— “笔记”

April 30, 2017; rev. Tuesday 26th September, 2017

Qian Wang

第一章　Adaboost

Introduction

本章内容来自于网络以及张志华老师的就《机器学习导论》课程

本章基本内容就是给定一堆弱分类器，然后通过各种组合组成一个人强的分类器

1.1　离散的 Adaboost

离散的 AdaBoost 算法步骤：

$\boldsymbol{w}$ 表示给数据的权值，$\boldsymbol{\alpha}$ 表示给分类器的权值

1. start with weights $\boldsymbol{w_i} = \frac{1}{N} \quad \boldsymbol{i = 1...N}$
2. repeat for m=1 to M
   - 使用输入数据训练一个分类器 $\boldsymbol{G_m(x)} \in \boldsymbol{(-1, 1)}$
   - 计算误差 err:

$$\boldsymbol{E_w[I_(y_i \neq G_m(x_i))] = \frac{\sum_{i=1}^{N} w_i I_(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}}$$

   - 输出 $\boldsymbol{\alpha_m = \frac{1}{2} log(\frac{1-err}{err})}$, 从这里可以看出分类器的误差越大，权值越小
   - set $\boldsymbol{w_i = \frac{w_i}{Z_m}} \exp(\boldsymbol{\alpha_m I_(y \neq G_m(x))}$，其中 $\boldsymbol{Z_m}$ 是规范化因子, $\boldsymbol{Z_m = \sum_{i=1}^{N} w_i} \exp(\boldsymbol{-\alpha_m y_i G_m(x_i)})$ 如果一个数据点分类错了，那么给这个点的权值大一点。
3. 这样我们就得到了 $\boldsymbol{G(x) = sign[\sum_{m=1}^{M} \alpha_m G_m(x)]}$

1.2　Forward Stagest Additive Modeling

考虑加法模型 (additive model)

$$\boldsymbol{f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)} \tag{2.1}$$

其中,$\boldsymbol{b}(\boldsymbol{x};\boldsymbol{\gamma}_m)$ 为基函数，$\boldsymbol{\gamma}_m$ 为基函数的参数，$\boldsymbol{\beta}_m$ 为基函数的系数。

在给定训练数据以及 Loss Function 的情况下，相当于一个加法模型 $\boldsymbol{f}(\boldsymbol{x})$ 相当于一个 minimize Loss Function 的问题：

$$\boldsymbol{min}_{\beta_m,\gamma_m}\sum_{i=1}^{N}\boldsymbol{L}(\boldsymbol{y}_i,\sum_{m=1}^{M}\boldsymbol{\beta}_m\boldsymbol{b}(\boldsymbol{x};\boldsymbol{\gamma}_m)) \tag{2.2}$$

从前向后，每一步值学习一个基函数及其系数，即每一步只需优化如下的损失函数：

$$\boldsymbol{min}_{\beta,\gamma}\sum_{i=1}^{N}\boldsymbol{L}(\boldsymbol{y}_i,\sum_{m=1}^{M}\boldsymbol{\beta}\boldsymbol{b}(\boldsymbol{x};\boldsymbol{\gamma})) \tag{2.3}$$

前向分布算法：

输入：训练数据 $\boldsymbol{T}=(\boldsymbol{x}_1,\boldsymbol{y}_1),...(\boldsymbol{x}_N,\boldsymbol{y}_N)$,Loss Function$\boldsymbol{L}(\boldsymbol{x},\boldsymbol{f}(\boldsymbol{x}))$, 基函数 $\boldsymbol{b}(\boldsymbol{x};\boldsymbol{\gamma})$

输出：加法模型 $\boldsymbol{f}(\boldsymbol{x})$

1. 初始化 $\boldsymbol{f_0}(\boldsymbol{x})=\boldsymbol{0}$
2. repeat for m=1 to M
   - 计算参数 $\boldsymbol{\beta}_m,\boldsymbol{\gamma}_m$

$$(\boldsymbol{\beta}_m,\boldsymbol{\gamma}_m)=\arg\min_{\beta,\gamma}\sum_{i=1}^{N}\boldsymbol{L}(\boldsymbol{y}_i,\boldsymbol{f}_{m-1}(\boldsymbol{x}_i)\boldsymbol{\beta}\boldsymbol{b}(\boldsymbol{x};\boldsymbol{\gamma})) \tag{2.4}$$

   - 更新 $\boldsymbol{f}_m(\boldsymbol{x})=\boldsymbol{f}_{m-1}(\boldsymbol{x})+\boldsymbol{\beta}_m\boldsymbol{b}(\boldsymbol{x};\boldsymbol{\gamma}_m)$
3. 得到加法模型

$$\boldsymbol{f}(\boldsymbol{x})=\sum_{m=1}^{M}\boldsymbol{\beta}_m\boldsymbol{b}(\boldsymbol{x};\boldsymbol{\gamma}_m) \tag{2.5}$$

第二章　Logistic Regression

Introduction

本章内容主要来自于个人 YY

2.1　Sigmoid 函数

sigmoid 函数的主要公式如下所示：

$$p(x) = \frac{1}{1 + e^{-x}} \tag{1.1}$$

该公式主要用在二分类的问题中作为最后的预测结果分类，或者作为激活函数在神经网络中使用。

为什么要使用 sigmoid?

由于 sigmoid 的良好性质，求导容易，反向传播速度快，输出全部在 0-1 之间，永远为正，严格递增

缺点：在中心点变化快，比较敏感

2.2　Logistic Regression

逻辑回归主要用在二分类问题中，最后给出该样本属于正类或者负类的概率，公式如下所示：

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} log h_\theta(x^{(i)}) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))\right]$$

(2.2)

通俗的来说，LR 就是用来将两堆数据分来

LR 优点:

- 预测结果是介于 0 和 1 之间的概率
- 容易使用和解释
- 预测速度较快，计算量小

缺点:

- 当特征空间很大时，LR 的性能不是很好，不能处理大量的特征
- 容易欠拟合
- 不能处理线性不可分的数据
- 对于非线性特征，需要进行转换
- 预测结果呈 S 型，中间的概率变化很大，很敏感

2.3 一些其他的问题

1. LR 是基于概率的一个模型，所有的点都会参与模型参数的更新。SVM 是基于最大化间隔的模型，由少量的支持向量决定。

第三章　正则化方法

Introduction

本章节内容主要介绍机器学习、深度学习总的正则化方法

一般来说，所有的监督学习都可以最小化下面的函数来表示：

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w}} \sum_{i} \boldsymbol{L}(\boldsymbol{y}_i, \boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{w})) + \boldsymbol{\lambda}\boldsymbol{\Omega}(\boldsymbol{w}) \tag{0.1}$$

其中，第一项一般为模型预测的结果与真实的结果之间的差距，可以用各种各样不同的函数来表示，第二项一般为正则化项，主要目的是使我们的模型更加简单，防止过拟合。

3.1　L1 L2

3.1.1　ill-condition

我们都知道优化问题有两大难题。一个是局部最小值的问题：我们要找的是全局最小值，如果局部最小值太多，那我们的优化算法就很容易陷入局部最小而不能自拔。另外一个就是 ill-condition 的问题。加入我们有个方程组 $\boldsymbol{Ax} = \boldsymbol{b}$，我们要做的是求解 $\boldsymbol{x}$, 如果 $\boldsymbol{A}$ 或者 $\boldsymbol{b}$ 稍微的改变，会使得 $\boldsymbol{x}$ 发生很大的变化，那么这个方程组系统就是 ill-condition 的，反之就是 well-condition 的。

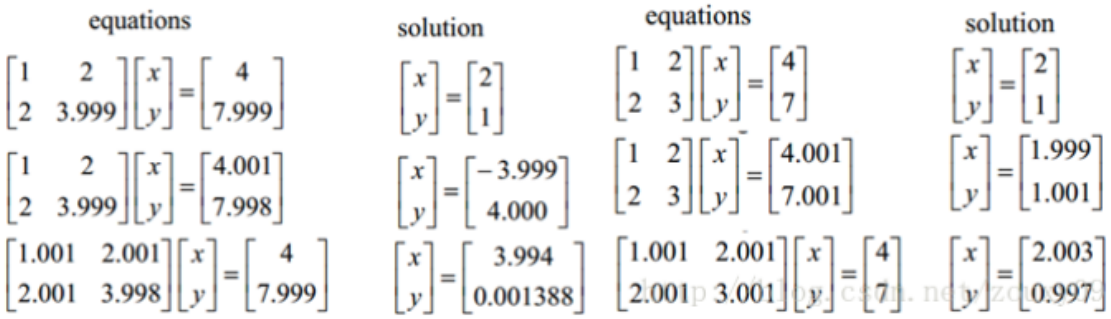


Figure 3.1: ill-condition

第一行我们假设 $\boldsymbol{Ax} = \boldsymbol{b}$，第二行我们稍微改变下 $\boldsymbol{A}$, 结果的变化就非常大，第三行我们

稍微改变下 $\boldsymbol{b}$，结果的改变同样是非常大的，因此我们可以认为我们的模型对错误的容忍力太低了，即对误差太敏感了。那么对于数据中难免存在的误差来说，模型的效果就非常差。

因此我们需要一个指标去衡量 ill-condition 问题中的变化问题。

condition number 衡量的就是在输入发生微小改变的时候，输出会发生多大的变化。也就是对系统微小变化的敏感度。condition number 比较小（在 1 附近）的就是 well-condition 的，比较大的（远大于 1 的）就是 ill-condition 的。

另外如果使用迭代优化的算法，当 condition number 太大的时候，会拖慢迭代的收敛速度。

3.1.2 L1

L1 norm 就是绝对值的和，公式如下

$\|\boldsymbol{x}\|_p = |\boldsymbol{x_1}| + |\boldsymbol{x_2}| + ... + |\boldsymbol{x}_n|$

3.1.3 L2

L2 norm 就是我们经常说的欧几里得范数，公式如下

$$\|\boldsymbol{x}\|_{\boldsymbol{2}} = (\sum_{\boldsymbol{i=1:n}} \boldsymbol{x}_i^{\boldsymbol{p}})^{\frac{\boldsymbol{1}}{\boldsymbol{p}}} \tag{1.2}$$

caffe 中 weight decay 这个参数代表 L2 范数前的系数。

L2 的优点：

- 可以防止过拟合，提升模型的泛化能力
- L2 范数有助于处理 condition number 不好的情况下逆矩阵求逆很困难的情况（待定）。

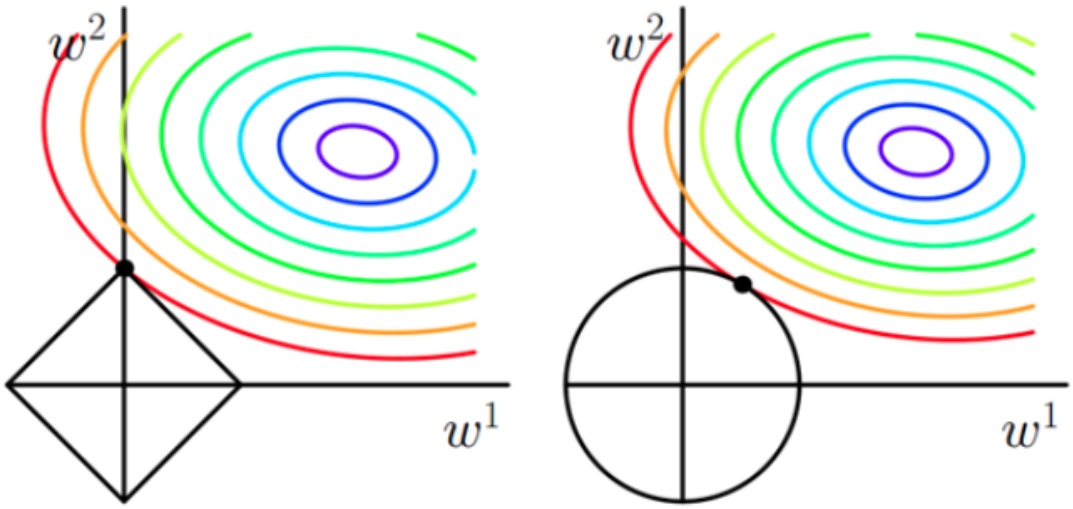
3.1.4 L1 与 L2 的不同

对于 L1 和 L2 规则化的代价函数来说，我们可以写成如下的形式

$$\boldsymbol{Lasso}\ \min_{\boldsymbol{w}} \frac{\boldsymbol{1}}{\boldsymbol{n}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|^{\boldsymbol{2}}, \boldsymbol{s.t.} \|\boldsymbol{w}\|_{\boldsymbol{1}} <= \boldsymbol{C} \tag{1.3}$$

$$\boldsymbol{Ridge}\ \min_{\boldsymbol{w}} \frac{\boldsymbol{1}}{\boldsymbol{n}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|^{\boldsymbol{2}}, \boldsymbol{s.t.} \|\boldsymbol{w}\|_{\boldsymbol{2}} <= \boldsymbol{C} \tag{1.4}$$

为了便于可视化，我们考虑两维的情况，在 $(\boldsymbol{w_1}, \boldsymbol{w_2})$ 平面上画出目标函数的等高线，而约束条件则成为平面上半径为 C 的一个 norm ball。等高线与 norm ball 相交的地方就是最优解：



(a) ℓ_1-ball meets quadratic function. ℓ_1-ball has corners. It's very likely that the meet-point is at one of the corners.

(b) ℓ_2-ball meets quadratic function. ℓ_2-ball has no corner. It is very unlikely that the meet-point is on any of axes."

Figure 3.2: 图

可以看到，在相交的地方，L1 在和每个坐标轴相交的地方都有出现解，即更容易出现 $\boldsymbol{w_1}$ 或者 $\boldsymbol{w_2}$ 为 0 的解，可以用来提取特征，更加容易产生稀疏性。

L2 的话更容易出现 $\boldsymbol{w_1}$ 和 $\boldsymbol{w_2}$ 都不是 0 的情况，虽然有时候会出现很小的值。在更高维的情况下也是这样的，即基本上只是用来正则化而已。

10

第四章　Deep Learing

Introduction

本章节内容主要是来自于深度学习中遇到的一些坑以及问题

4.1　一些小的 trick

1. 一定要对数据进行归一化

2. 训练可能会出现 loss 一开始迅速下降，然后稳定在一定的值上，这时候不要以为网络已经收敛了，有时候会出现一个拐点，即在训练很后面的时候会重新出现一个新的下降的区间，这时候网络基本上会收敛。但是不排除会出现多的拐点的情况。结论：训练过程中一定要耐心耐心再耐心。

4.2　weight decay

在损失函数中，weight decay 是放在正则项（regularization）前面的一个系数，正则项一般指示模型的复杂度，所以 weight decay 的作用是调节模型复杂度对损失函数的影响，若 weight decay 很大，则复杂的模型损失函数的值也就大。我所理解的 weight decay 就是一个调节正则化项的系数，增大则对正则化项的依赖会更高，不容易过拟合，反之则反之。

利用 weight decay 给损失函数加了个惩罚项，使得在常规损失函数值相同的情况下，学习算法更倾向于选择更简单（即权值和更小）的 NN。是一种减小训练过拟合的方法。

Weight decay is equivalent to L2 regularizer.

在训练神经网络的时候，可以先设置 weight decay 为 0，然后使网络过拟合，查看测试结果。结果正确后，设置 weight decay 为大一点的值，即加入正则化项，这样可以防止过拟合现象。具体大小需要在网络中调试。

遇到的问题：在训练 U-net 的时候，出现训练结果全是黑图的情况。原因是 weight decay 设置过大，导致网络结果没有过拟合。解决方案：设置 weight decay 为更小的值。

4.3 momentum

momentum 是梯度下降法中一种常用的加速技术。对于一般的 SGD，其表达式为 $\boldsymbol{x} \leftarrow \boldsymbol{x} - \boldsymbol{\alpha} * \boldsymbol{dx}$,x 沿负梯度下降。而带 momentum 项的 SGD 则写生如下形式：$\boldsymbol{v} = \boldsymbol{\beta} * \boldsymbol{v} - \boldsymbol{a} * \boldsymbol{dx}, \boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{v}$, 其中 $\boldsymbol{\beta}$ 是 momentum 系数，通俗的理解上面式子就是，如果上一次的 momentum（即 v）与这一次的负梯度方向是相同的，那这次下降的幅度就会加大，所以这样做能够达到加速收敛的过程。

神经网络的训练过程（也就是梯度下降法）是在高维曲面上寻找全局最优解的过程（也就是寻找波谷），每经过一次训练 epoch，搜寻点应该更加靠近最优点所在的区域范围，这时进行权重衰减便有利于将搜寻范围限制在该范围内，而不至于跳出这个搜索圈，反复进行权重衰减便逐渐缩小搜索范围，最终找到全局最优解对应的点，网络收敛。momentum 是冲量单元，也就是下式中的 m，作用是有助于训练过程中逃离局部最小值，使网络能够更快速地收敛，也是需要经过反复地 trial and error 获得的经验值

主要作用：防止陷入局部最小值

第五章　Caffe

Introduction

本章节内容主要是关于 caffe 框架的一些知识。

5.1　lmdb

lmdb 格式的文件在使用 Python 程序进行生成的时候，如果需要重复生成，则需要先删除原来的。否则会在原先的 lmdb 上重新添加文件。

5.2　net protobuf

此部分内容有待添加

5.3　solver

solver 文件为 caffe 的训练参数的文件，主要存储一些训练的超参数

运行代码为： caffe train –solver=*slover.prototxt

一个例子：

```
1     train_net: "lenet_train.prototxt"
2     test_net: "lenet_test.prototxt"
3         test_iter: 100
4         test_interval: 500
5
6         base_lr: 0.01
7         lr_policy: "fixed"
8
9         momentum: 0.9
```

```
type: SGD
weight_decay: 0.0005
display: 100
max_iter: 20000
snapshot: 5000
snapshot_prefix: "models"
solver_mode: CPU
```

下面来一个一个解释这些程序的意思

1.

train_ net: "lenet_ train.prototxt"

test_ net: "lenet_ test.prototxt"

这两行用于定于训练网络和测试网络，可以是同一个网络，用 net：train_ test.prototxt 来表示, 为上一节的内容。注意：文件的路径要从 caffe 的根目录开始，其他所有的配置都是这样的。

2. 接下来 test_ iter 表示一次训练需要加载多少个数据，许训练中的 batch size 是一致的；test_ iterval 表示经过多少此训练的 Iteration 后进行一次测试，如 500 表示没经过 500 个 Iteration 进行一次测试。另外，如果网络不想进行测试的话，可以在 solver 文件中加入如下的参数 test_ initialization: false 这样不管经过多少次的 Iteration 都不会进行测试。

3. 有关于 learing rate 的东西

base_ lr 表示初始的一个 learing rate，如果 lr_ policy 如果设置为 fixed，训练过程中会一直维持这个 learing rate 不再改变，其他的都是会在训练过程中逐渐变化的。

lr_ policy 可设置的值如下所示：

- fixed: 保持 base_ lr 不变
- step: 如果设置为 step, 则还需要设置一个 stepsize, 返回 $base\_lr * gamma^{floor * \frac{iter}{stepsize}}$, 其中 iter 表示当前的迭代次数。如设置 gamma=0.9，stepsize=100 表示在第一百次迭代后 learning rate 下降到原来的 0.9 倍
- exp: 返回 $base\_lr * gamma^{i}ter$，iter 为当前迭代次数
- inv: 如果设置为 inv, 还需要设置一个 power, 返回 $base_{l}r * (1 + gamma * iter)^{-power}$
- multistep: 如果设置为 multistep, 则还需要设置一个 stepvalue。这个参数和 step 很相似，step 是均匀等间隔变化，而 multistep 则是根据 stepvalue 值变化
- poly: 学习率进行多项式误差, 返回 $base\_lr(1 - iter/max\_iter)^{power}$
- sigmoid: 学习率进行 sigmod 衰减，返回 $base\_lr(1/(1 + exp(-gamma * (iter - stepsize))))$

需要设置参数的数量随着 lr pilicy 的不同而有所变化。如设置为 fixed 则不需要添加任何参数，设置为 step 则需要添加 gamma 和 stepsize 两个参数，设置为 step 的策略后 solver 配置如下所示：

```
1          base_lr: 0.01
2          lr_policy: "step"
3          gamma: 0.9
4          stepsize: 100
```

4. 对于 momentum，一般取值在 0.5–0.99 之间。通常设为 0.9，momentum 可以让使用 SGD 的深度学习方法更加稳定以及快速。详细的资料，参考 Hinton 的论文《A Practical Guide to Training Restricted Boltzmann Machines》

5. type:SGD

表示优化算法，总共有六种：SGD、AdaDelta、AdaGrad、Adam、NAG、RMSprop

6.weight_ decay 为权重衰减项，详细的内容已经在上一章解释过了。

7.display：100 表示没训练 100 个 Iteration 显示一次 loss

8.max_ iter:2000 表示最大的迭代此时为 2000

9.

snapshot:500

snapshot_ prefix :"models"

表示没训练 500 个 Iteration，保存一次网络的参数数据，保存路径为 models。同时会保存另外一个 solverstate 文件，以便下次训练的时候可以从这一步继续训练。

10.solver_ mode: CPU 设置运行模式为 CPU

5.4 一些其他的问题

5.4.1 loss=87.3365

- 如果一开始就出现这种情况，有可能是参数的初始化方式不对，使用 xavier 可以避免这种情况
- 在 caffe 中数据的标签必须从 0 开始，且为整数（1.0,2.0 也可以）
- 数据可能会出现问题（一定要反复确定数据没有问题才可以）
- 如果是图片数据可能是图片没有进行归一化，最好是进行归一化并且减去均值
- 如果一开始的 loss 是在下降的，但是训练到后期出现了 87.3365 这个值，代表网络发散了，需要调整 learing rate 到一个较小的值，建议从 0.01 开始，到 0.0001 即可适合

大部分网络结构

5.4.2 一些差别

solver.step(1) 表示网络进行一次迭代，即进行一次前向过程和一次反向传播

solver.net.forward() 表示网络只进行一次前向的过程，可用于网络的预测

5.4.3 多 GPU 计算

如果使用 solver 文件：build/tools/caffe train –solver=models/bvlc_ alexnet/solver.prototxt –gpu=0,1

如果使用 Python 文件： python yourpythonfile.py CUDA_ VISIBLE_ DEVICES=0, 1

Bibliography