基于朴素贝叶斯的 MNIST 分类

前言

环境信息。本实验在仅能使用 `torch` 库的基础上进行，`torch` 版本：`2.3.1+cpu`。

数据信息。本实验使用经典的 MNIST 数据集进行朴素贝叶斯 (Naive Bayes) 分类任务。数据集已提前划分为 4 部分，分别为：

- 60000 条训练集的特征 `train-images.idx3-ubyte` 和标签 `train-labels.idx1-ubyte`；
- 10000 条测试集的特征 `t10k-images.idx3-ubyte` 和标签 `t10k-labels.idx1-ubyte`。

文件后缀中 `idx{x}` 表示含有 x 个 `rgb` 维度信息，`ubyte` 表示存储格式为二进制。

一、数据预处理

封装 `DataProcess` 类。

1.1 加载数据

封装 `def __init__(self, images_path: str, labels_path: str, nums: int) -> None:` 方法。

由于使用了二进制的方式存储数据，因此采用内置函数 `open()` 加上 `binary mode` 模式进行读取。查阅资料可知 MNIST 数据集的前几行属于魔数信息，不应当被作为有效数据进行使用，使用 `read()` 方法直接跳过即可。具体样本数官网已明确指出，直接读取 60000 条训练样本，10000 条测试样本即可。

1.2 张量展平

封装 `def reshape(self) -> None:` 方法。

在数据处理之前我们往往需要进行异常值检测。但是由于数据集官方已经明确指出数据的规格是 $60000 \times 28 \times 28$ 和 $10000 \times 28 \times 28$，因此本数据集我们无需进行数据异常值处理。可以**直接进行**相关预处理操作。首先我们需要将三维的张量数据展平为二维的矩阵数据，便于后续朴素贝叶斯算法的训练与测试。我们直接使用 `torch`

库自带的 `view()` 方法进行展平，展平后的数据规格是 60000×784 和 10000×784，我们成功获得了二维的矩阵信息。

1.3 二值化处理

封装 `def binary(self) -> None:` 方法。

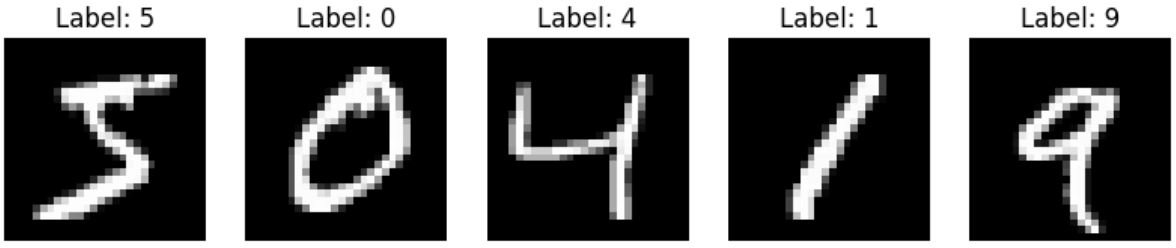
由于数据仅仅是灰度图像，只有一个 `rgb` 通道，因此我们作出假设：二值图像比灰度图像的识别准确率更高。当然这样的假设并非空穴来风。由于朴素贝叶斯的概率采用条件联合概率，这就导致过多取值的属性连乘 $\prod$ 时容易造成浮点数的下溢，而采用二值化后的属性取值数量从可能的 `[0,255]` 降低到 `[0,2]`，这有利于计算机进行浮点数运算而不会造成下溢。而这也为后续算法超参的选择进行了铺垫。

1.4 数据可视化

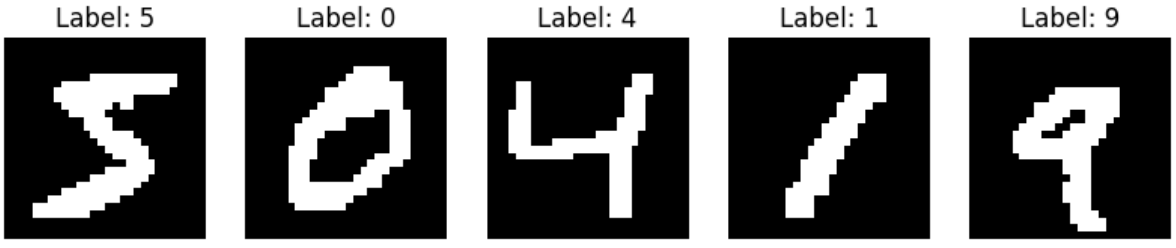
封装 `def show_data(self, bin: bool = False, cnt: int = 5) -> None:` 方法。

为了更好的阅读手写数据集的信息，本项目将数据加载完毕以后，封装了数据可视化的方法。可以直接使用 `show_data()` 方法将手写信息可视化展示出来。分别以训练集和测试集的前五个数据为例进行展示，效果如下：

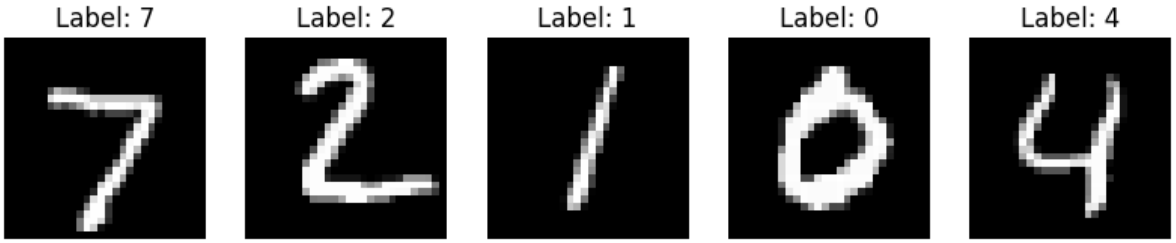
训练集前 5 个数据可视化 - 灰度图像：



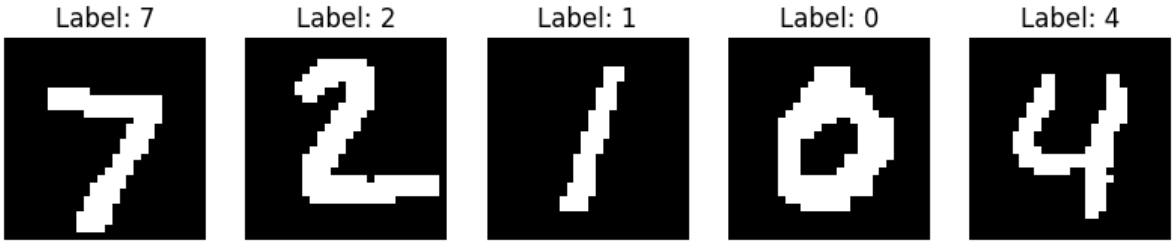
训练集前 5 个数据可视化 - 二值图像：



测试集前 5 个数据可视化 - 灰度图像：



测试集前 5 个数据可视化 - 二值图像：



二、模型构建

封装 `MyNaiveBayes` 类。

2.1 初始化模型

封装 `def __init__(self, opt: str = "log") -> None:` 方法。

定义超参为：是否使用「取对数」处理。并定义 4 个类变量，如下：

- `self.opt = opt`：数据类型为 `str`，表示**超参选项**；
- `self.classes = None`：数据类型为 `torch.tensor(x)`，表示**类别标签**。`self.classes[i]` 表示第 `i` 个标签的类别；
- `self.cla_pri = None`：数据类型为 `torch.tensor(x)`，表示**类别先验概率**。`self.cla_pri[i]` 表示第 `i` 个类别的先验概率；
- `self.feat_prob = None`：数据类型为 `torch.tensor((x, y))`，表示**特征条件概率**。`self.feat_prob[i][j]` 表示第 `i` 个类别第 `j` 个取值的条件概率。

2.2 定义训练方法

封装 `def fit(self, train_X: torch.Tensor, train_y: torch.Tensor) -> None:` 方法。

通过训练集计算类别先验概率向量 `self.cla_pri` 和特征条件概率矩阵 `self.feat_prob`。

2.3 定义预测方法

封装 `def predict(self, test_X: torch.Tensor) -> torch.Tensor:` 方法。

懒惰学习的代表，直接查表计算结果即可。

使用「不取对数」的计算公式为：

$$P(c \mid x) = \frac{P(c)P(x \mid c)}{P(x)} = \frac{P(c)}{P(x)} \prod_{i=1}^{d} p(x_i \mid c)$$

使用「取对数」优化的计算公式为：

$$\begin{aligned} \log P(c \mid x) &= \log \frac{P(c)P(x \mid c)}{P(x)} \\ &= \log \frac{P(c)}{P(x)} \prod_{i=1}^{d} p(x_i \mid c) \\ &= \log \frac{P(c)}{P(x)} + \sum_{i=1}^{d} \log p(x_i \mid c) \end{aligned}$$

三、模型测试与评估

分为 4 种情况进行测试与评估：

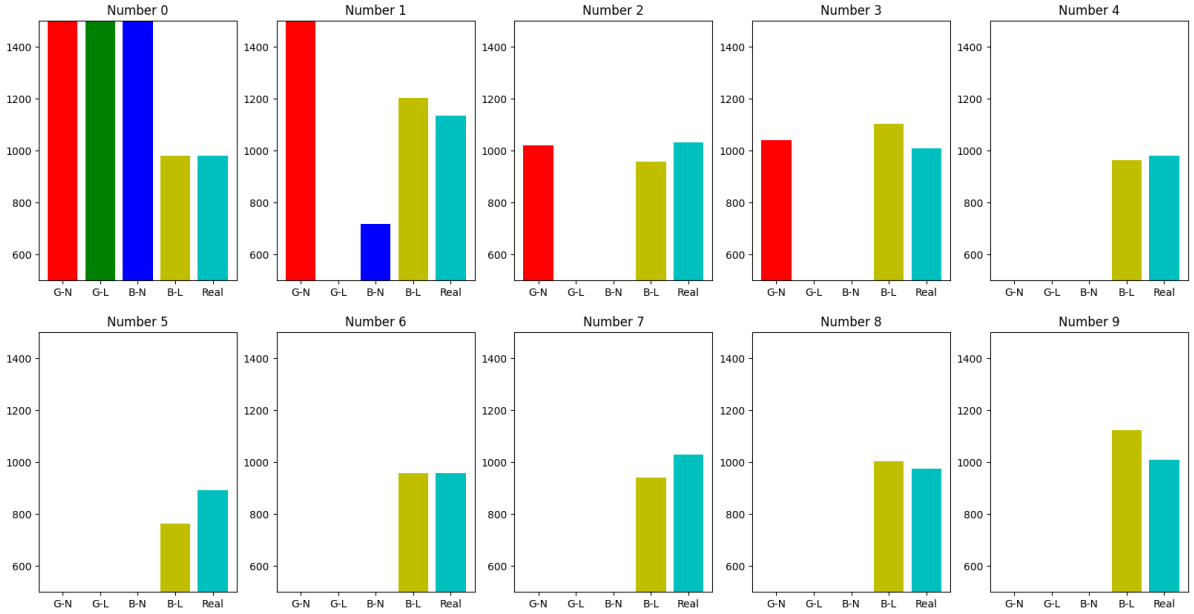
1. G-N 即 grey-nolog：使用原始灰度图像和不取对数方法计算概率；
2. G-L 即 grey-log：使用原始灰度图像和取对数方法计算概率；
3. B-N 即 bin-nolog：使用二值化图像和不取对数方法计算概率；
4. B-L 即 bin-log：使用二值化图像和取对数方法计算概率。

最终的预测精度如下：

G-N	G-L	B-N	B-L
0.093	0.098	0.170	0.841

这的确符合预期的猜想：使用灰度图像会导致类别取值过多而使得连乘结果过小以至于下溢；使用不取对数的计算方法同样也会使得连乘结果过小以至于下溢。使用二值化处理以及取对数处理均可以使预测结果改善，且两者均使用可以使得模型预测结果得到显著提升。

为了更好地展示预测结果，使用柱状图进行可视化。分别展示 10 个数字的预测情况。每一个数字在上述 4 种模式预测数量的基础之上增加 1 列标准结果的数量。可以发现：标准测试集的 10000 条数据的标签几乎均匀分布在 10 个数字上，前 3 种模式的预测结果大多集中在 $0, 1, 2, 3$ 数字上，只有采用二值化和取对数模式的预测结果才合理的分布在 10 个数字上。



四、参考资料

[1] PyTorch documentation

[2] 周志华. 机器学习[M]. 北京: 清华大学出版社, 2016.

[3] 贝叶斯定理

五、总结与反思

- ☐ **增加超参。**拉普拉斯平滑处理，与取对数处理进行比较。但由于特征维度过大，因此这种平滑处理并没有实现；
- ☐ **特征选择。**由于 28×28 的图像展开后一共有 784 维的特征，也许特征筛选后的效果会更好，但是由于对图像学不了解因此没有实现。