算法。显然，我对逻辑回归算法有了更加深入的认识。之后，由于老师课堂上提过核方法在线性分类器上都可以有很好的应用，我就想到了核方法在逻辑回归中的应用。我就去查阅了一些资料，但发现国内对于逻辑回归的核方法的课程是比较少的。但通过搜索引擎我还是有学到一些有关核方法的处理模式。通过这点尝试，我第一次把核方法与线性分类器结合起来。本来课程上学的核方法就比较迷糊，觉得挺难用，但实际上真正应用起来其实也不是那么难。最后由于梯度下降算法有很多的优化版本，我想了解一下各种优化其本质，于是就做了一下各个优化版本的 loss 曲线进行了比较。通过这，我也学到了很多有关梯度下降算法优化的一些本质特点。

总体来说，通过这次结课项目，我能够较好地将课上学到的知识与实验结合起来，基于逻辑回归去应用了很多课上学到的其它地知识。我觉得这个结课项目收获满满。

4 参考文献

[1] Juliuszh.一个框架看懂优化算法之异同 SGD/AdaGrad/Adam.https://zhuanlan.zhihu.com/p/32230623

[2] winrar_setup.rar.逻辑回归(logistics regression).https://blog.csdn.net/weixin_39445556/article/details/83930186

[3] 何琨.机器学习内部讲义

[4] Byron_NG.机器学习数据预处理——标准化/归一化方法.https://www.cnblogs.com/bjwu/p/8977141.html