机器学习导论

第二次作业

191220008 陈南曈

1 Decision tree

(1)

由题可知：

$$|\mathcal{Y}| = 2,\ p_1 = \frac{3}{5},\ p_2 = \frac{2}{5}$$

可以计算出根结点的信息熵为：

$$\mathrm{Ent}(D) = -\sum_{k=1}^{2} p_k \log_2 p_k = -(\frac{3}{5}\log_2\frac{3}{5} + \frac{2}{5}\log_2\frac{2}{5}) = 0.97095059$$

若用属性 A 进行划分：

$$\begin{aligned}
\mathrm{Ent}(D^1) &= -(\frac{2}{3}\log_2\frac{2}{3} + \frac{1}{3}\log_2\frac{1}{3}) = 0.91829583 \\
\mathrm{Ent}(D^2) &= -(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2}) = 1.00000000 \\
\mathrm{Gain}(D, A) &= Ent(D) - \sum_{v=1}^{2}\frac{|D^v|}{|D|}\mathrm{Ent}(D^v) \\
&= 0.97095059 - (\frac{3}{5}\times 0.91829583 + \frac{2}{5}\times 1.00000000) \\
&= 0.01997309199999997
\end{aligned}$$

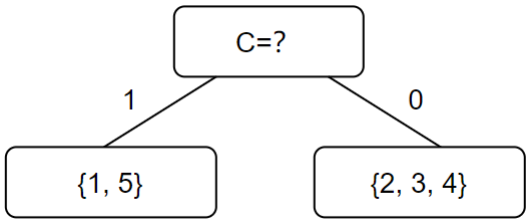
若用属性 B 进行划分：

$$\begin{aligned}
\mathrm{Ent}(D^1) &= -(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2}) = 1.00000000 \\
\mathrm{Ent}(D^2) &= -(\frac{2}{3}\log_2\frac{2}{3} + \frac{1}{3}\log_2\frac{1}{3}) = 0.91829583 \\
\mathrm{Gain}(D, B) &= Ent(D) - \sum_{v=1}^{2}\frac{|D^v|}{|D|}\mathrm{Ent}(D^v) \\
&= 0.97095059 - (\frac{2}{5}\times 1.00000000 + \frac{3}{5}\times 0.91829583) \\
&= 0.01997309199999997
\end{aligned}$$

若用属性 C 进行划分：

$$
\begin{aligned}
\mathrm{Ent}(D^1) &= -(\frac{2}{2}\log_2\frac{2}{2} + \frac{0}{2}\log_2\frac{0}{2}) = 0.00000000 \\
\mathrm{Ent}(D^2) &= -(\frac{1}{3}\log_2\frac{1}{3} + \frac{2}{3}\log_2\frac{2}{3}) = 0.91829583 \\
\mathrm{Gain}(D, C) &= Ent(D) - \sum_{v=1}^{2}\frac{|D^v|}{|D|}\mathrm{Ent}(D^v) \\
&= 0.97095059 - (\frac{2}{5}\times 0.00000000 + \frac{3}{5}\times 0.91829583) \\
&= 0.41997309199999997
\end{aligned}
$$

显然，属性 C 的信息增益最大，因此属性 C 被选为划分属性，划分结果为：



其中，左边的分支结点包含的样本全属于同一类别，无需再划分，故该结点标记为叶结点。

对右边的分支结点继续进行属性划分，该结点的信息熵为：

$$
\mathrm{Ent}(D^2) = -\sum_{k=1}^{2} p_k \log_2 p_k = -(\frac{1}{3}\log_2\frac{1}{3} + \frac{2}{3}\log_2\frac{2}{3}) = 0.91829583
$$

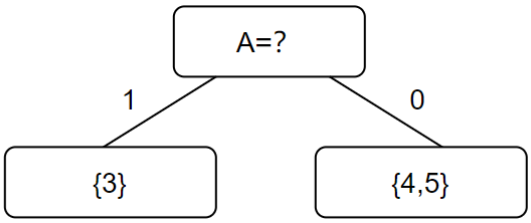
若用属性 A 进行划分：

$$
\begin{aligned}
\mathrm{Ent}(D^3) &= -(\frac{0}{1}\log_2\frac{0}{1} + \frac{1}{1}\log_2\frac{1}{1}) = 0.00000000 \\
\mathrm{Ent}(D^4) &= -(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2}) = 1.00000000 \\
\mathrm{Gain}(D^2, A) &= Ent(D^2) - \sum_{v=3}^{4}\frac{|D^v|}{|D^2|}\mathrm{Ent}(D^v) \\
&= 0.91829583 - (\frac{1}{3}\times 0.00000000 + \frac{2}{3}\times 1.00000000) \\
&= 0.2516291633333334
\end{aligned}
$$

若用属性 B 进行划分：

$$
\begin{aligned}
\mathrm{Ent}(D^3) &= -(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2}) = 1.00000000 \\
\mathrm{Ent}(D^4) &= -(\frac{0}{1}\log_2\frac{0}{1} + \frac{1}{1}\log_2\frac{1}{1}) = 0.00000000 \\
\mathrm{Gain}(D^2, B) &= Ent(D^2) - \sum_{v=3}^{4}\frac{|D^v|}{|D^2|}\mathrm{Ent}(D^v) \\
&= 0.91829583 - (\frac{2}{3}\times 1.00000000 + \frac{1}{3}\times 0.00000000) \\
&= 0.2516291633333334
\end{aligned}
$$

可见，属性 A 和属性 B 均取得了最大的信息增益，按照题目要求，选取字母序靠前的属性 A 作为划分属性，划分结果为：



其中，左边的分支结点包含的样本全属于同一类别，无需再划分，故该结点标记为叶结点。

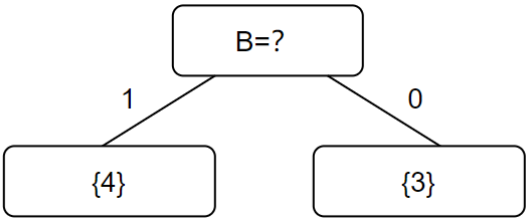
对右边的分支结点继续进行属性划分，该结点的信息熵为：

$$\mathrm{Ent}(D^4) = -\sum_{k=1}^{2} p_k \log_2 p_k = -(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2}) = 1.00000000$$

若用属性 B 进行划分：

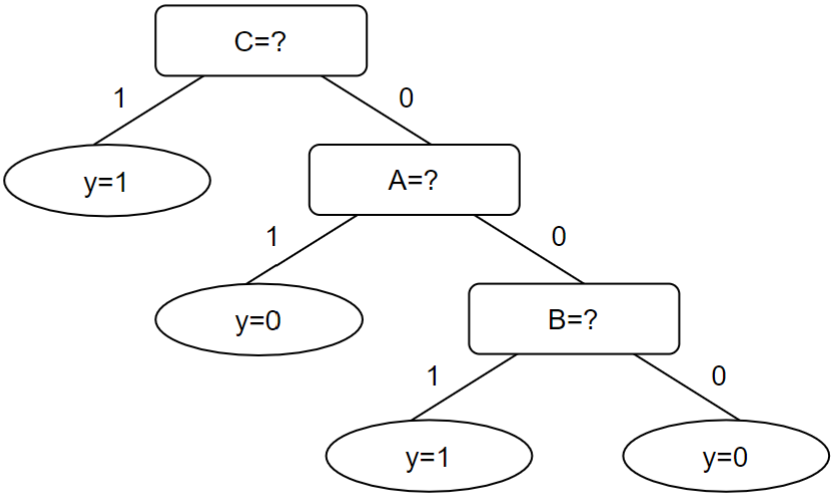
$$\begin{aligned}
\mathrm{Ent}(D^5) &= -(\frac{1}{1}\log_2\frac{1}{1} + \frac{0}{1}\log_2\frac{0}{1}) = 0.00000000 \\
\mathrm{Ent}(D^6) &= -(\frac{0}{1}\log_2\frac{0}{1} + \frac{1}{1}\log_2\frac{1}{1}) = 0.00000000 \\
\mathrm{Gain}(D^4, B) &= Ent(D^4) - \sum_{v=5}^{6} \frac{|D^v|}{|D^4|}\mathrm{Ent}(D^v) \\
&= 1.00000000 - (\frac{1}{2} \times 0.00000000 + \frac{1}{2} \times 0.00000000) \\
&= 1.00000000
\end{aligned}$$

显然，属性 B 的信息增益最大，因此属性 B 被选为划分属性，划分结果为：



其中，左边和右边的分支结点包含的样本均分别全属于同一类别，无需再划分，故均标记为叶结点。

综合上述的划分结果，最终得到的决策树为：



(2)

将上述决策树用于 Table 3 的测试，得到测试结果为：

A	B	C	y（真实值）	f(x)（预测值）
0	0	0	0	0
0	1	1	1	1
1	1	1	0	1
1	0	0	0	0

根据上表，可以得到决策树在测试集上的错误率和精度分别为：

$$E(f;D')=\frac{1}{4}\sum_{i=1}^{4}\mathbb{I}(f(x_i)\neq y_i)=\frac{1}{4}$$

$$\mathrm{acc}(f;D')=\frac{1}{4}\sum_{i=1}^{4}\mathbb{I}(f(x_i)=y_i)=1-E(f;D')=\frac{3}{4}$$

鉴于该问题属于二分类问题，因此可以得到分类结果的混淆矩阵：

二分类决策树		预测结果	预测结果
		正例	反例
真实结果	正例	1	0
真实结果	反例	1	2

根据混淆矩阵可以得到：

$$TP=1,\ FP=1,\ FN=0,\ TN=2$$

分别计算查准率、查全率和 F1 进行性能度量：

$$P=\frac{TP}{TP+FP}=\frac{1}{1+1}=\frac{1}{2}$$

$$R=\frac{TP}{TP+FN}=\frac{1}{1+0}=1$$

$$F1=\frac{2\times TP}{\text{样例总数}+TP-TN}=\frac{2\times 2}{4+1-2}=\frac{2}{3}$$

2 Neural network

代码文件：pendigits.py

代码运行方法：直接运行即可（若需要更改参数，可直接在以下两处进行替换；数据集须与代码文件位于同一目录下）。

```
# Adadelta, Adagrad, Adam, Adamax, ASGD, RMSprop, Rprop, SGD
optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
```

```
EPOCHS = 100
LEARNING_RATE = 0.01
INPUT = 16
HIDDEN1 = 32
HIDDEN2 = 64
HIDDEN3 = 32
OUTPUT = 10
```

在本题中，基于pendigits数据集搭建了一个由输入层，三层全连接层和输出层构成的简单神经网络。

接下来，我将分别在不同的超参数，学习率和优化器下，展示该模型在训练集和测试集上的表现情况。

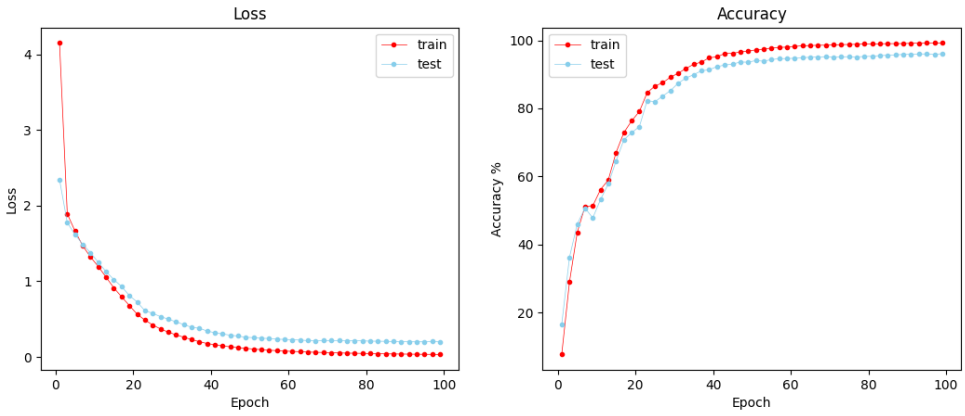
我们固定训练的轮数为 `Epoch = 100`，损失函数为对数似然代价函数 `NLLLoss`，输入层神经元数目 `INPUT= 16`，输出层神经元数目 `OUTPUT = 10`。

1、超参数

这里主要考虑三个全连接层的神经元数目。

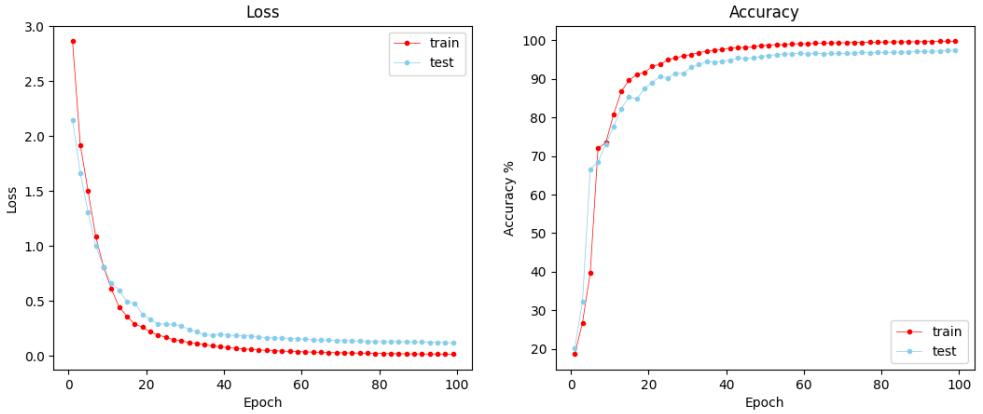
固定学习率为 `LEARNINGRATE = 0.01`，优化器为 `Adam`。

① `HIDDEN_1 = 24, HIDDEN_2 = 32, HIDDEN_3 = 16`：



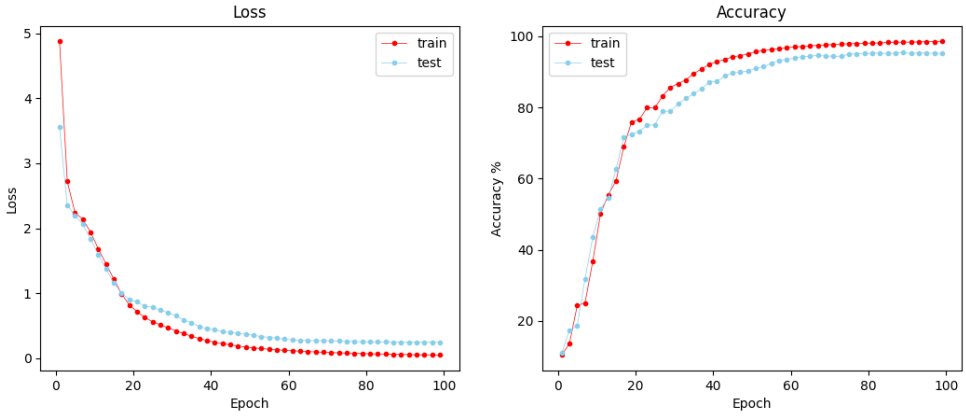

```
Training Set: Epoch [100/100], Loss: 0.0318, Accuracy: 7436/7494 (99.23%)
Testing Set: Epoch [100/100], Loss: 0.2016, Accuracy: 3365/3498 (96.20%)
```

② `HIDDEN_1 = 32, HIDDEN_2 = 64, HIDDEN_3 = 32`：




```
Training Set: Epoch [100/100], Loss: 0.0135, Accuracy: 7474/7494 (99.73%)
Testing Set: Epoch [100/100], Loss: 0.1190, Accuracy: 3409/3498 (97.46%)
```

③ `HIDDEN_1 = 20, HIDDEN_2 = 40, HIDDEN_3 = 20`：




```
Training Set: Epoch [100/100], Loss: 0.0505, Accuracy: 7388/7494 (98.59%)
Testing Set: Epoch [100/100], Loss: 0.2480, Accuracy: 3331/3498 (95.23%)
```

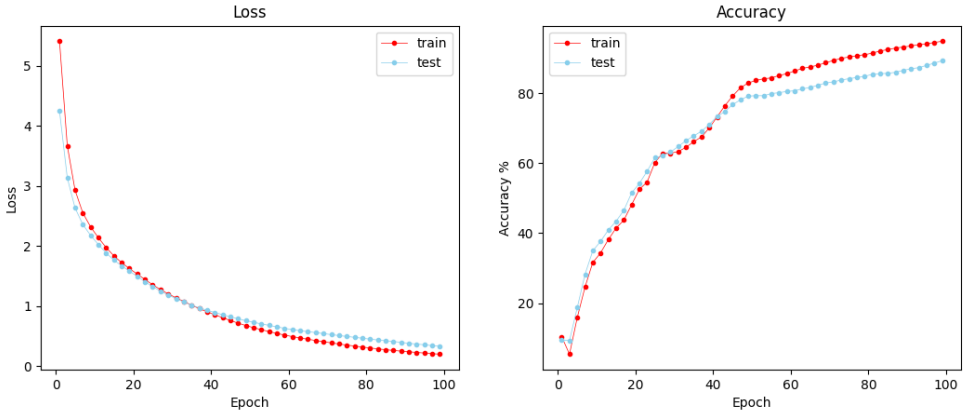
通过对比，发现第二组超参数 `HIDDEN_1 = 32, HIDDEN_2 = 64, HIDDEN_3 = 32` 的效果最好（实际上继续增加全连接层的神经元数目仍能保持不错的效果，但增幅不明显，故不再展示）。

2、学习率

固定超参数为 `HIDDEN_1 = 32, HIDDEN_2 = 64, HIDDEN_3 = 32`，优化器为 `Adam`。

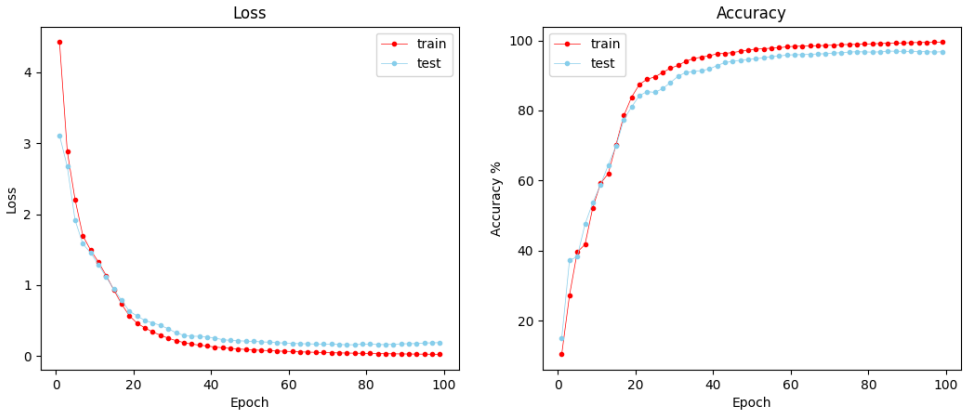
学习率分别取 `LEARNING_RATE = 0.001, 0.005, 0.01, 0.05, 0.1`。

① `LEARNING_RATE = 0.001`：



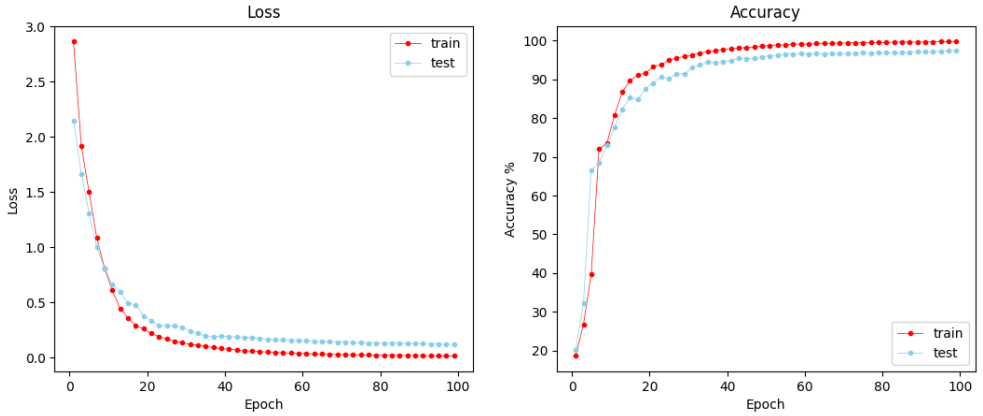

```
Training Set: Epoch [100/100], Loss: 0.1929, Accuracy: 7124/7494 (95.06%)
Testing Set: Epoch [100/100], Loss: 0.3241, Accuracy: 3139/3498 (89.74%)
```

② `LEARNING_RATE = 0.005`:



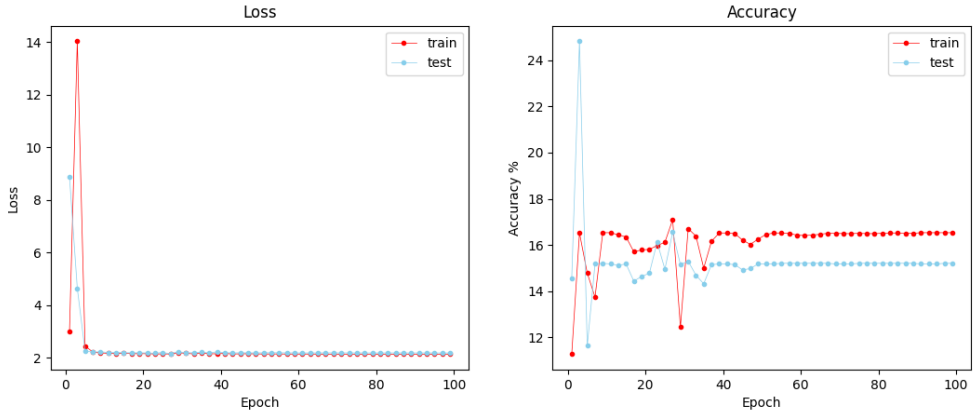

```
Training Set: Epoch [100/100], Loss: 0.0253, Accuracy: 7456/7494 (99.49%)
Testing Set: Epoch [100/100], Loss: 0.1917, Accuracy: 3384/3498 (96.74%)
```

③ `LEARNING_RATE = 0.01`:



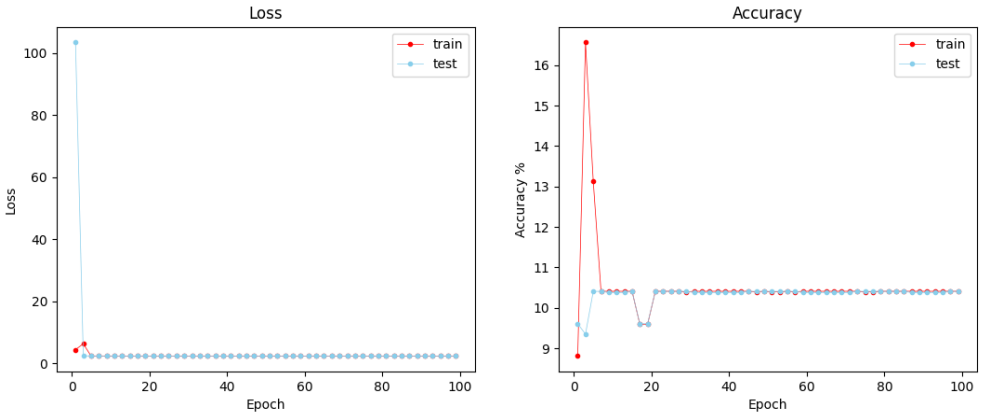

```
Training Set: Epoch [100/100], Loss: 0.0135, Accuracy: 7474/7494 (99.73%)
Testing Set: Epoch [100/100], Loss: 0.1190, Accuracy: 3409/3498 (97.46%)
```

④ `LEARNING_RATE = 0.05`:




```
Training Set: Epoch [100/100], Loss: 2.1354, Accuracy: 1239/7494 (16.53%)
Testing Set: Epoch [100/100], Loss: 2.1824, Accuracy: 532/3498 (15.21%)
```

⑤ `LEARNING_RATE = 0.1`：




```
Training Set: Epoch [100/100], Loss: 2.3018, Accuracy: 780/7494 (10.41%)
Testing Set: Epoch [100/100], Loss: 2.3018, Accuracy: 364/3498 (10.41%)
```

经过五组不同学习率的对比，可知第三组学习率 `LEARNING_RATE = 0.01` 的效果最好。其中前三组随着学习率的增大，损失趋近于 0 和查准率的趋近 100% 的速率越快；但在后两组中，由于学习率过大，步长过大，导致无法找到全局最优解。

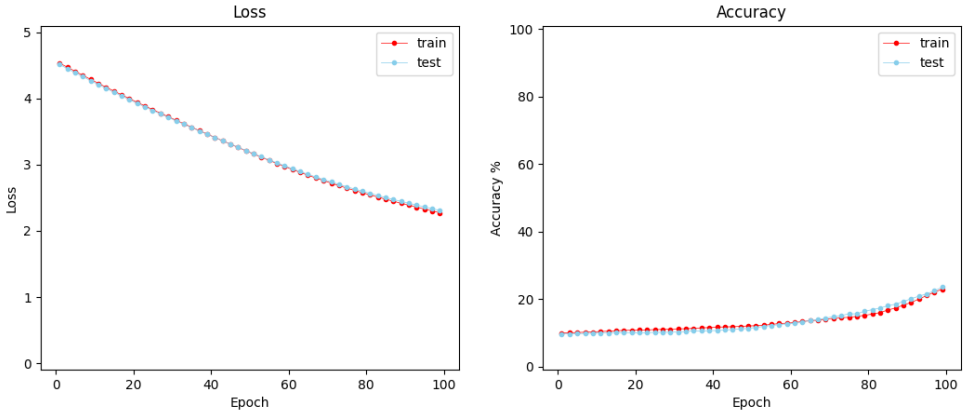
3、优化器

固定超参数为 `HIDDEN_1 = 32, HIDDEN_2 = 64, HIDDEN_3 = 32`，学习率为 `LEARNING_RATE = 0.01`。

优化器分别取：`Adadelta, Adagrad, Adam, Adamax, ASGD, RMSprop, Rprop, SGD`。

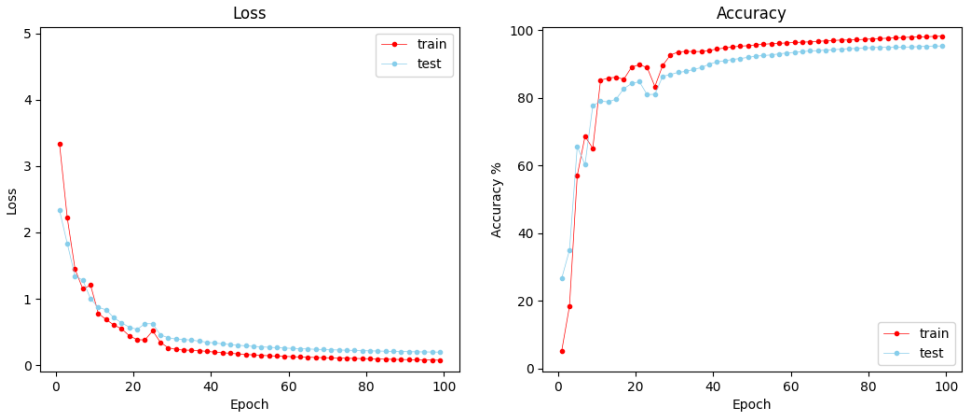
为了方便比较和观察，下列坐标图中固定纵坐标的范围，`Loss∈(0, 5), Accuracy∈(0, 100)`。

① `Adadelta` 优化器：



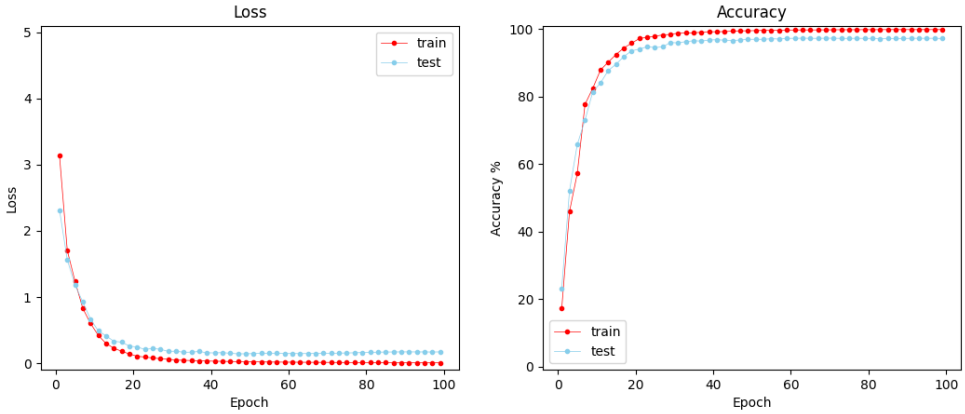

```
Training Set: Epoch [100/100], Loss: 2.2589, Accuracy: 1758/7494 (23.46%)
Testing Set: Epoch [100/100], Loss: 2.2944, Accuracy: 835/3498 (23.87%)
```

② `Adagrad` 优化器：



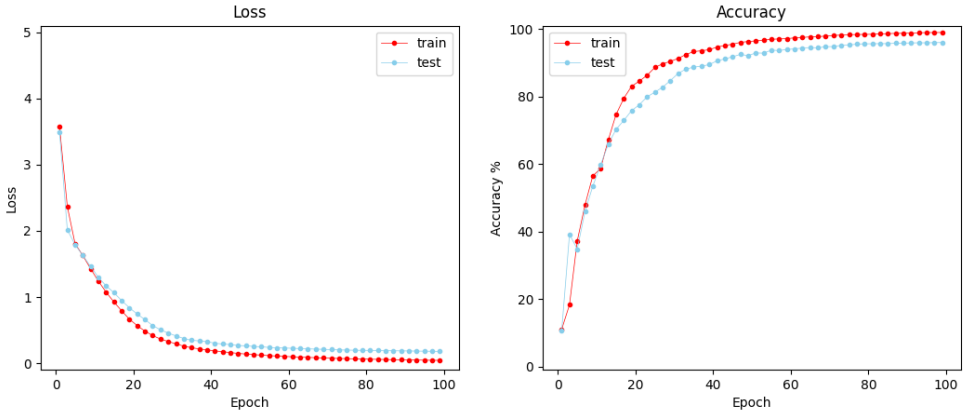

```
Training Set: Epoch [100/100], Loss: 0.0733, Accuracy: 7377/7494 (98.44%)
Testing Set: Epoch [100/100], Loss: 0.1940, Accuracy: 3322/3498 (94.97%)
```

③ `Adam` 优化器：



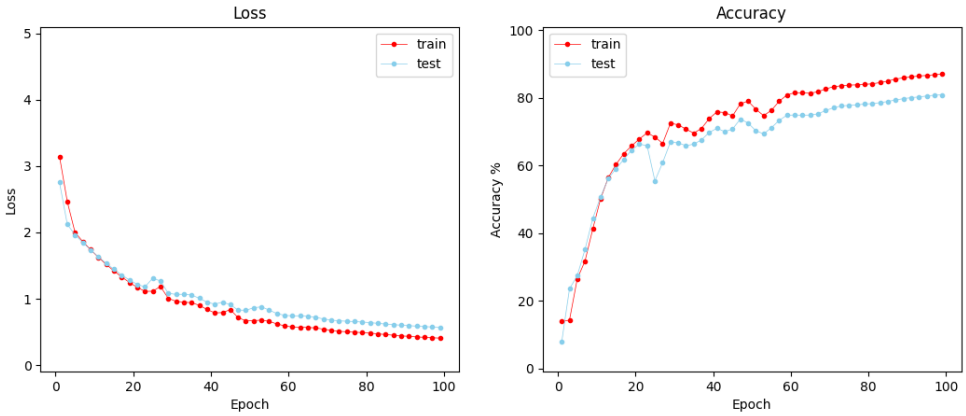

```
Training Set: Epoch [100/100], Loss: 0.0064, Accuracy: 7485/7494 (99.88%)
Testing Set: Epoch [100/100], Loss: 0.1711, Accuracy: 3402/3498 (97.26%)
```

④ `Adamax` 优化器：



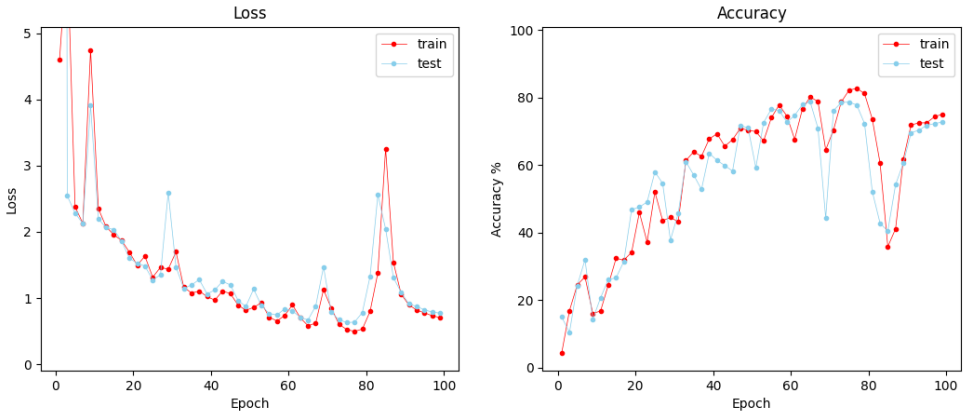

```
Training Set: Epoch [100/100], Loss: 0.0424, Accuracy: 7419/7494 (99.00%)
Testing Set: Epoch [100/100], Loss: 0.1775, Accuracy: 3358/3498 (96.00%)
```

⑤ `ASGD` 优化器：



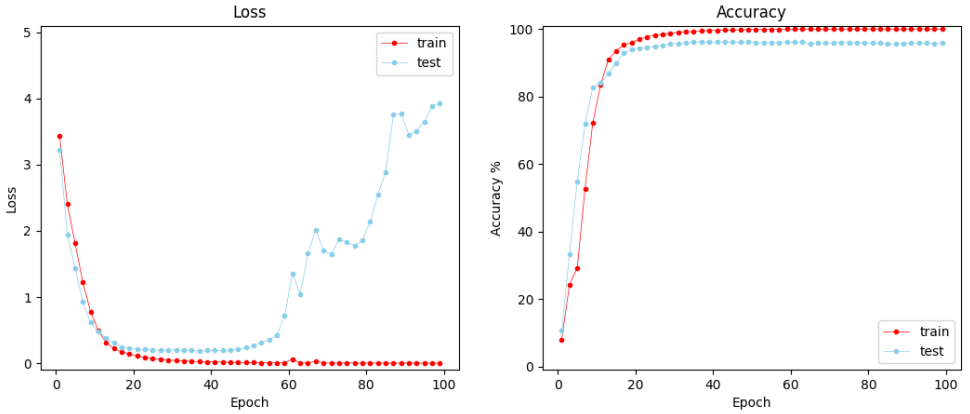

```
Training Set: Epoch [100/100], Loss: 0.4081, Accuracy: 6524/7494 (87.06%)
Testing Set: Epoch [100/100], Loss: 0.5482, Accuracy: 2882/3498 (82.39%)
```

⑥ `RMSprop` 优化器：



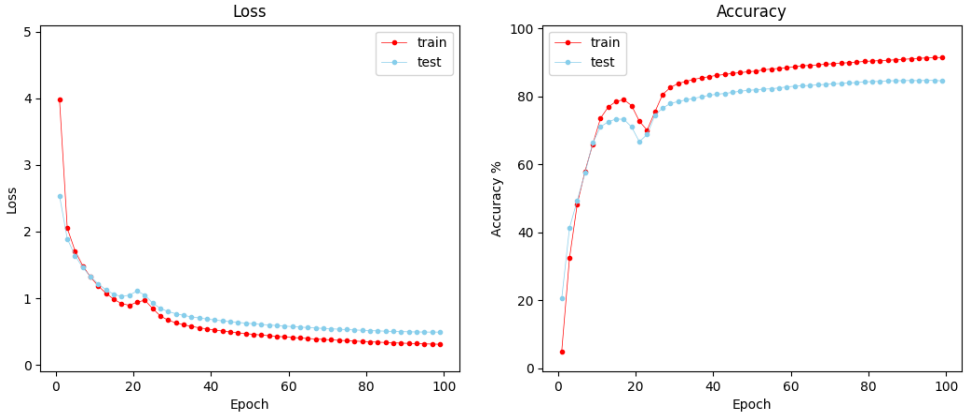

```
Training Set: Epoch [100/100], Loss: 0.6953, Accuracy: 5628/7494 (75.10%)
Testing Set: Epoch [100/100], Loss: 0.7742, Accuracy: 2502/3498 (71.53%)
```

⑦ `Rprop` 优化器：




```
Training Set: Epoch [100/100], Loss: 0.0002, Accuracy: 7494/7494 (100.00%)
Testing Set: Epoch [100/100], Loss: 3.9868, Accuracy: 3353/3498 (95.85%)
```

⑧ `SGD` 优化器：




```
Training Set: Epoch [100/100], Loss: 0.3081, Accuracy: 6848/7494 (91.38%)
Testing Set: Epoch [100/100], Loss: 0.4543, Accuracy: 3052/3498 (87.25%)
```

经过上述对比，综合考虑训练集和测试集的效果，`Adam` 优化器的表现最好。在其余的优化器中，`Adadelta` 优化器的表现很不理想；`RMSprop` 优化器的抖动幅度很大；`Rprop` 优化器在训练集上达到了惊人的 100%，但当训练轮次增大到一定程度后在测试集上性能突转下降，可能是过拟合所导致；`ASGD` 优化器和 `SGD` 优化器的表现平平；`Adagrad` 优化器和 `Adamax` 优化器的表现较为理想。

综上所述，当

超参数为 `HIDDEN_1 = 32, HIDDEN_2 = 64, HIDDEN_3 = 32`，

学习率为 `LEARNING_RATE = 0.01`，

优化器为 `Adam`，

此时，模型的性能达到最佳。

3 Learn from inbalanced and noisy data

（1）

代码文件：dicision_tree.py, neural_network.py, svm.py, k_neighbors.py, logistic_regression.py, naive_bayes.py

代码运行方法：直接运行即可（若需要切换训练集，在下列两行数据导入的代码中将不需要的训练集注释掉即可；代码文件需位于 pendigits-corrupted-main 目录下）。

```
#original training data
#input_train, target_train = get_data(train=True)
#corrupted training data
input_train, target_train = get_data(train=True, corrupt=True)
```

1、基于 pendigits-corrupted.tra 训练的结果

① 决策树（Dicision Tree）

```
TimeCost: 0.036902s
Accuracy: 54.95%
Classification Report:
              precision    recall  f1-score   support

           0       0.54      0.81      0.65       363
           1       0.46      0.77      0.57       364
           2       0.59      0.74      0.66       364
           3       0.60      0.54      0.57       336
           4       0.63      0.89      0.74       364
           5       0.45      0.66      0.53       335
           6       0.57      0.26      0.35       336
           7       0.64      0.24      0.35       364
           8       0.54      0.19      0.28       336
           9       0.65      0.34      0.45       336

    accuracy                           0.55      3498
   macro avg       0.57      0.54      0.52      3498
weighted avg       0.57      0.55      0.52      3498
```

② 神经网络（Neural Network）

```
TimeCost: 6.022891s
Accuracy: 65.47%
Classification Report:
              precision    recall  f1-score   support

           0       0.57      0.96      0.71       363
           1       0.59      0.98      0.73       364
           2       0.77      0.78      0.77       364
           3       0.74      0.78      0.76       336
           4       0.61      0.82      0.70       364
           5       0.57      0.85      0.69       335
           6       0.98      0.25      0.40       336
           7       0.96      0.30      0.46       364
           8       0.75      0.36      0.49       336
           9       0.69      0.41      0.52       336

    accuracy                           0.65      3498
   macro avg       0.72      0.65      0.62      3498
weighted avg       0.72      0.65      0.62      3498
```

③ 支持向量机（SVM）

```
TimeCost: 0.409901s
Accuracy: 62.41%
Classification Report:
              precision    recall  f1-score   support

           0       0.51      0.98      0.67       363
           1       0.48      0.97      0.64       364
           2       0.97      0.99      0.98       364
           3       0.95      0.98      0.96       336
           4       0.55      0.98      0.70       364
           5       0.54      0.93      0.68       335
           6       0.00      0.00      0.00       336
           7       0.00      0.00      0.00       364
           8       0.94      0.22      0.35       336
           9       1.00      0.12      0.22       336

    accuracy                           0.62      3498
   macro avg       0.59      0.62      0.52      3498
weighted avg       0.59      0.62      0.53      3498
```

④ K-近邻（K-Neighbors）

```
TimeCost: 0.000997s
Accuracy: 66.41%
Classification Report:
              precision    recall  f1-score   support

           0       0.56      0.98      0.71       363
           1       0.51      0.96      0.67       364
           2       0.74      0.86      0.80       364
           3       0.77      0.74      0.75       336
           4       0.67      0.95      0.78       364
           5       0.62      0.93      0.75       335
           6       0.96      0.20      0.33       336
           7       1.00      0.15      0.26       364
           8       0.95      0.44      0.60       336
           9       0.91      0.40      0.56       336

    accuracy                           0.66      3498
   macro avg       0.77      0.66      0.62      3498
weighted avg       0.77      0.66      0.62      3498
```

⑤ 逻辑回归（Logistic Regression）

```
TimeCost: 0.311164s
Accuracy: 59.55%
Classification Report:
              precision    recall  f1-score   support

           0       0.44      0.92      0.60       363
           1       0.45      0.87      0.59       364
           2       0.85      0.97      0.90       364
           3       0.89      0.98      0.93       336
           4       0.59      0.98      0.73       364
           5       0.56      0.77      0.65       335
           6       1.00      0.01      0.02       336
           7       0.33      0.01      0.01       364
           8       0.67      0.21      0.31       336
           9       0.98      0.19      0.32       336

    accuracy                           0.60      3498
   macro avg       0.68      0.59      0.51      3498
weighted avg       0.67      0.60      0.51      3498
```

⑥ 朴素贝叶斯（Naive Bayes）

```
TimeCost: 0.005990s
Accuracy: 74.64%
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.88      0.86       363
           1       0.62      0.60      0.61       364
           2       0.76      0.88      0.81       364
           3       0.69      0.91      0.79       336
           4       0.87      0.99      0.92       364
           5       0.40      0.50      0.45       335
           6       0.98      0.91      0.94       336
           7       0.99      0.70      0.82       364
           8       0.64      0.32      0.43       336
           9       0.74      0.76      0.75       336

    accuracy                           0.75      3498
   macro avg       0.76      0.74      0.74      3498
weighted avg       0.76      0.75      0.74      3498
```

由上述结果可见，基于 pendigits-corrupted.tra 训练时：

时间开销：神经网络 > 支持向量机 > 逻辑回归 > 决策树 > 朴素贝叶斯 > K-近邻

准确率：朴素贝叶斯 > K-近邻 > 神经网络 > 支持向量机 > 逻辑回归 > 决策树

2、基于 pendigits.tra 训练的结果

① 决策树（Dicision Tree）

```
TimeCost: 0.040890s
Accuracy: 87.68%
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.96      0.97       363
           1       0.65      0.89      0.75       364
           2       0.87      0.94      0.90       364
           3       0.88      0.67      0.76       336
           4       0.94      0.90      0.92       364
           5       0.90      0.81      0.85       335
           6       0.97      0.90      0.93       336
           7       0.96      0.80      0.87       364
           8       0.89      0.97      0.93       336
           9       0.83      0.91      0.87       336

    accuracy                           0.88      3498
   macro avg       0.89      0.88      0.88      3498
weighted avg       0.89      0.88      0.88      3498
```

② 神经网络（Neural Network）

```
TimeCost: 5.394673s
Accuracy: 96.74%
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.96      0.98       363
           1       0.90      0.99      0.95       364
           2       0.99      0.98      0.99       364
           3       0.96      0.99      0.97       336
           4       0.96      0.98      0.97       364
           5       0.97      0.98      0.98       335
           6       1.00      0.98      0.99       336
           7       0.99      0.87      0.92       364
           8       0.93      0.98      0.95       336
           9       0.99      0.97      0.98       336

    accuracy                           0.97      3498
   macro avg       0.97      0.97      0.97      3498
weighted avg       0.97      0.97      0.97      3498
```

③ 支持向量机（SVM）

```
TimeCost: 0.205489s
Accuracy: 97.48%
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.96      0.98       363
           1       0.90      0.97      0.93       364
           2       0.97      0.99      0.98       364
           3       0.99      0.99      0.99       336
           4       1.00      0.99      0.99       364
           5       0.98      0.98      0.98       335
           6       1.00      1.00      1.00       336
           7       0.99      0.90      0.95       364
           8       0.96      1.00      0.98       336
           9       0.98      0.98      0.98       336

    accuracy                           0.97      3498
   macro avg       0.98      0.98      0.98      3498
weighted avg       0.98      0.97      0.97      3498
```

④ K-近邻（K-Neighbors）

```
TimeCost: 0.000997s
Accuracy: 97.06%
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.97      0.98       363
           1       0.90      0.95      0.92       364
           2       0.95      0.99      0.97       364
           3       0.96      0.99      0.98       336
           4       0.99      0.98      0.99       364
           5       0.97      0.98      0.97       335
           6       0.99      1.00      1.00       336
           7       0.99      0.92      0.95       364
           8       0.99      0.99      0.99       336
           9       0.97      0.95      0.96       336

    accuracy                           0.97      3498
   macro avg       0.97      0.97      0.97      3498
weighted avg       0.97      0.97      0.97      3498
```

⑤ 逻辑回归（Logistic Regression）

```
TimeCost: 1.509057s
Accuracy: 91.34%
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.90      0.92       363
           1       0.84      0.82      0.83       364
           2       0.96      0.98      0.97       364
           3       0.96      0.98      0.97       336
           4       0.98      0.98      0.98       364
           5       0.78      0.93      0.85       335
           6       0.98      0.96      0.97       336
           7       0.95      0.82      0.88       364
           8       0.84      0.85      0.84       336
           9       0.93      0.92      0.92       336

    accuracy                           0.91      3498
   macro avg       0.92      0.91      0.91      3498
weighted avg       0.92      0.91      0.91      3498
```

⑥ 朴素贝叶斯（Naive Bayes）

```
TimeCost: 0.009972s
Accuracy: 72.01%
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.83      0.88       363
           1       0.63      0.60      0.61       364
           2       0.75      0.89      0.82       364
           3       0.83      0.88      0.86       336
           4       0.00      0.00      0.00       364
           5       0.52      0.44      0.48       335
           6       1.00      0.91      0.95       336
           7       0.99      0.80      0.89       364
           8       0.72      0.93      0.81       336
           9       0.44      0.95      0.61       336

    accuracy                           0.72      3498
   macro avg       0.68      0.72      0.69      3498
weighted avg       0.68      0.72      0.69      3498
```

由上述结果可见，基于 pendigits.tra 训练时：

时间开销：神经网络 > 逻辑回归 > 支持向量机 > 决策树 > 朴素贝叶斯 > K-近邻

准确率：朴素贝叶斯 > K-近邻 > 神经网络 > 支持向量机 > 逻辑回归 > 决策树

(2)

代码文件：dicision_tree.py, neural_network.py, svm.py, k_neighbors.py, logistic_regression.py, naive_bayes.py

代码运行方法：直接运行即可（若需要更改 imb_ratio 或 noise_level，直接在data_utils.py 里修改即可；代码文件需位于 pendigits-corrupted-main 目录下）。

本题需要探究类别不平衡和标签噪声对模型的影响，因此需要采用控制变量法来分别研究单个变量对模型的影响。

故取

① `noise_level = 0.3; imb_ratio = 10, 30, 50, 70, 90`

② `imb_ratio = 10; noise_level = 0.1, 0.3, 0.5, 0.7, 0.9`

1、类别不平衡的影响

固定 `noise_level = 0.3`

① `imb_ratio = 10`

	时间开销（Time Cost）/s	准确率（Accuracy）
决策树（Dicision Tree）	0.036902	54.95%
神经网络（Neural Network）	6.022891	65.47%
支持向量机（SVM）	0.409901	62.41%
K-近邻（K-Neighbors）	0.000997	66.41%
逻辑回归（Logistic Regression）	0.311614	59.55%
朴素贝叶斯（Naive Bayes）	0.005990	74.64%

② `imb_ratio = 30`

	时间开销（Time Cost）/s	准确率（Accuracy）
决策树（Dicision Tree）	0.071809	51.54%
神经网络（Neural Network）	4.441236	59.35%
支持向量机（SVM）	0.104720	59.86%
K-近邻（K-Neighbors）	0.000999	60.63%
逻辑回归（Logistic Regression）	0.540680	57.43%
朴素贝叶斯（Naive Bayes）	0.000997	72.50%

③ `imb_ratio = 50`

	时间开销（Time Cost）/s	准确率（Accuracy）
决策树（Dicision Tree）	0.034907	53.26%
神经网络（Neural Network）	4.053168	60.95%
支持向量机（SVM）	0.116687	60.41%
K-近邻（K-Neighbors）	0.000998	59.03%
逻辑回归（Logistic Regression）	0.196475	56.46%
朴素贝叶斯（Naive Bayes）	0.001995	73.21%

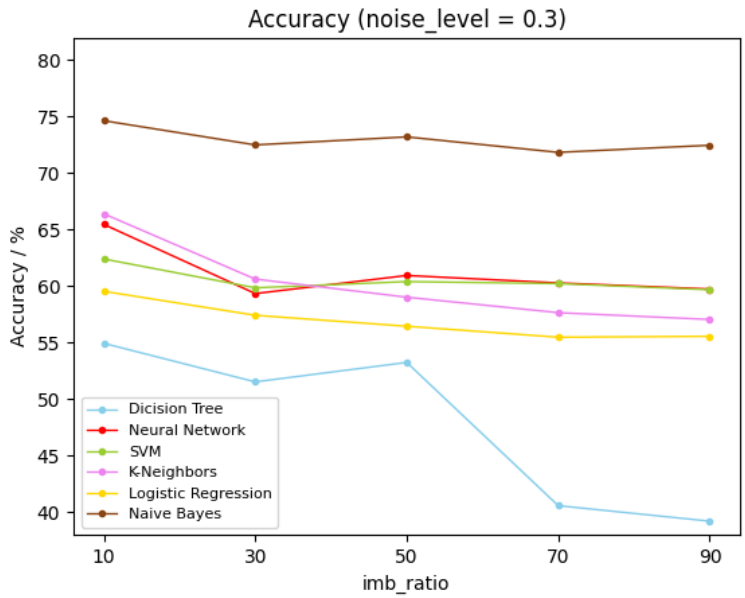
④ `imb_ratio = 70`

	时间开销（Time Cost）/s	准确率（Accuracy）
决策树（Dicision Tree）	0.040890	40.59%
神经网络（Neural Network）	4.901894	60.29%
支持向量机（SVM）	0.071809	60.23%
K-近邻（K-Neighbors）	0.000997	57.66%
逻辑回归（Logistic Regression）	0.523607	55.49%
朴素贝叶斯（Naive Bayes）	0.000997	71.84%

⑤ `imb_ratio = 90`

	时间开销（Time Cost）/s	准确率（Accuracy）
决策树（Dicision Tree）	0.039890	39.22%
神经网络（Neural Network）	2.373658	59.75%
支持向量机（SVM）	0.081776	59.69%
K-近邻（K-Neighbors）	0.000998	57.06%
逻辑回归（Logistic Regression）	0.615363	55.57%
朴素贝叶斯（Naive Bayes）	0.001994	72.47%

将准确率用折线图展示：



可见，随着 `imb_ratio` 的增大，决策树的准确率下降较为明显，其余模型的准确率仅略微下降。

由上表可知，随着 `imb_ratio` 的增大，上述模型的时间开销大致为为先上升后下降的趋势。

2、标签噪声的影响

固定 `imb_ratio = 10`

① `noise_level = 0.1`

	时间开销（Time Cost）/s	准确率（Accuracy）
决策树（Dicision Tree）	0.036902	70.01%
神经网络（Neural Network）	1.563818	78.22%
支持向量机（SVM）	0.113696	94.85%
K-近邻（K-Neighbors）	0.000000	88.42%
逻辑回归（Logistic Regression）	0.229385	80.90%
朴素贝叶斯（Naive Bayes）	0.001994	80.67%

② `noise_level = 0.3`

	时间开销（Time Cost）/s	准确率（Accuracy）
决策树（Dicision Tree）	0.037900	54.95%
神经网络（Neural Network）	2.401579	65.47%
支持向量机（SVM）	0.163562	62.41%
K-近邻（K-Neighbors）	0.000997	66.41%
逻辑回归（Logistic Regression）	0.283243	59.55%
朴素贝叶斯（Naive Bayes）	0.001995	74.64%

③ `noise_level = 0.5`

	时间开销（Time Cost）/s	准确率（Accuracy）
决策树（Dicision Tree）	0.041888	45.37%
神经网络（Neural Network）	1.750319	52.63%
支持向量机（SVM）	0.210438	49.63%
K-近邻（K-Neighbors）	0.001002	46.66%
逻辑回归（Logistic Regression）	0.299199	42.00%
朴素贝叶斯（Naive Bayes）	0.002991	37.19%

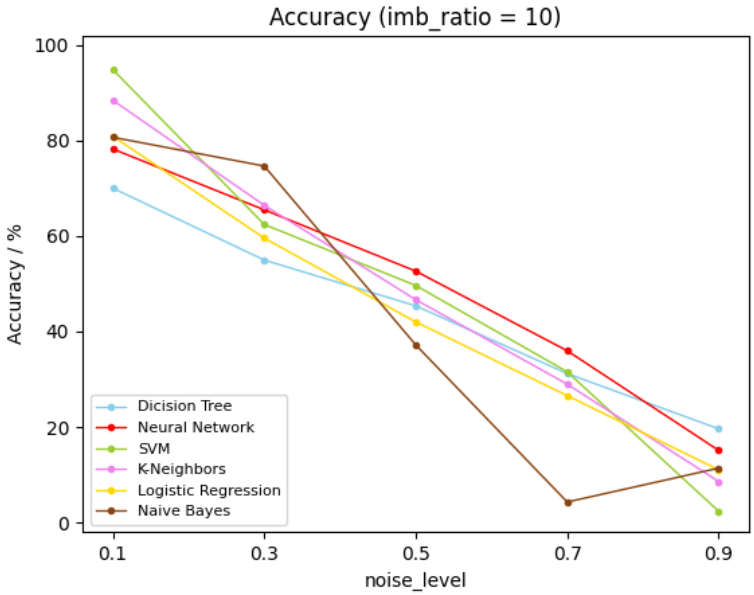
④ `noise_level = 0.7`

	时间开销（Time Cost）/s	准确率（Accuracy）
决策树（Dicision Tree）	0.035903	31.19%
神经网络（Neural Network）	2.034560	35.96%
支持向量机（SVM）	0.161568	31.56%
K-近邻（K-Neighbors）	0.000996	28.96%
逻辑回归（Logistic Regression）	0.294213	26.56%
朴素贝叶斯（Naive Bayes）	0.000997	4.29%

⑤ `noise_level = 0.9`

	时间开销（Time Cost）/s	准确率（Accuracy）
决策树（Dicision Tree）	0.029921	19.61%
神经网络（Neural Network）	2.051514	15.09%
支持向量机（SVM）	0.102725	2.34%
K-近邻（K-Neighbors）	0.000997	8.58%
逻辑回归（Logistic Regression）	0.270279	11.03%
朴素贝叶斯（Naive Bayes）	0.001993	11.38%

将准确率用折线图展示：



可见，随着 `noise_level` 的增大，所有模型的准确率均大幅下降，但朴素贝叶斯模型在 `noise_level` 过大时，准确率反而有所上升。

由上表可知，随着 `noise_level` 的增大，上述模型的时间开销大致为为先上升后下降的趋势。

（3）

1、类别不平衡的处理

固定 `imb_ratio = 90, noise_level = 0`

注：需提前安装 `Imbalanced-learn` 库

方法一：调节样本类别权重

代码文件：dicision_tree.py, svm.py, logistic_regression.py

代码运行方法：在下图中更换为带有参数 class_weight='balanced' 的语句后，直接运行即可（代码文件需位于 pendigits-corrupted-main 目录下）。

```
#model = DecisionTreeClassifier(criterion='entropy', random_state=14)
model = DecisionTreeClassifier(criterion='entropy', random_state=14, class_weight='balanced')
```

① 决策树（Dicision Tree）

	时间开销（Time Cost）/s	准确率（Accuracy）
调节权重前	0.009972	52.20%
调节权重后	0.010968	69.21%

② 支持向量机（SVM）

	时间开销（Time Cost）/s	准确率（Accuracy）
调节权重前	0.030917	81.79%
调节权重后	0.037898	85.76%

③ 逻辑回归（Logistic Regression）

	时间开销（Time Cost）/s	准确率（Accuracy）
调节权重前	0.182512	72.90%
调节权重后	0.207445	81.25%

由上述结果可知，

在调节样本类别权重后，各模型的准确率均有所提高，但时间开销也略有增多。

方法二：过（上）采样（基于 SMOTE）

方法三：欠（下）采样 （基于随机欠采样）

方法四：过采样和欠采样结合（基于 SMOTEENN）

代码文件：dicision_tree.py, neural_network.py, svm.py, k_neighbors.py, logistic_regression.py, naive_bayes.py

代码运行方法：选择下图中的需要的语句后，直接运行即可（代码文件需位于 pendigits-corrupted-main 目录下）。

```
# 样本类别不平衡处理
#input_train, target_train = SMOTE(random_state=14).fit_resample(input_train, target_train)
#input_train, target_train = RandomUnderSampler(random_state=14).fit_resample(input_train, target_train)
#input_train, target_train = SMOTEENN(random_state=14).fit_resample(input_train, target_train)
```

① 准确率（Accuracy）

	处理前	过采样	欠采样	过采样+欠采样
决策树（Dicision Tree）	52.20%	70.18%	55.37%	70.10%
神经网络（Neural Network）	81.33%	83.93%	78.64%	84.31%
支持向量机（SVM）	81.79%	84.88%	75.19%	85.19%
K-近邻（K-Neighbors）	74.84%	87.76%	70.95%	87.54%
逻辑回归（Logistic Regression）	72.90%	80.70%	76.21%	80.79%
朴素贝叶斯（Naive Bayes）	50.37%	45.97%	43.94%	46.14%

② 时间开销（Time Cost）

	处理前	过采样	欠采样	过采样+欠采样
决策树（Dicision Tree）	0.009973	0.118682	0.003989	0.113696
神经网络（Neural Network）	2.419537	5.166189	0.045877	3.555489
支持向量机（SVM）	0.063827	0.124666	0.001994	0.123667
K-近邻（K-Neighbors）	0.000997	0.000997	0.000997	0.000997
逻辑回归（Logistic Regression）	0.192523	0.950470	0.013962	2.419534
朴素贝叶斯（Naive Bayes）	0.000997	0.012957	0.000997	0.008978

由上述结果可知，

在以上的模型中，过采样和过采样+欠采样的处理让准确率有一定的提高，但欠采样的处理基本没有效果，甚至由负面效果；此外，时间开销均有所增大。

2、标签噪声的处理

常见的噪声数据的处理方法：

① 人工检查：人为的进行数据筛选。

② 统计模型：对于正态数据，利用3个标准差原则进行去噪，或使用四分位差进行去噪。

③ 分箱：通过考察相邻数据来确定最终值。

④ 聚类：将类似的值组织成群或“簇”，那些落在簇之外的值（孤立点），将被视为噪声。

⑤ 回归 ： 用一个函数拟合数据来光滑数据。

（4）

代码文件：imb_problem.py

代码运行方法：直接运行即可（代码文件需位于 anonymous-dataset 目录下）。

采用支持向量机（SVM）的方法，模型参数加上 `class_weight='balanced'`，然后进行预测。