Report of Assignment 1

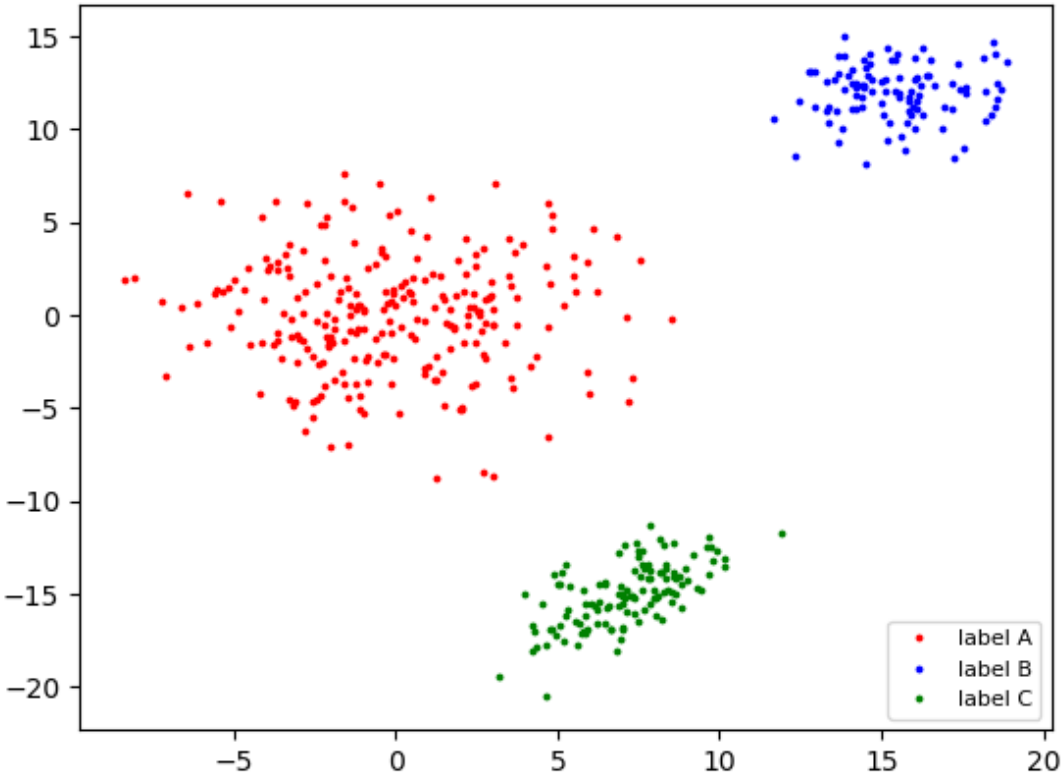
PRML-Spring20-FDU

Part 1：设计高斯分布并取样获得数据集

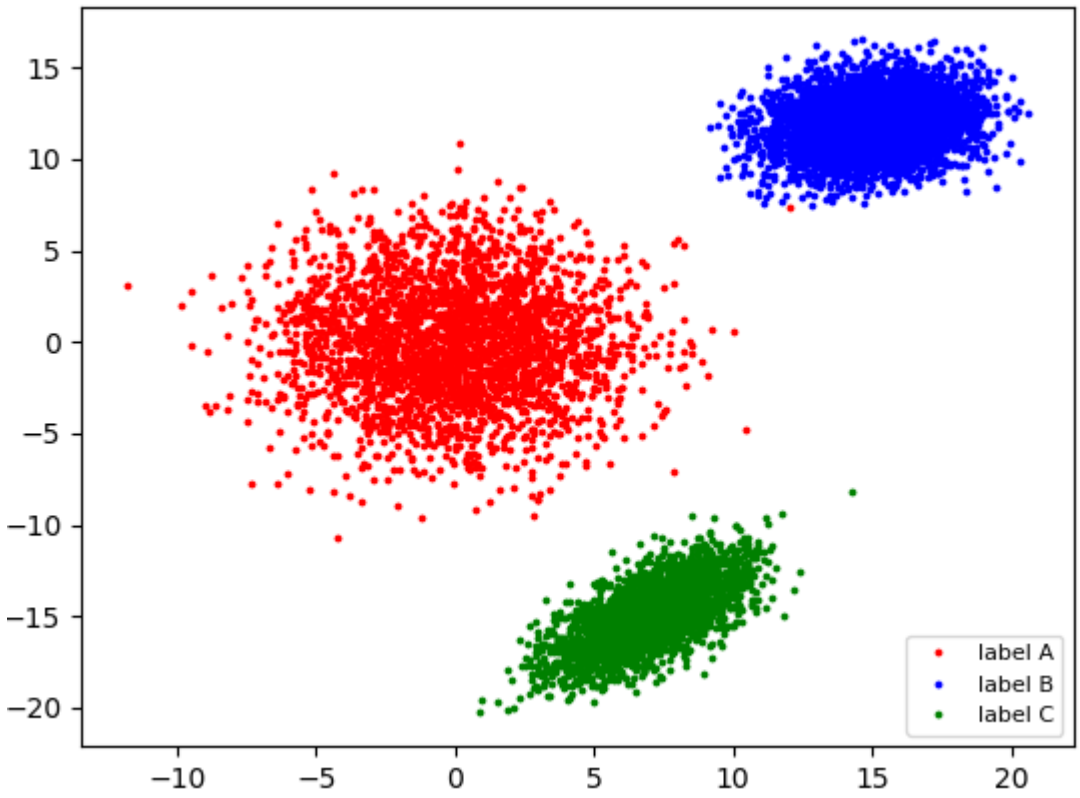
描述：

使用三个二维高斯分布。其中，高斯分布 A 的默认参数为均值μ=[0,0]，协方差矩阵Σ = [[10, 0],[0, 10]]，高斯分布 B 的默认参数为μ=[15,12]，Σ=[[3, 0.5], [0.5, 2]]，高斯分布 C 的默认参数为μ=[7, -15]，Σ=[[3, 2], [2, 3]]。默认情况下从三个数据集中采样的次数分别为 0 到 10000 内的一个随机数。

文件夹内提供了不大于 20Kb 的数据集 **dataset.data**，从 A、B、C 分别采样 250 次、100 次、120 次。



链接中附上一个 410Kb 的数据集 **dataset_big.data**，从 A、B、C 中分别采样 2814 次、4861 次、2614 次。



Command Lines:

e.g. python source.py --mean_A 0 0 --Cov_A 10 0 0 10 --size_A 1000

创建二维正态分布 A、B、C，其中正态分布 A 的均值为[0, 0]，协方差矩阵为[[10,0],[0,10]]，从正态分布 A 中取样 1000 次。正态分布 B、C 的参数为默认值，取样次数为 0 到 10000 的随机数。

e.g. python source.py **--load_data**

从文件 dataset.data 里读取数据集。

相关参数解释：

--mean_A --mean_B --mean_C：高斯分布 A、B、C 的均值μ，--mean_A x y 即设置高斯分布 A 的均值为[x, y]

--cov_A --cov_B --cov_C：高斯分布 A、B、C 的协方差矩阵Σ，--cov a b c d 即设置高斯分布 A 的协方差矩阵为[[a, b], [c, d]] 。其中 a、b、c、d 应满足协方差矩阵的性质，如 b=c。

--size_A --size_B --size_C：从高斯分布 A、B、C 中采样的次数。默认值为 0 到 10000 内的随机整数。

--load_data：如果使用该参数，即从数据集文件里读取数据集，否则将生成数据集并存入文件

--dataset：使用 load_data 参数时，读取 dataset 的文件名，或不适用 load_data 参数时，生成数据存储的文件名。

Part 2：设计并比较生成式模型与判别式模型的区别。

I. 判别式模型：前馈神经网络

描述：

使用**前馈神经网络**作为判别式模型。默认为 4 层，**各层维度为[2, 20, 7, 5, 3]**，其中 2 是输入层维度（二维正态分布），3 是第 4 层的输出维度（softmax 的结果）。

设 A0 = 输入 X，则对任意满足 0<l<=层数的整数 l，有：

$$\mathbf{z}^{(l)} = W^{(l)} \cdot \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)},$$
$$\mathbf{a}^{(l)} = f_l(\mathbf{z}^{(l)}).$$

其中，Wl、bl 为第 l 层的权重和偏置，fl 为第 l 层的激活函数，**最后一层的 fl 取 softmax 函数，其余层的 fl 取 relu。**

使用**交叉熵**作为损失函数 L，并使用 **L2 正则项**，将正则化项和损失函数 L 相加得到总的 Loss R。

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = -\mathbf{y}^{\mathrm{T}} \log \hat{\mathbf{y}},$$
$$\mathcal{R}(W, \mathbf{b}) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(\mathbf{y}^{(n)}, \hat{\mathbf{y}}^{(n)}) + \frac{1}{2} \lambda \|W\|_F^2,$$

其中，yn 为第 n 个样本的 label，而 y-hat n 为第 n 个样本的预测。

使用 **MBGD 和反向传播**对模型的参数进行优化。

默认的学习率为 0.001，默认的 batch_size 为 12，默认的正则化项系数 lambda 为 1e-8。

实验时，split_rate 为 0.95，即取 dataset 中的前 95%作为训练集，后 5%作为测试集。

Command Lines：

e.g. python source.py --epoch 1000 --layers_dims 2 20 7 5 3 --l_r 0.001 --batch_size 12

相关参数解释：

--epoch：训练的 epoch 数

--batch_size：每个批次的大小

--l_r：训练时的学习率

--layers_dims：每层的维度。其中，输入层必须为 2，输出层必须 3。如果要建立 i 层的网络，则这里应该有 i+1 个数，以空格隔开。

--use_generative：使用生成式模型，**默认为 false。**

实验结果：

Epoch	1	50	100	200	300
Train	0.527	0.742	0.951	0.991	0.993
Evaluation	0.625	0.583	0.833	0.958	**1.0**

采用 dataset.data（470 个点）作为数据集，训练若干个 epoch 后的准确率

Epoch	1	10	20	50	100
Train	0.902	0.996	0.998	0.999	0.993
Evaluation	0.878	0.975	0.977	**0.979**	0.979

采用 dataset_big.data（10289 个点）作为数据集，训练若干个 epoch 后的准确率

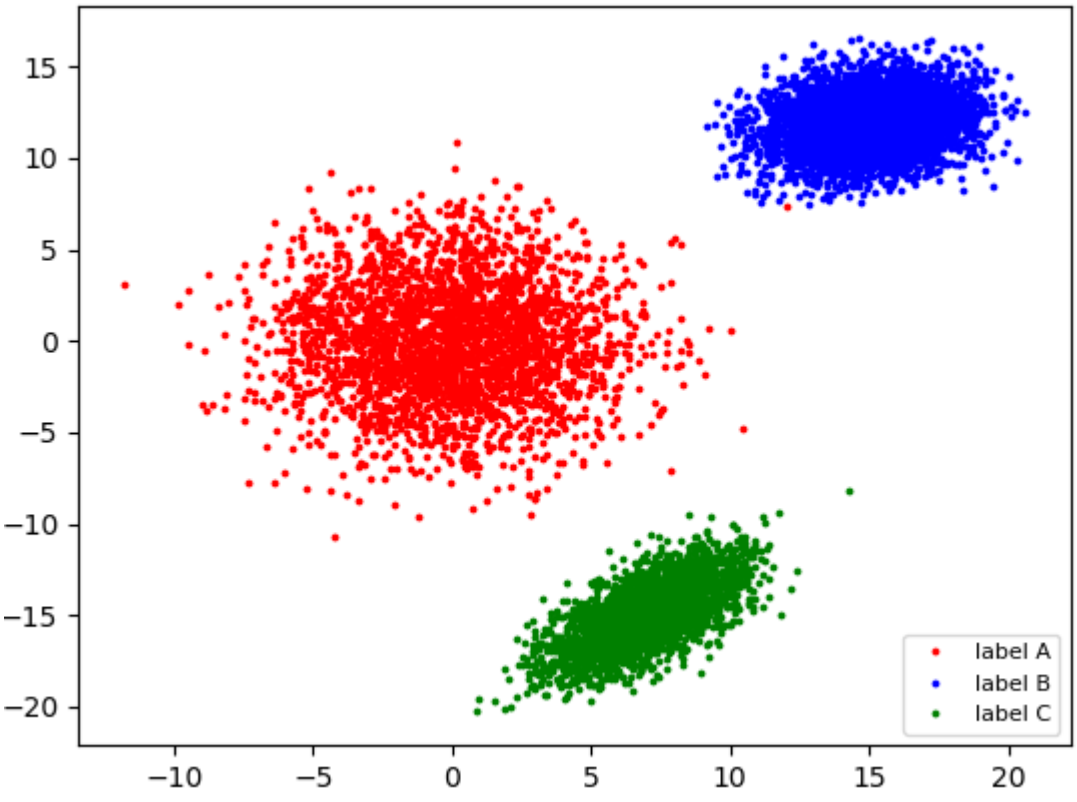
Lambda	0	1e-8	1	5	10
Evaluation	0.979	0.979	0.979	0.971	0.468
相应 epoch	34	35	26	352	2

采用 dataset_big.data 作为数据集，不同的 lambda_regularization 的最终准确率

分析：

当使用 dataset.data 作为数据集时，由于**不同类别的边界很清晰**，因而能在测试集上达到 1.0 的准确率。

而当使用 dataset_big.data 作为数据分布时，可以看到，**有一个 A 类点（红色）非常靠近 B 类点（蓝色）聚簇的地方**，这个点就给分类带来了难度，因而在测试集上无法达到 1.0 的准确率。



dataset.big 的数据分布

Lambda_regularization：因为 dataset_big.data 中三个正态分布的间隔较大，数据的边界清晰，因此在这里当 lambda 取 0 时，也不会有过拟合的现象。而当 lambda 取到 5 时，模型的拟合能力已经有所减弱。lambda 取到 10 时，模型在第 0 个 epochepoch 能取到 0.726 的准确率，但第二个 epoch 开始就迅速下降，顶个在了 0.468，体**现出 lambda 项过大时，对模型的拟合能力有较大限制。**

II. 生成式模型：朴素贝叶斯分类器

描述：

使用**最大似然估计**根据训练集的数据进行参数估计，利用公式：

$$\hat{\mu} = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} = \overline{x}$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)} - \mu\right)\left(x^{(i)} - \mu\right)^{\mathrm{T}}$$

其中 m 使 len(train_dataset)，x(i)是样本 i，μ这里即样本的平均值，与μ-hat 相等。

也就是说，对于训练集中，标签分别为 A、B、C 的点，分别计算他们的分布的**均值和协方差矩阵**，从而得到**估计的分布**。同时，也需要计算类 **A、B、C 的先验概率**，即对应类点的个数除以数据集总的点的个数。

得到参数分布之后，根据贝叶斯公式，**p(ci | d) 正比于 p(ci) * p(d|ci)**，其中 d 为某个样本，ci 为某个类别。这里 p(ci)在上面已经算好了，而 p(d|ci)服从第 i 个标签的正态分布，正态分布参数已知，故可以算出来。因此，将**先验**概率表示为 shape 为（3，1）的列向量（每一行分别是对应类的概率），将**似然**表示为 shape 为（3，len(test_dataset)）的矩阵（第 ij 个元素表示第 p(dj | ci)），然后将先验概率*似然，shape 为（3，1）的先验会沿列方向广播（复制），最终得到（每一列）正比于后验概率的矩阵，按列方向取最大值的下标即预测的标签。

Command Lines：

e.g. python source.py --load_data --use_generative

相关参数解释：

--use_generative：使用生成式模型，默认为 false，因而这里必须显式输入。

实验结果：

准确率	
dataset.data	1.0
dataset_big.data	1.0

dataset.data 和 dataset_big 上朴素贝叶斯分类器的准确率

III. 生成式模型与判别式模型的比较：

对生成式模型与判别式模型这 2 个名词，一直以来都经常听到，但其中的区别，此前并没有很好领悟。但通过本次实验，我发现以下区别：

最直观地就模型的作用来说，判别式模型**解决也仅解决**了分类的问题，对于一个训练好的模型，无法再要求它去做其他事；而生成式模型则通过数据集，估计出了**数据的分布**，从而不仅可以对数据属于哪个分布进行分类，还可以根据分布**生成数据**。生成式模型具备非常**齐全的信息**。

其训练模型的手段也不同。在**训练**的时候，对于判别式模型，我们的目标是训练一个能分类的函数，因此希望得到 **p(c |x)**（c 是标签）。而对于生成式模型，我们是根据我们有的样本(x, c)，估计出(x,c)的联合分布 **p(x, c)**。在**推断**的时候，判别式模型就直接计算 p(c |x)，生成式模型通过贝叶斯公式转换为 p(c |x)。也就是说，**生成式模型能得到判别式模型得到的 p(c |x)，而判别式模型却不能得到生成式模型的 p(x, c)**。

因为生成式模型要估计出数据的分布，对三个分布便有 **3 套参数**，而判别式模型对所有的样本只构建了 **1 套参数**。

IV. 感想：

以前做神经网络时，都是通过 pytorch、tensorflow 搭建，自备反向传播功能，一直以来对反向传播的计算也就没有太大重视。这次实验中，设计前馈神经网络时，需要自己实现反向传播，恶补了矩阵、向量、标量间的求导，大大熟悉了矩运算的表达，受益颇丰。

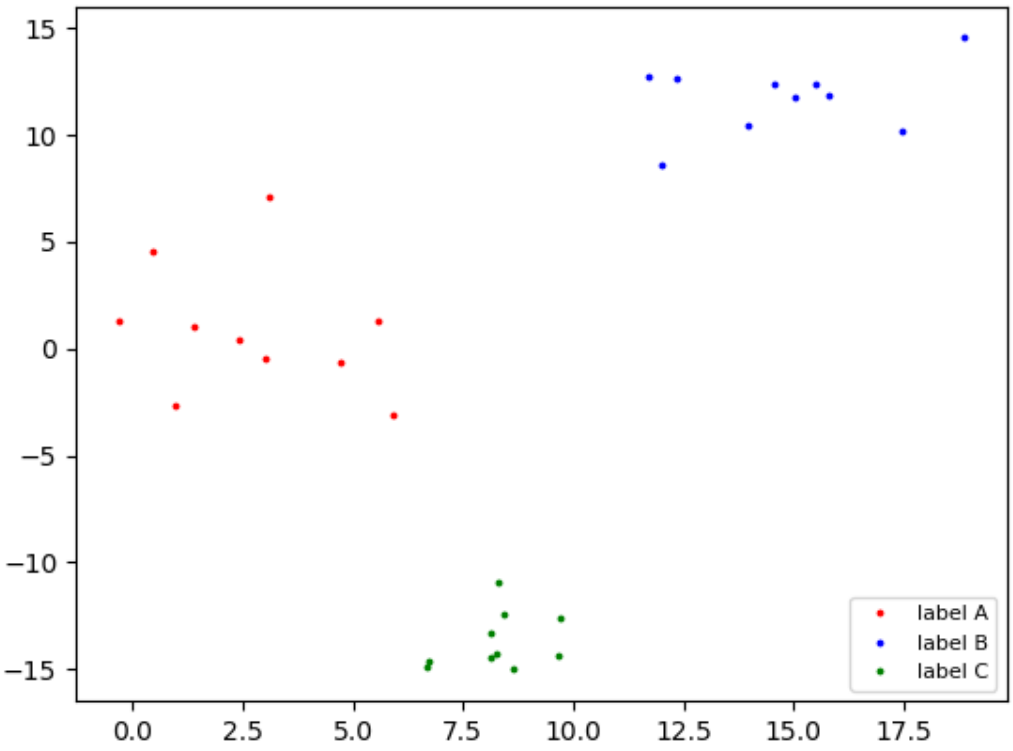
以前对生成式模型、判别式模型，听过许多次，但每次都没很好地理清他们的关键区别，比如 crf 和 hmm，通过这一次实验，我搞懂了生成式模型与判别式模型。

Part 3：重新设置数据集的规模和分布间的重叠进行实验

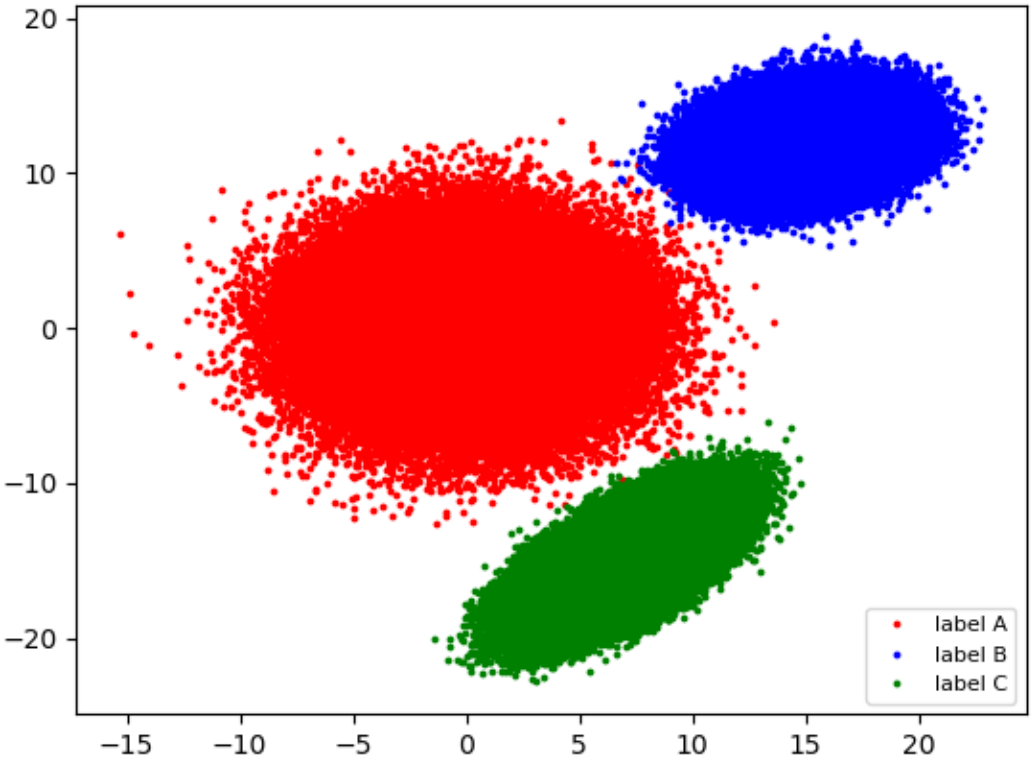
描述：

在原始分布的基础上，将分布 B 的均值改为 **（8，5）**，将分布 C 的均值改为（**5，-6）**，使 A 与 B、A 与 C 之间的数据有**较大重叠**

分别使用**生成式模型、判别式模型**，对**大小**在 30(dataset_small.data，调整 split_rate 为 0.7 防止测试集过小，判别式模型还应将 batch_size 改为 1)、470（dataset.data）、102,89(dataset_big.data)、1,100,000(dataset_extra.data)的数据集，就上述**原始分布和调整后的较大重叠分布**进行实验。（调整重叠后的分布的名字为 data_osmall.data，data_o.data,data_obig.data,data_oextra.data）

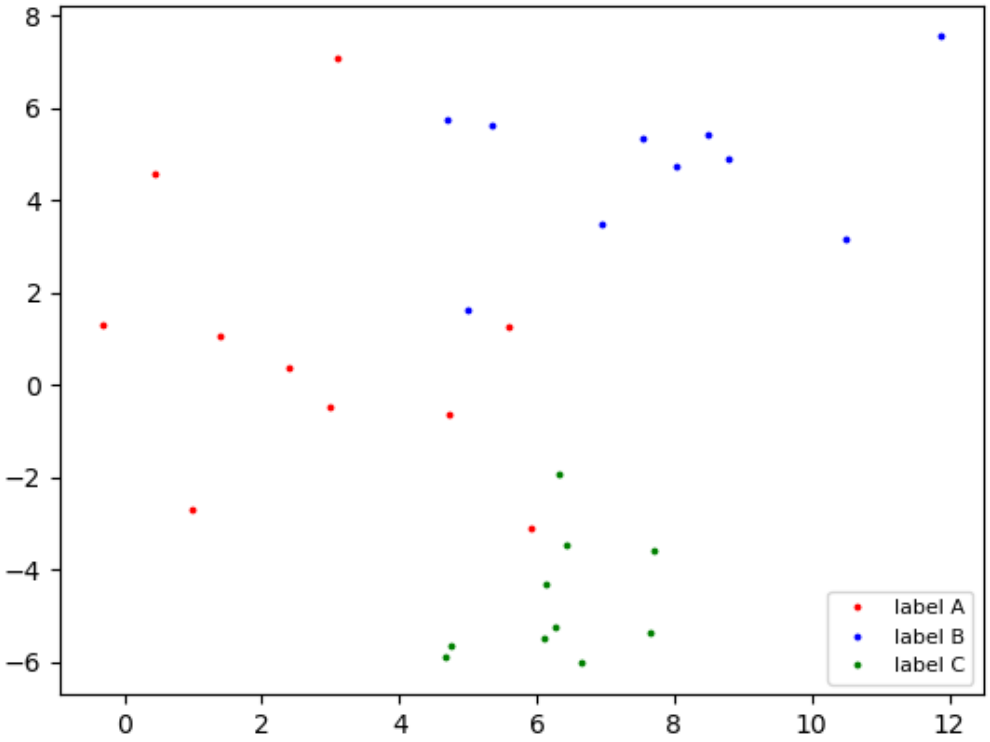


dataset_small.data 的分布（30 个点）

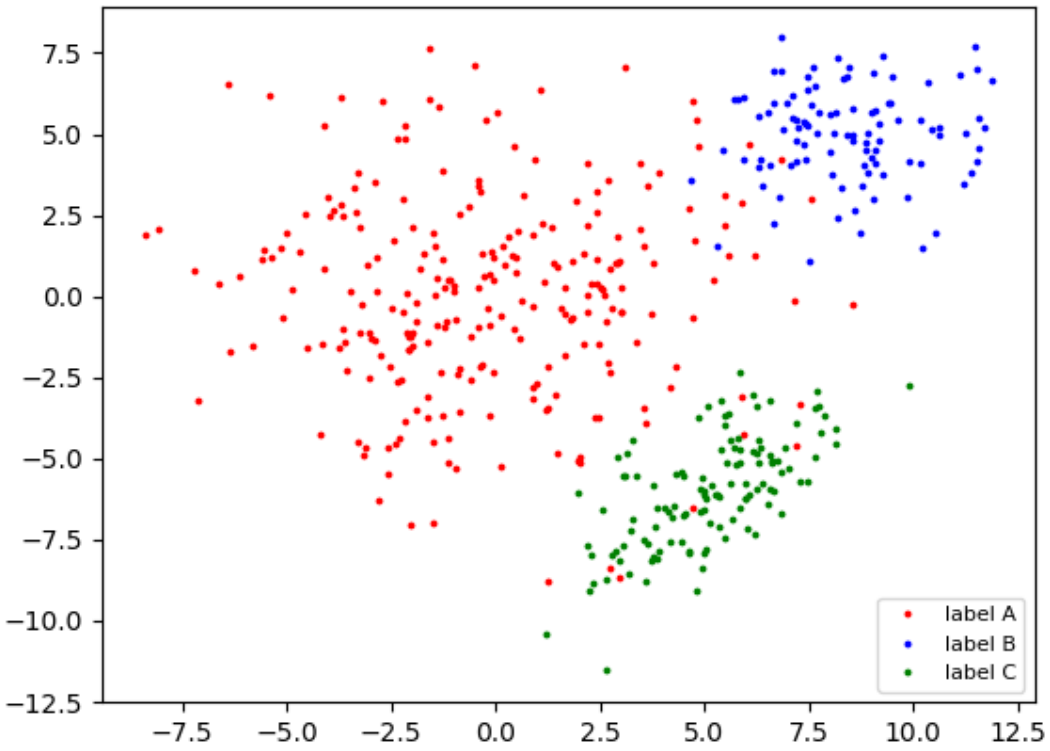


dataset_extra.data 的分布（1,100,000 个点）

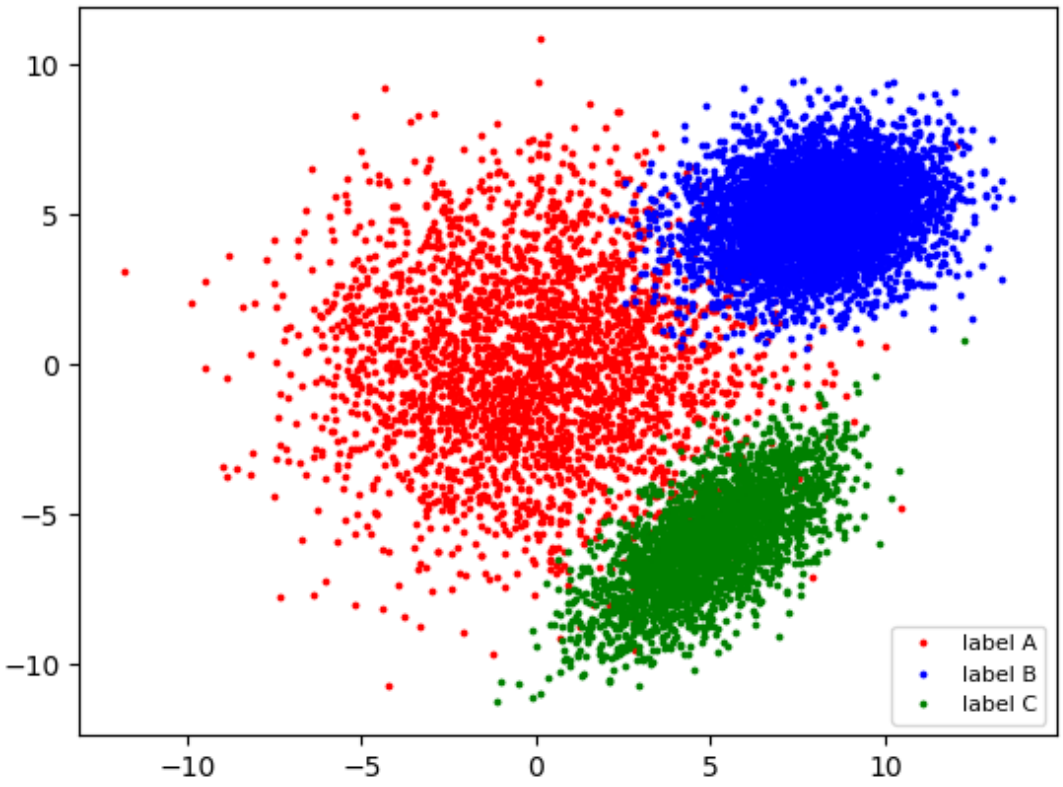
以下是**调整过重叠后的分布**的采样结果:



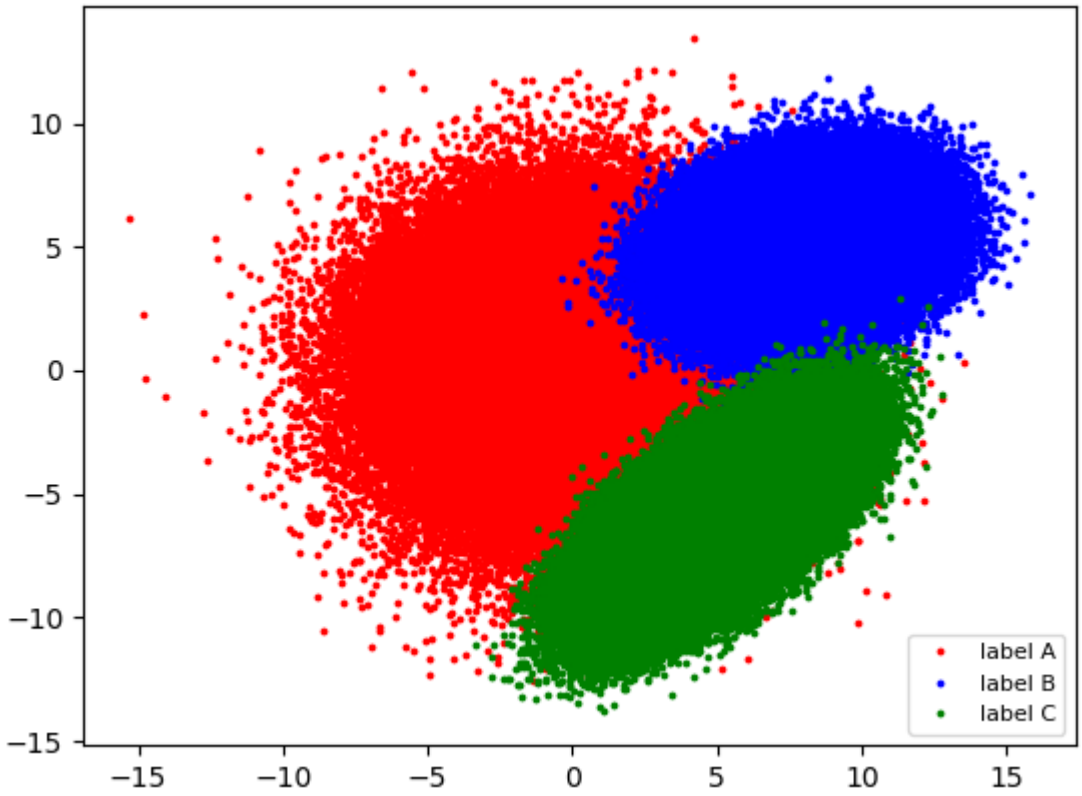
dataset_osmall.data 的分布（30 个点）



dataset_o.data 的分布（470 个点）



dataset_obig.data 的分布（10289 个点）



dataset_oextra.data 的分布（1,100,000 个点）

Command lines：

就原始分布生成有 1,100,000 个点的数据集并使用生成式模型（判别式模型仅需去掉--use_generative）：

python source.py --size_A 100000 --size_B 400000 --size_C 600000 --use_generative --dataset dataset_extra.data

或

python source.py --use_generative --load_data --dataset dataset_extra.data

就原始分布使用仅 30 个点的数据集并使用生成式模型：

python source.py --load_data --dataset dataset_small.data --split_rate 0.7 --use_generative

就原始分布使用仅 30 个点的数据集并使用判别式模型：

python source.py --load_data --dataset dataset_small.data --split_rate 0.7 --batch_size 1 --epoch 10000

如果要修改分布 B 和分布 C 并生成数据，即加上--mean_B 8 5 --mean_C 5 -6，以及对应的 size_A，size_B，size_C。

就调整后的分布使用仅 30 个点的数据并使用生成式模型，设置 split_rate 为 0.6

python source.py --mean_B 8 5 --mean_C 5 -6 --size_A 10 --size_B 10 --size_C 10 --dataset dataset_osmall.data --use_generative --split_rate 0.6

使用判别式模型对 dataset_osmall 进行训练和测试：

python source.py --load_data --dataset dataset_osmall.data --split_rate 0.7 --epoch 10000 **--batch_size 1**

使用判别式模型对 dataset_o 进行训练和测试：

python source.py --load_data --dataset dataset_o.data --epoch 1000

实验结果：

数据集大小	30	500	102,89	1,100,000
生成式模型	1.0	1.0	1.0	0.99996
判别式模型（训练）	1.0	0.993	0.999	0.99996
判别式模型（测试）	1.0	1.0	0.979	0.99989

原始数据分布在不同模型、不同数据集大小下的准确率

数据集大小	30	500	102,89	1,100,000
生成式模型	1.0	0.95833	0.98252	0.98836
判别式模型（训练）	0.95238	0.95964	0.97278	0.98730
判别式模型（测试）	0.90476	1.0	0.96117	098825

调整重叠后的分布在不同模型、不同数据集大小下的准确率

分析：

对于**原始分布**，生成式模型总能达到很高的准确率，除了当数据集有一百万个点时，因为分布 A 和 B、A 和 C 之间的部分点靠得太近，有些模糊，导致准确率未能达到 1.0，但也高达 99.996%。判别式模型也取得了不错的准确率。起初当数据集仅有 30 的时候其在训练集、测试集上分别仅有 0.57 和 0.0 的准确率，后来意识到数据集太小，不适合 MBGD，将 batch_size 改为 1，split_rate 改为 0.7，就可以了。

对于调整重叠后的分布，对于生成式模型，当数据集仅有 30 的大小时，得到了 1.0 的准确率，存在一定的偶然性，而如果将 split_rate 进一步换为 0.6，就仅有 0.833 的准确率了。大小为 500 的数据集上，判别式模型取得 1.0 的准确率，一定程度上也有其偶然性。而当数据集大小 10289 时，朴素贝叶斯的准确率比前馈神经网络多了约 2 个百分点。当数据集大小为百万的级别时，差距反倒缩小了，朴素贝叶斯略优于前馈神经网络。

为什么大小为万的数据集和大小为百万的数据集上会有这样的差异？这应该与 A、B、C 中采样次数的比例有关。大小为 102，89 的数据集中有 2814 个 A 类，4861 个 B 类，2614 个 C 类，而大小为 1,100,100 的数据集中有 100000 个 A 类，400000 个 B 类，600000 个 C 类。这导致了 2 个数据集上的先验有较大差距，因此会对许多点的判定造成影响。大小为百万级的数据集中，**A 的先验较小，故对重叠点，更倾向于分类到 B 和 C 中**。而由于 B 和 C 本身协方差矩阵的范围较小，对于重叠部分，其概率密度更大，而重叠部分在 A 分布中的概率密度较小（**这一点可以从重叠部分是蓝色、绿色而不是红色看出**），因而重叠部分本身大部分便是属于 B 类和 C 类。因此，当 A 类的数据集所占比重较小时，其先验较小，实际上有利于 B 类和 C 类点被正确划分。

值得一提的是，虽然在图上我们看到了相当大一部分的重叠，但实验的结果来看，准确率仍然普遍在 96%、98%左右，其背后的原因在于**正态分布的局部性**。虽然正态分布 A 和 B、A 和 C 乃至 B 和 C 在边缘上的**重叠面积似乎很大**，但本身对于正态分布 A、B、C 来说，点就集中于均值附近，而边缘部分点的密度较低，故**重叠部分密度很低**。另一方面，虽说是重叠部分，但肉眼可见重叠部分主要是蓝色覆盖红色，绿色覆盖红色，即**重叠部分的点也以 B 类、C 类为主**，即使我们盲猜重叠部分属于 B 类、C 类，也能取得不错的准确率。因而结果

来看，准确率比想象的高。

数据集链接：

https://pan.baidu.com/s/10xF7X3crICWqKCHOf0dDoA
提取码：baiu