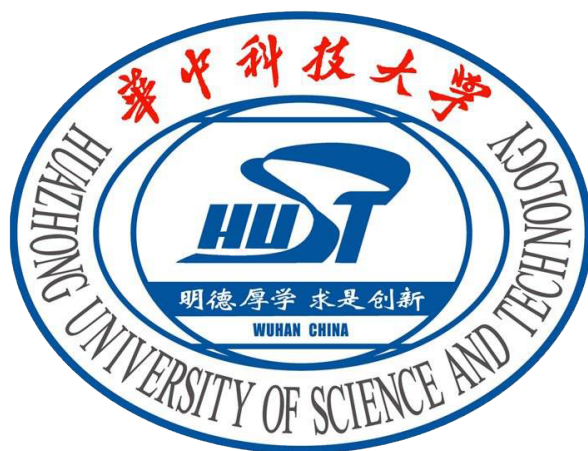


华中科技大学计算机科学与技术学院

《机器学习》结课报告



专 业	<u>计算机科学与技术</u>
班 级	<u>CS2208</u>
学 号	<u>U202215642</u>
姓 名	<u>田清林</u>
成 绩	<u> </u>
指导教师	<u>何琨</u>
时 间	<u>2024 年 5 月 18 日</u>

目录

1	实验要求	1
2	算法设计与实现	2
2.1	数据分析	2
2.2	softmax算法	3
2.2.1	算法原理	3
2.2.2	Min-Batch SGD+Momentum	4
2.2.3	正则化	4
2.2.4	softmax回归算法实现	4
2.3	K近邻算法(KNN)	5
2.3.1	距离的度量	6
2.3.2	k值的选择	6
2.3.3	kd树	6
2.4	KNN算法实现	7
3	实验环境与平台	8
3.1	环境	8
3.1.1	开发机器配置	8
3.1.2	开发系统	8
3.2	实验平台	8
4	结果与分析	9
4.1	参数调优	9
4.1.1	softmax参数调优	9
4.1.2	knn参数调优	9
4.2	模型评估	10
4.2.1	验证集	10
4.2.2	测试集	11
5	个人体会	12
5.1	项目总结	12
5.2	反思	12
5.3	学习体会	12
5.4	建议	12
	参考文献	13

1 实验要求

总体要求：

1. 模型训练需自己动手实现，严禁直接调用已经封装好的各类机器学习库（包括但不限于sklearn，功能性的可以使用，比如 `sklearn.model_selection.train_test_split`），但可以使用numpy等数学运算库（实现后，可与已有库进行对比验证）；
2. 使用机器学习及相关知识对数据进行建模和训练，并进行相应参数调优和模型评估；
3. 鼓励使用多种模型或不同数据集进行实验，并给出相应的分析思考；
4. 鼓励自主拓展探索；
5. 提交压缩包中包含（压缩包命名为“学号_姓名_机器学习结课.zip”）：
 - (a) 实验报告“学号_姓名_机器学习结课报告.pdf”；
 - (b) 代码文件；
 - (c) 预测结果；
 - (d) 额外用到的数据集等必要文件；
 - (e) “README.md”说明文件夹结构；
6. 严禁抄袭任何来源的代码或报告，一经发现大作业直接记 **0** 分处理；如需借鉴或引用，请标明出处；
7. 自行选择“.docx”、“.md”或“.tex”模板完成报告，最终提交“.pdf”格式报告。

2 算法设计与实现

本次要解决的问题obesity risk，输入数据是性别、身高、体重等描述一个人的特征，输出对其肥胖等级的判定，是一个多分类问题。

2.1 数据分析

输入数据相关信息如表2.1所示：

表 2.1: 输入数据信息

列名	数据类型	描述
id	int	每个人的唯一的序号
Gender	object	性别（男/女）
Age	float64	年龄（14.0至61.0）
Height	float64	身高
Weight	float64	体重
family_history_ with_overweight	object	是否有肥胖病史(yes/no)
FAVC	object	是否频繁食用高热量食物（yes/no）
FCVC	float64	食用蔬菜的频率
NCP	float64	正餐的数量（1到4）
CAEC	object	两餐之间进食的频率（no/sometimes/frequently/always）
SMOKE	object	是否抽烟（yes/no）
CH2O	float64	每天喝水的量（1 到3）
SCC	object	卡路里消耗量（yes/no）
FAF	float64	锻炼次数（0到3）
TUE	float64	使用电子设备时间（0-到2之间）
CALC	object	酒精摄入频率（Sometimes, no, Frequently）
MTRANS	object	出行交通方式:Automobile,Walking, Motorbike,Bike,Public_Transportation

可以看到，输入数据中既有离散值又有连续值，需要根据采用的模型来对数据进行处理。像CAEC、MTRANS这样有多个取值的离散型变量，在程序预处理时采用onehot编码，将其转化为多个0，1取值的变量。之后还要根据需要进行选择是否要进行标准化。

根据kaggle网站上的说明，该数据集是由一深度学习模型生成的，因此有不少从解释上应该为离散值（比如正餐的数量）的数据变成了连续值。

输出数据为肥胖类型，共有七个类型，如表2.2所示：

表 2.2: Weight Categories

Code	Category
0	Insufficient Weight
1	Normal Weight
2	Obesity Type I
3	Obesity Type II
4	Obesity Type III
5	Overweight Level I
6	Overweight Level II

使用LabelEncoder来将标签数据转化为0到6的整数值，可以看到各类别有明显的偏序关系。

可以采用logistic回归、svm并采取“one-vs-all”或“one-vs-one”的策略训练若干个模型，也可以使用softmax回归借助一个模型实现多分类，还可以采取合适的距离度量，用KNN算法求解。

如果将数据离散化，还可以采用朴素贝叶斯和决策树算法，但要注意数据离散化的方式，选择合适的边界。

当然还可以选择使用感知机算法、卷积神经网络等来解决问题。

经过综合考虑，本人选择采取softmax和KNN算法解决该多分类问题。

2.2 softmax算法

在机器学习课堂上讲到了一种处理二分类问题的方法，即逻辑回归。而softmax回归实际上是逻辑回归的一般形式，用于处理多分类问题。

2.2.1 算法原理

softmax算法是为了学习到一个模型，使其输出输入数据x归属于各类的概率，即

$$h_{\theta}(x_i) = \begin{bmatrix} p(y_i = 1|x_i; \theta) \\ p(y_i = 2|x_i; \theta) \\ \vdots \\ p(y_i = k|x_i; \theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^k e^{\theta_j^T x_i}} \begin{bmatrix} e^{\theta_1^T x_i} \\ e^{\theta_2^T x_i} \\ \vdots \\ e^{\theta_k^T x_i} \end{bmatrix}$$

其中 $\theta_1, \theta_2, \dots, \theta_k \in \theta$ 是模型的参数，前方乘 $\frac{1}{\sum_{j=1}^k e^{\theta_j^T x_i}}$ 是为了将概率归一化。

即输入数据 x_i 对于归属于类别j的概率如下： $p(y_i = j|x_i; \theta) = \frac{e^{\theta_j^T x_i}}{\sum_{l=1}^k e^{\theta_l^T x_i}}$

softmax 回归的参数矩阵为 θ , 有: $\theta = \begin{bmatrix} \theta_1^T \\ \theta_2^T \\ \vdots \\ \theta_k^T \end{bmatrix}$

2.2.2 Min-Batch SGD+Momentum

定义softmax其损失函数如下: $L(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m \sum_{j=1}^k 1\{y_i = j\} \log \frac{e^{\theta_j^T x_i}}{\sum_{l=1}^k e^{\theta_l^T x_i}} \right]$

求解其梯度为: (此处省略推导) $\frac{\partial L(\theta)}{\partial \theta_j} = -\frac{1}{m} \left[\sum_{i=1}^m x_i (1\{y_i = j\} - p(y_i = j|x_i; \theta)) \right]$

可用梯度下降法最小化损失函数。但是由于数据集较大, 每次迭代速度很慢, 此处采取小批量随机梯度下降+momentum的策略。

实现时, 采用线性衰减学习率直到第 τ 次迭代, 之后使学习率保持常数, 如下式所示:

$$\eta_k = (1 - \alpha)\eta_0 + \alpha\eta_\tau$$

SGD一定有一些缺点, 如果学习率太大, 容易造成参数更新呈现锯齿状的更新, 这是很没有效率的路径, 因此加入momentum, 类似为梯度下降添加惯性, 可以更好地趋近loss极小值。更新规则如下:

$$\mathbf{V}_t = \beta \mathbf{V}_{t-1} + (1 - \beta) \nabla_{\mathbf{w}_t} l$$

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta \mathbf{V}_t$$

β 是momentum项(一般设定为0.9), 主要是用在计算参数更新方向前会考虑前一次参数更新的方向, 如果当下梯度方向和历史参数更新的方向一致, 则会增强这个方向的梯度, 若当下梯度方向和历史参数更新的方向不一致, 则梯度会衰退。然后每一次对梯度作方向微调。这样可以增加学习上的稳定性(梯度不更新太快), 学习的更快, 并且有摆脱局部最佳解的能力。

2.2.3 正则化

为防止过拟合, 使模型过于复杂, 损失函数需加上一个正则项。此处选用L2正则, 修正后的损失函数如下所示:

$$L(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m \sum_{j=1}^k 1\{y_i = j\} \log \frac{e^{\theta_j^T x_i}}{\sum_{l=1}^k e^{\theta_l^T x_i}} \right] + \frac{\lambda}{2} \sum_{i=1}^k \sum_{j=1}^n \theta_{ij}^2$$

修正后的梯度如下所示:

$$\frac{\partial L(\theta)}{\partial \theta_j} = -\frac{1}{m} \left[\sum_{i=1}^m x_i (1\{y_i = j\} - p(y_i = j|x_i; \theta)) \right] + \lambda \theta_j$$

2.2.4 softmax回归算法实现

将softmax回归的主要实现放在了softmax类中。

softmax类的参数主要有学习率、正则项大小、迭代次数、每次随机梯度下降选择数据集的大小等。其成员函数如下所示：

1. `fit(self, X, y)`:使用 x 和 y 作为训练数据集训练模型，不断调用选用的更新函数来更新 w ，损失到达预期范围或迭代次数达到预期次数时停止训练。
2. `loss(self, X, y)`:根据 X 和 y 计算损失 $loss$ 和 w 的变化量 dw
3. `sample(self, X, y)`:在训练集中随机取样，返回大小为`batch_size`的数据集。
4. `predict(self, X)`:计算 $X*W$ ，对于每个测试集，给出计算出的概率最大的类别。
5. 以及三个用于更新参数的函数：
 - (a) `bgd(self, X, y)`:批梯度下降。
 - (b) `sgd(self, X, y)`:随机梯度下降，每次在测试集中选取一个较小的数据集来更新参数。
 - (c) `sgd_with_momentum(self, X, y)`:加入momentum优化的sgd。

模型训练过程中每次迭代均会调用`sgd_with_momentum(self, X, y)`函数，其实现代码如下：

```
1 def sgd_with_momentum(self, X, y):
2     X_batch, y_batch = self.sample(X, y)
3     loss, dw = self.loss(X_batch, y_batch)
4     self.velocity = self.momentum*self.velocity
5                     - self.learning_rate*dw
6     self.W += self.velocity
7     return loss
```

即先调用`sample`函数得到一个较小的数据集，再将得到的训练集作为参数调用`loss(self, X, y)`计算 $loss$ 和 dw 并返回，之后根据momentum算法，用 dw 更新 $velocity$ ，再用 $velocity$ 更新 W 。

2.3 K近邻算法(KNN)

K近邻算法(KNN)属于无参数模型，模型的参数随着训练数据的增加而增长。相较于有参数模型，无参数模型更加灵活，但需要更高的计算开销。

其基本思想在于“物以类聚，人以群分”。

- 假设：相似的输入有相似的输出

- 分类规则:对于一个测试样例即输入 x ，在它的 k 个最相似的训练输入中选择出现最多次数的标签作为输出
- 回归规则:输出是对象的一个属性值。这个值是 k 个最近邻值的平均值

2.3.1 距离的度量

k 近邻分类器的表现依赖于距离的度量。距离反映的相似性越好，分类器的表现就越好。最常见的选择是闵可夫斯基距离。

$$d(x_i, x_j) = (\sum_{k=1}^n |x_{ik} - x_{jk}|^p)^{1/p}$$

- 当 $p=1$ 时，就是曼哈顿距离
- 当 $p=2$ 时，就是欧几里得距离
- 当 $p \rightarrow \infty$ 时，就是切比雪夫距离

除此以外，还有余弦距离、汉明距离等距离。对于obesity-risk的多分类问题，使用 $p=2$ 的欧几里得距离。

2.3.2 k 值的选择

k 值同样影响分类器的表现，这里将 k 从1到50训练分类器并作出预测，选出表现最好的分类器对应的 k 值。

2.3.3 kd树

由于需要求解最近邻问题，如果使用暴力策略，算出某一点到训练集中全部点的距离再排序，耗时较长，这里使用kdtree对算法进行优化。

kd树是一种对 k 维空间内的实例点进行存储以便对其进行快速检索的树形数据结构，是一个二叉树，表示对 k 维空间的一个划分。

kd树的构造:

1. 构造根节点使其对应 k 维空间内包含所有实例点的超矩形区域。
2. 每次选择一个坐标轴和在此坐标轴上的一个切分点（一般采用实例点的该维的中位数）确定一个超平面，将超矩形区域切分为左右两个子区域（子结点）。
3. 直到子区域中没有实例时终止。

kd树的搜索:

1. 在kd树中找到包含目标点 x 的叶结点：从根结点出发递归地向下访问子树。若目标点 x 当前维的坐标小于切分点的坐标则移动到左子结点，否则到右子结点。直到到达叶结点，以此叶结点为“当前最近点”
2. 递归地向上回退，对每个结点查找是否存在与目标点最邻近的结点，并检查该结点的另一子节点对应的区域是否有更近的点，具体地，检查另一个子节点对应区域是否与以目标点为球心、以目标点与“当前最近点”间距离为半径的超球体相交。若相交，则移动到另一个子结点；若不相交，上回退。
3. 回退到根节点时搜索结束，最终的“当前最近点”即为 x 的最近邻点

2.4 KNN算法实现

为了实现KNN中kd树，本人建立了一个Treenode类，包括该结点的，标签，用来分隔的坐标序号 i ，分隔点的第 i 个坐标值，左子树和右子树。

将KNN算法的主要实现放在了KNN类中，该类成员主要有：选取邻居的数量 k ，选用的求解最近点的算法`algorithm`，(可选'brute'和'kdtree'),存储的`kdtree`(如果选用'kdtree'), 训练数据 x 和 y 。各成员函数如下：

1. `fit(self,x,y)`:使用 x 和 y 作为训练数据集训练模型，将输入的 x 和 y 存在`self.x`, `self.y`中，若使用算法为`kdtree`时执行`buildTree`。
2. `euc_dis(a,b)`:计算 a , b 之间的欧几里得距离
3. `isin(self,x,finded)`:查看 x 是否在`finded`内，两个点之间的欧氏距离小于一个极小的常数则说明是同一个点。
4. `buildTree(self,x,y,depth)`:使用数据集 x 和 y 在当前结点深度为`depth`时递归的建立`kdtree`
5. `find_nearest(self,x,finded)`:寻找不在`finded`内的距离 x 最近的点
6. `predict(self,test)`: 对`test`中每一个样例找到距离其最近的 k 个邻居并将出现次数最多的标签作为预测结果。

其中`buildTree`和`find_nearest`是模型中的重点，根据先前对于kd树构造和搜索的算法表述编写。

若采用`kdtree`算法，模型训练时会执行`buildTree(self,x,y,depth)`，预测时对于每个样例调用 k 次`find_nearest(self,x,finded)`，并将出现次数最多的标签作为预测结果。若采用`brute`算法，模型训练时只会将数据存入模型，预测时对于每个样例算出其到训练集中所有样例间的距离并排序，将距离最近的前 k 个样例中出现次数最多的标签作为预测结果。

3 实验环境与平台

3.1 环境

3.1.1 开发机器配置

- Processor: AMD Ryzen 7 6800H with Radeon Graphics 3.20 GHz
- Installed RAM: 16.0 GB (15.2 GB usable)
- Device ID: 78059776-C668-4BAF-A511-3A78D26ACD44
- Product ID: 00342-30659-80470-AAOEM
- System type: 64-bit operating system, x64-based processor

3.1.2 开发系统

- Edition: Windows 11 家庭中文版
- Version: 23H2
- Installed on: 2022/ 11/ 7
- OS build: 22631.3447
- Experience: Windows Feature Experience Pack 1000.22688.1000.0

3.2 实验平台

使用IDE为VScode,版本为1.89.0, Python版本为 3.11.9

4 结果与分析

4.1 参数调优

4.1.1 softmax参数调优

softmax回归的超参数主要有：学习率`learning_rate`、最大迭代次数`iters`、每次随机梯度下降使用的训练集的大小`batch_size`、正则项大小 λ 、`momentum`值。

其中`momentum`取经验值0.9，表现较为优良。

对于其他四个参数进行参数调优过程中，采取网格搜索的方式：

首先从估计值(0.5,1000,50,1e-4)开始以较大步长进行搜索，根据模型表现确定各参数的粗略值，之后在附近缩小步长进行精细化搜索，最终找到一个合理的值。

在训练过程中，每次迭代后程序都会输出此时的loss值，多次改变参数后可以看出，当学习率过大时的loss会大幅震荡，不能收敛，在最优值附近徘徊，但有时能在若干次震荡后找到正确的学习方向；学习率过小时loss会较慢、较稳定地减小，可能会进入局部极值点就收敛，没有找到真正的最优解。

4.1.2 knn参数调优

由于此处距离函数直接选择了欧几里得距离，因此只需确定超参数 k 。对 k 在1到50时分别训练模型并测试其性能，如图4.1所示。

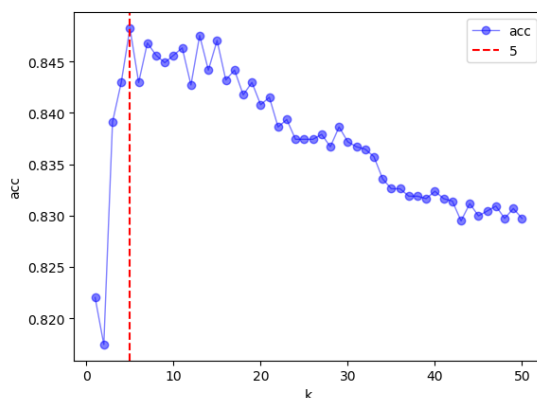


图 4.1: knn在不同k值下的训练准确率

更小的 k 值减小近似误差而放大估计误差，容易过拟合，而更大的 k 值放大近似误差减小估计误差，模型更简单，容易欠拟合。

随着 k 增大，模型准确率整体上呈现先上升后下降的趋势，与理论相符。

可以看出，在 $k=5$ 时模型在训练集上表现最好，若是综合考虑模型准确率和运行效率， $k=3$ 或4也是较为可行的。

4.2 模型评估

本人共实现了两个模型，分别在训练集划分出的验证集和kaggle官网上测试集进行模型评估，由于是多分类问题，采取accuracy和各类别的precision、recall、f1-score来评估模型性能，由于KNN算法时间效率较差，将两个模型执行的总时间也用于模型性能评估。

4.2.1 验证集

采取留出法，将数据集划分为训练集和验证集，两者占比分别为80%和20%，重复执行评估后算出平均值。Softmax表现如表4.1所示，KNN表现如表4.2所示。

表 4.1: Softmax表现

id	类别	precision	recall	f1-score	accuracy	support
0	Insufficient Weight	0.87	0.93	0.90		524
1	Normal Weight	0.84	0.79	0.82		626
2	Obesity Type I	0.81	0.83	0.82		543
3	Obesity Type II	0.96	0.97	0.97		657
4	Obesity Type III	0.99	1.00	0.99		804
5	Overweight Level I	0.71	0.73	0.72		484
6	Overweight Level II	0.72	0.67	0.69		514
total					0.86	4152

表 4.2: KNN表现

id	类别	precision	recall	f1-score	accuracy	support
0	Insufficient Weight	0.84	0.90	0.87		524
1	Normal Weight	0.78	0.74	0.76		626
2	Obesity Type I	0.80	0.82	0.81		543
3	Obesity Type II	0.96	0.96	0.96		657
4	Obesity Type III	0.99	0.99	0.99		804
5	Overweight Level I	0.69	0.67	0.68		484
6	Overweight Level II	0.70	0.70	0.70		514
total					0.84	4152

此外，考虑模型运行时间，softmax平均用时为0.97s，而KNN平均用时为443.27s。可以看出：

1. 对于softmax和KNN模型，模型对于Obesity Type II和III的分类表现极好，在Insufficient Weight、Normal Weight和Obesity Type I上的表现一般，而对于两个Overweight类型的数据表现不尽人意。各类别中precision、recall、f1-score均遵循类似的分布，无异常值。
2. 比较softmax模型与KNN模型，可以看到softmax模型准确率略优于KNN模型，差距较小，但在运行时间上，KNN算法时间复杂度远超Softmax回归。

猜测原因主要有以下几点：

1. 两模型都没有意识到一个先验信息：7个类别之间不是独立的，而是具有明显的偏序关系。也许使用决策树算法可以表现更好。
2. 客观上，准确区分各类别的难度不同，尤其是两个Overweight类别特征不明显。
3. 没有进行主成分分析，输入数据中存在噪声和不重要的特征。
4. 关于KNN模型用时过长，当然这是KNN模型本身的特点，但也可能存在数据维度过多、数据分布较紧密以至于在树上递归搜索次数过多的问题。

4.2.2 测试集

在kaggle官网上进行测试并得到准确率，softmax和KNN准确率如图4.2所示



Submission and Description		Private Score ⓘ	Public Score ⓘ	Selected
 submission_softmax.csv Complete (after deadline) · 13s ago		0.85233	0.85693	<input type="checkbox"/>
 submission_knn.csv Complete (after deadline) · 41s ago		0.84754	0.84826	<input type="checkbox"/>

图 4.2: softmax与KNN在kaggle数据集上的准确率

两个模型的准确率均大于80%,且两个模型在测试集中的表现与在验证集中的表现基本相同，可见两模型均有较好的泛化性能。

5 个人体会

5.1 项目总结

本次项目是老师给出的作业四个选题之一，来自于kaggle。当时选择这个多分类问题主要是考虑到自己对于多分类问题的解决经验较少，期望通过项目获得提升，最终确实受益匪浅。

在系统的实现上，代码分为数据读入、预处理、训练模型、测试模型、输出结果五个部分，我按照类似sklearn的形式将一个模型包装在一个类中，以此使整个系统分为便于理解，也便于修改和维护。

在实现系统的过程中，我加深了对所采用算法的理解，也遇到并解决了在平时理论学习中没有想到的各种实现上的问题，对python的numpy库的使用也更加熟练。

5.2 反思

虽然本次项目的基本目的达到了，但是回首反思这给项目依旧存在诸多可以进一步改进和优化的地方。

当时之所以选择knn算法，是因为sklearn中的knn模型(算法选用kd树)不仅预测速度很快且模型表现满足要求。但在本人的knn模型的实现中，虽然为了提高预测效率实现了kd树，但是受限于实现方式和数据维度，在 $k = 5$ 时最终预测效率甚至比不上暴力算法，没有达成优化的目的。

其实对于多分类问题还有很多机器学习算法，比如决策树，但是受限于时间，最终没有实现。

5.3 学习体会

机器学习这门课涵盖了大量数学原理和统计学知识，为此本人重温了不少微积分、统计学和线性代数的知识。虽然很多理论推导让人一头雾水，但证明后的成就感令人着迷。同时该课程具有实践性，课堂上关于调试模型和评测模型的知识实实在在地帮助我完成大作业，提高了我的理论学习能力。

尽管在最后的项目中，我只实现了一些基础的、较为简单的算法，但这加深了我对算法的理解，我认为这是学习机器学习的重要一步。未来，我将继续深入学习和研究这个领域，希望能用来解决实际生活的问题。

5.4 建议

我认为课程各章节之间的联系较少，随着课程的进行，我见识到了各种机器学习算法，却缺少对他们之间联系的认识，只是浮于表面，没有建立起完备的知识体系。

参考文献

- [1] 周志华. 机器学习:第3章. 清华大学出版社, 2016.
- [2] 李航. 统计学习方法:第3章. 清华大学出版社, 2019.