基于机器学习的文本分类

日期：2023 年 3 月 12 日

摘　要

本次任务，我使用 numpy 实现了基于 logistic/softmax regression 的文本分类。首先分析任务数据集，后续根据模型使用部分数据设置多组实验，并且在一组中设置多个学习率，训练多个模型，探究不同训练策略、特征提取方法等的效果差异。最后根据实验中得到的结论，使用所有数据训练最佳模型，最终使用 4-Gram 方法在 batch_size 为 400 的 mini-batch 方法中训练到第 122 个 epoch 出现了最佳模型，在验证集上准确率达 59.3778%，在 Kaggle 上的得分为 0.38211。最后就得到的最佳模型做了实例验证分析。

关键词： 逻辑回归，文本分类，N_Gram

1　问题简述

本次任务为 Kaggle 比赛Sentiment Analysis on Movie Reviews，比赛提供了烂番茄电影评价数据集，且要求在五个尺度上标注短语：消极、有点消极、中性、有点积极、积极。其中句子否定、讽刺、简洁、语言歧义等障碍使这项任务极具挑战性。本任务要求使用 numpy 实现基于 logistic/softmax regression 的线性回归模型，探究短句和分类标签之间的映射关系，实现对句子中短语的细粒度情感分析。

	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	A series of escapades demonstrating the adage ...	1
1	2	1	A series of escapades demonstrating the adage ...	2
2	3	1	A series	2
3	4	1	A	2
4	5	1	series	2

图 1: train.tsv 部分数据

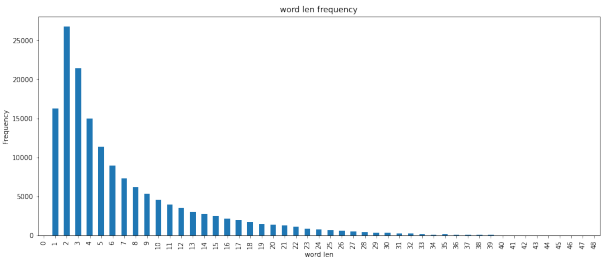

图 2: train.tsv 短语中单词个数频率统计

2　数据集分析

首先使用 pandas 读取 train.tsv 和 test.tsv 文件，并在 jupyter 中观察如图 1 所示数据。数据有 PhraseId、SentenceID、Phrase 以及 Sentiment 四属性。图中显示的前 5 个短语均来自 SentenceId 为 1 的句子，且相互之间存在重复部分。本次的任务即为探究 Phrase 属性和 Sentiment 属性之间的关系，并通过该关系预测 test.tsv 中不同 Phrase 对应的 Sentiment。

后续使用正则表达式，统计了 Phrase 中单词数量。发现存在一些全是特殊字符如'?!?'、'...' 等短句，此类短句在 train.tsv 中 Sentiment 类别为 2，但由于 test.tsv 中同样存在该类全由特殊字符组成的，故保留 train.tsv 中此类训练数据。除此之外，还有部分如'An'、'A' 等冠词存在，但分别所属的 Sentiment 类别不同。经过对短语词长的统计，得到如图 2 所示的柱状图，发现大多数短语单词组成个数不超过 5，单词最多的一个短语包含 48 个单词。

同样测试集中也呈现类似的单词个数分配，其中最长的短句包含 52 个单词。

3　模型介绍

3.1　特征提取方法

Bag-of-Word: Bag-of-words 模型是信息检索领域常用的文档表示方法。在信息检索中，BOW 模型假定对于一个文档，忽略它的单词顺序和语法、句法等要素，将其仅仅看作是若干个词汇的集合，文档中每个单词的出现都是独立的，不依赖于其它单词是否出现。Bag-of-words 首先提取出文本独特的 word，然后给每个 word 赋予一个 one hot vector。

N-Gram: N-Gram 是一种基于统计语言模型的算法。它的基本思想是将文本里面的内容按照字节进行大小为 N 的滑动窗口操作，形成了长度是 N 的字节片段序列。每一个字节片段称为 gram，后续整合所有 gram 作为词汇集合，按照 Bag-of-Word 的方法为每一个 gram 赋予 one hot vector，并根据句子中是否出现该 gram 置 0 或者 1。在本任务中，N-Gram 的词汇集合包括从 1 到 N 的所有 gram。

3.2 训练集拆分方法

在常见的深度学习中，数据通常被拆分为训练集、验证集和测试集三部分。训练集用于模型训练更新参数；验证集用于在模型完成一个阶段的训练后检验模型效果以及是否发生过拟合，也有实验使用验证集调整超参数以达到模型最优效果；测试集用于最后评估模型的效果。在本任务中 train.tsv 文件即被用于拆分为训练集和验证集，test.tsv 即为用于最后评估模型效果的测试集。

在从 train.tsv 拆分训练集和验证集时，考虑到相邻短语取自同一句子，可能含有较大重复并且在一些深度学习中包装 batch 的 DataLoader 在训练集中通常选择使用 shuffle 打乱数据，因此尝试使用 random 函数模拟此过程，并与直接的顺序拆分做了实验对比，这部分在 4.2 节中有具体阐述。

顺序拆分: 根据训练集占比以及验证集占比将 train.tsv 分为两部分，将中前部分作为训练集，后部分作为验证集。一般情况下验证集的占比不应大于训练集，一般设置为 30% 已经较大。

随机拆分: 通过 random.sample 函数从总数据中随机取样，先抽取训练集数据后深度拷贝一份数据，再剔除训练集中数据，再次为验证机随机抽取。抽取数量由训练集占比以及验证集占比决定。

3.3 激活函数

基于线性回归模型实现分类，本质上为使用权重矩阵计算出样本属于每类别的分数，在最后对各项分数使用激活函数规范到 0-1 之间，将最后归一化的分数作为样本属于该类的概率。对于最后激活函数的选择，很多论文针对多分类问题都是使用 softmax 函数，因为其计算得到每一个类别概率之和为 1，是对所有类别的整体建模，每个类之间不独立。而 Logistic 函数——sigmoid 是单独对每个类别的建模，单独计算属于该类别的概率，各个类之间相互独立，概率之和不为 1。最后选择预测的类别都是使用 argmax 选取最大概率类别作为预测结果，但由于 softmax 和 sigmoid 函数本身都是增函数，在归一化各类得分时并不会改变各得分之间的相对大小关系，因此并不对预测结果产生影响，甚至若只为求得分类预测，不考虑解释模型的预测结果，可以不使用最后的归一化。从理论来所，我认为这两种不同激活函数并不会影响模型训练结果，后续在 4.3 节也对此做了实验对比验证。

3.4 训练策略

机器学习中常见的训练策略有 shuffle、batch、mini-batch 三种方式，在深度学习中使用较多的都是 mini-batch。pytorch 中的 DataLoader 其内部封装的就是 mini-batch 这种训练方式，除此之外，mini-batch 方法也是 shfflle 和 batch 这种方法的折中，尽可能地减轻了两种方法的缺点。因此对此认为 mini-batch 会是本任务效果最佳的训练策略，这部分在 4.4 节中有具体实验验证。

shuffle: 循环 $train\_times$ 轮，每轮随机不放回地抽取一个样本计算 loss 并更新矩阵参数，该方法因为频繁更新矩阵参数因此训练速度较慢。

batch: 外层循环 $train\_times$ 轮，内层再遍历所有样本，内层遍历时依次计算 loss 并累加，最后累加完所有样本 loss 后使用总 loss 更新矩阵参数，该方法因为更新次数较少，容易出现局部最优的情况。

mini-batch: 外层循环 $train\_times$ 轮，内层再循环 $mini\_size$ 轮，其中每轮随机不放回地抽取一个样本计算 loss 并累加，完成一个 minibatch 后统一根据累加的 loss 更新矩阵参数，可以根据 mini_batch 的选取避免局部最优的同时加快训练速度。

为了后续实验对比对于每种训练策略的公平性，不同策略的 $train\_times$ 有如下的计算方式：

$$train\_times = \begin{cases} num_{item} * epoch, & \text{shuffle} \\ epoch, & \text{batch} \\ int(\frac{num_{item}}{batch\_size}) * epoch, & \text{mini-batch} \end{cases}$$

公式中 num_{item} 指训练使用的样本数量，$epoch$ 指训练中使用所有数据的轮数 (epoch 为 2 即说明使用全部数据训练两次)，$batch\_size$ 指 mini-batch 训练策略中一个 batch 的样本数量。

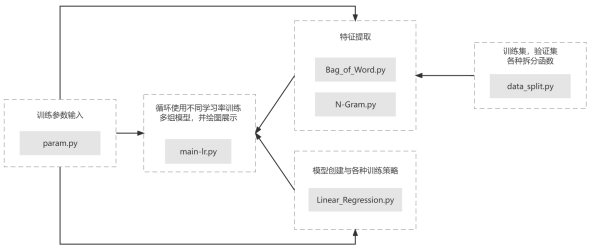

图 3: 各实验 python 脚本介绍

3.5 损失函数

对于分类问题，因为标签的离散性，通用的损失函数为交叉熵。除此之外对于连续形函数常用的还有 L2 损失函数。但在实际参数更新时只需要损失函数对参数求导的函数来更新参数，且两者求导后分别为：$\frac{1}{N}\sum_{i=1}^{N}\left(\mathrm{x}^{(i)}\cdot\left(\sigma\left(\mathrm{w}^{\mathrm{T}}\mathrm{x}^{(i)}\right)-y^{(i)}\right)\right)$ 与 $\frac{2}{N}\sum_{i=1}^{N}\left(\mathrm{x}^{(i)}\cdot\left(\sigma\left(\mathrm{w}^{\mathrm{T}}\mathrm{x}^{(i)}\right)-y^{(i)}\right)\right)$，式子中 σ 表示 softmax 或者 sigmoid 函数，后续将其整体替换为 $\hat{y}_i$。除去定量后都为该式：$\sum_{i=1}^{n}(\hat{y^{(i)}}-y^{(i)})$，而被除去的定量也都可以通过改变学习率的方式达到相同效果，因此并未设置不同损失函数的实验。

3.6 整体代码实现

实验整体代码使用 6 个 python 脚本，一个 jupyter 文件。其中 jupyter 文件用于最初观察分析数据集。python 脚本用于进行各种实验。各 python 脚本的作用以及关联如图 3 所示。

根据本次任务的不同流程 (数据拆分、特征提取、模型训练以及结果分析) 将对应部分进行封装，由 main-lr.py 文件进行整体调用。图中箭头指向指在所指板块调用了原板块的代码，如被调用最多的 param.py 脚本，使用 argparse 库，汇总来自终端输入的参数，在实验中根据参数在不同板块使用不同的策略或方法。

4 实验

实验设计通过终端运行 main-lr.py，并传入相应实验参数进行实验。每轮实验使用多个学习率，取最优准确率作为实验结果，同时因为 N-Gram 特征提取方法在实验中运行时间较长，在其余的实验中统一使用 Bag-of-Word 方法控制变量。

4.1 不同学习率

统一使用随机拆分法、抽取 30000 个样本数据、Bag-of-Word 特征提取方法、softmax 作为激活函数、训练 7 个 epoch、使用 mini-batch 训练、设置 batch_size 为 100、验证集占比为 0.2 以及交叉熵损失函数，实验结果如表 1 所示[1]。

表 1: 不同学习率对模型效果影响

学习率取值区间	准确率 (%)	最佳学习率
0.01,0.1,1,2,3	24,26,32,37,38	3
6,7,8,9,10,11,12	41,42,42,43,42,43,42	9
12,15,17,19,20	43,45,43,45,44	19
20,21,22,23,24,25	44,45,43,45,44,45	23
26,28,30,32,34,35	44,46,44,45,44,44	**28**
32,35,38,40,42,45	43,45,44,45,44,44	35
20,100,1000,3000	44,43,43,44	3000

可见，学习率较小容易导致模型无法达到较优效果，在学习率为 28 时模型达到最佳效果，当继续增大学习率时模型有轻微下降后不在有明显变化。据此后续实验使用 26,27,28,29,30,31,32 这组学习率，并认为模型最适合的学习率会取自该组。

4.2 不同训练集拆分方法

统一使用 26,27,28,29,30,31,32 几组学习率、抽取 30000 个样本数据、Bag-of-Word 特征提取方法、softmax 作为激活函数、训练 7 个 epoch、使用 mini-batch 训练、设置 batch_size 为 100、验证集占比为 0.2 以及交叉熵损失函数，实验结果如表 2 所示。

表 2: 不同训练集拆分方法对模型效果影响

训练集拆分方法	准确率 (%)	最佳学习率
random_split	44,46,44,45,45,44,42	27
ordered_split	36,38,37,34,35,39,39	31

从实验结果来看与 3.2 中的观点一致，使用随机拆分训练的模型准确率更高。

[1]在做此实验之前已经尝试过一些学习率，在此仅展示部分有良好效果的。准确率一项为保留小数后的结果，最佳学习率为该组实验中保留之前的最佳. 后续实验表格均参照此规则

4.3 不同激活函数

统一使用随机拆分法、统一使用 26,27,28,29,30,31,32 几组学习率、抽取 30000 个样本数据、Bag-of-Word 特征提取方法、训练 7 个 epoch、使用 mini-batch 训练、设置 batch_size 为 100、验证集占比为 0.2 以及交叉熵损失函数，实验结果如表 3 所示。softmax 和 sigmoid

表 3: 不同不同激活函数对模型效果影响

激活函数	准确率 (%)	最佳学习率
random_split	44,46,44,45,45,44,42	27
ordered_split	45,46,44,45,45,45,44	27

从实验结果可见使用两种激活函数，模型均能达到 46% 的准确率，与 3.3 节观点一致，两种激活函数对模型准确率并不产生影响。

4.4 不同训练策略

随机拆分法、统一使用 26,27,28,29,30,31,32 几组学习率学习率、抽取 30000 个样本数据、Bag-of-Word 特征提取方法、softmax 作为激活函数、训练 7 个 epoch、验证集占比为 0.2 以及交叉熵损失函数，实验结果如表 4 所示。

表 4: 不同训练集拆分方法对模型效果影响

训练集拆分方法	batch_size	准确率 (%)
shuffle	-	43,44,42,44,44,43,42
batch	-	30,30,29,31,30,32,31
mini-batch	50	44,45,43,44,43,45,41
mini-batch	80	45,46,43,45,45,46,42
mini-batch	100	44,46,44,45,45,44,42
mini-batch	150	44,42,40,45,42,42,43
mini-batch	200	44,44,44,44,44,44,44
mini-batch	400	36,39,43,37,42,41,43

从实验结果可得，与 3.4 中的观点一致，mini-batch 效果优于 shuffle 和 batch，其中 shuffle 的训练效果优于参数更新数量较少的 batch 训练策略。而对于 mini-batch 策略的不同 batch_size，取值较大就会导致参数更新次数下降，会出现类似 batch 训练策略的问题，取值较小就会出现类似 shuffle 策略的效果。整体来说对于 30000 条样本数据，选择 batch_size 为 80 左右，使参数在 7 次 epoch 中更新 2625 次左右会有较好效果。

4.5 不同特征提取方法

随机拆分法、统一使用 26,27,28,29,30,31,32 几组学习率学习率、抽取 30000 个样本数据、softmax 作为激活函数、训练 7 个 epoch、使用 mini-batch 训练、设置 batch_size 为 100、验证集占比为 0.2 以及交叉熵损失函数，实验结果如表 5 所示[2]。在这两种方法上也尝试过使用 gram 在句子中出现频数改进是否出现的 0-1 矩阵，但是结果并没有原始 0-1 矩阵准确率高，在此并未展示结果。

表 5: 不同特征提取方法对模型效果影响

特征提取方法	词长 (对于 N-Gram)	准确率 (%)
Bag-of-Word	-	44,46,44,45,45,44,42
N-Gram	2	45,46,46,47,45,47,45
N-Gram	3	45,47,46,47,45,46,46
N-Gram	4	44,46,46,46,46,47,47

从实验结果分析，词长大于 1 时使用 N-Gram 方法，能出现优于 Bag-of-Word 方法的准确率结果。而随着词长的增大，模型效果并未出现明显的提升。

5 最佳模型

在此使用前面实验中的结论做出部分改进，并使用所有 156060 条数据训练模型。在该部分固定使用 N-Gram 特征提取方法，且在不断实验中发现增加词长 N 到 4 对模型准确率有显著提升。并且不断增加训练 epoch，模型在验证集上仍然会有一些提升。目前训练出最好的模型使用所有参数，随机拆分，4-Gram 方法，batch_size 为 400，使用 softmax 激活函数，验证集占比为 0.2，一共训练 125 个 epoch，训练耗时 18 小时左右，在第 122 个 epoch[3]处达到最佳效果，最终在 train.tsv 随机拆分 20% 的验证集中准确率达 59.3778%，在Kaggle上评分 0.38211。后续也考虑过训练样本中各分类不均衡的影响，对占

[2]当 N-Gram 方法中 N 为 1 时等同于 Bag-of-Word 方法，因此未设置该项实验

[3]该结果是在总共 125 个 epoch 上得到的结果，有理由认为继续增加 epoch，模型准确率可能有进一步提升，但后续因为训练耗时过大，并未训练更多轮次

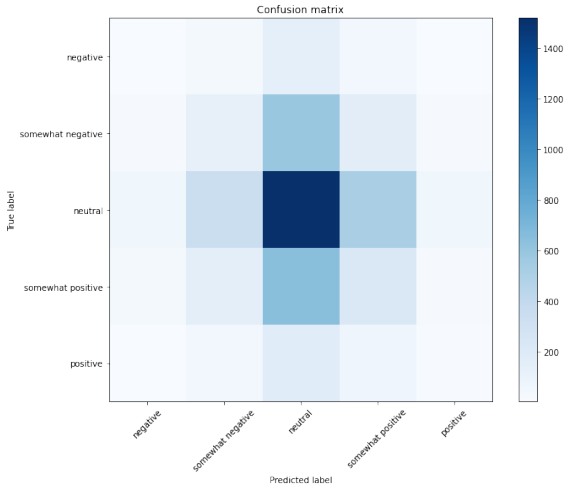

图 4: 随机抽取 5000 条样本预测混淆矩阵热力图

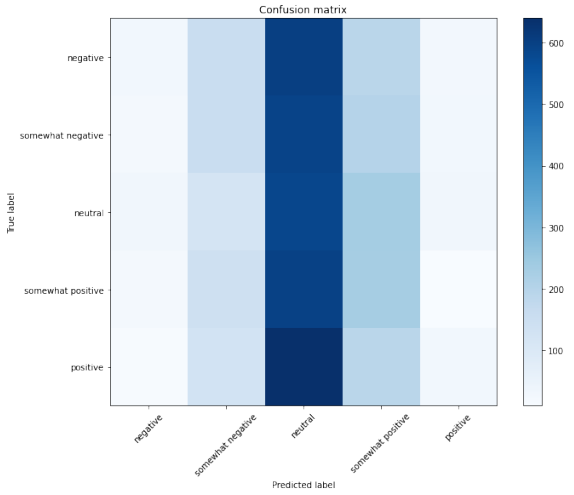

图 5: 平均抽取 5000 条样本预测混淆矩阵热力图

比最多的标签为 2 的样本随机取样 30000 条，与标签 1、3 两类数据样本接近后再进行训练，结果与最佳模型在相同训练条件下，在验证集上准确率为 54.1536%。

6 模型验证

后续使用最后训练出的最佳模型进行验证。首先从 train.tsv 文件中随机抽取 5000 条数据，使用模型进行预测后将结果与真实分类标签对比，并使用 sklearn 库的 confusion_matrix 函数以及 matplotlib 库进行可视化，结果如图 4 所示。验证中模型准确率为 37.8%，且从热力图看出多是因为将占比较大的 neutral 分类预测正确，同时也发现模型容易将 somewhat positive 和 somewhat negative 类别的样本预测为 neutral，而 positive 和 negative 分类则极少有正确分类的。

接着我又尝试了平均对每个类别取样 1000 份，并采用与上面验证相同的步骤，得到如图 5 所示的混淆矩阵热力图。验证中模型准确率为 19.9% 发现确实是因为模型将较大部分数据预测为 neutral，且存在将部分原本是 neutral 的样本预测为 somewhat positive 和 somewhat negative 的现象。

因此得出最后的结论，模型整体效果并不理想，模型将较大部分样本预测为 neutral 类，一定程度上是因为训练样本中 neutral 类样本约占总样本的 50% 且训练前并未对数据做采样方面的预处理。考虑到在训练最佳模型时，尝试过随机筛选类别为 neutral 的样本 30000 条与其他样本一起训练，结果在训练中的验证集上表现不如使用完整数据[4]。

7 反思与总结

本次任务我使用 pandas 对数据进行了初步的分析，后续使用 numpy 搭建了整个 logistic/softmax regression 模型，并结合 argparse 库实现便捷化实验。本次实验也是我第一次从底层实现一个机器学习模型，先前的学习中多是直接使用 sklearn 中的包或是在深度学习中直接使用 PyTorch 框架中写好的优化器，梯度更新，计算等。在实验中，许多结论与先前的知识相符合，也让我对 mini-batch 等方法有了更深的理解。

对于本模型的表现，本模型主要在特征提取、模型本身性质和训练数据分布上受到制约。在特征提取方面，Bag-of-Word 方法和 N-Gram 方法并不能完善提取一个句子的语意特征；在模型本身性质上，模型无法拟合非线性的函数关系；在训练数据分布上，模型为了提升准确率，“故意”将大部分数据预测为占比较大的 neutral 类别。

由于模型在训练 125 个 epoch 时，在第 122 个 epoch 达到最佳效果，可能并未达到模型的上限，继续训练的效果可能是对 neutral 类别的分类更为准确，对于其他如 positive 或者 negative 类样本占比较少的分类效果可能最后仍然强差人意。

[4]随机抽样 30000 份，训练出的模型因为构建的语言模型不同，模型参数维度也与使用全部样本的模型参数维度不同，当时训练的 4-Gram 语言模型无法复现，因此无法使用该模型对 test.tsv 文件进行预测