机器学习导论结课作业

韦俊林 (201928016029023)

2020 年 6 月 24 日

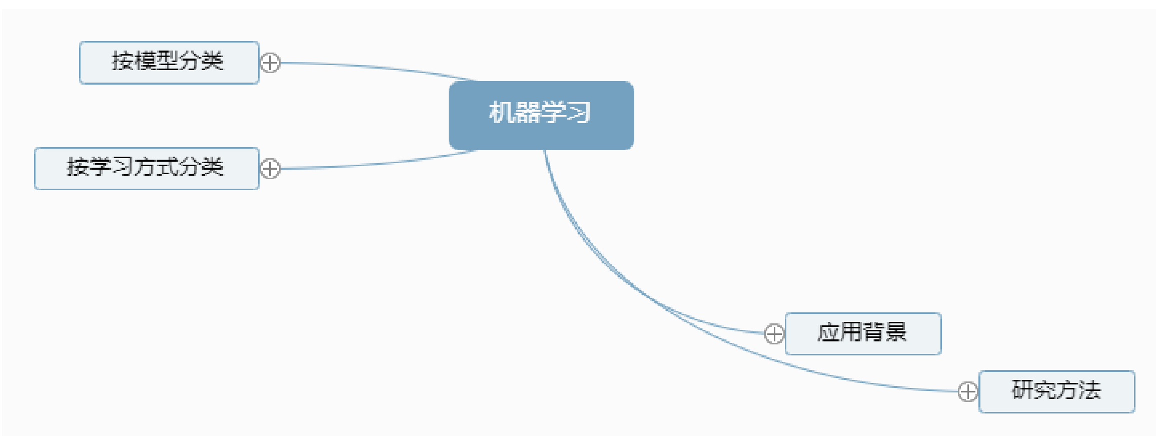
本课程基本知识组织

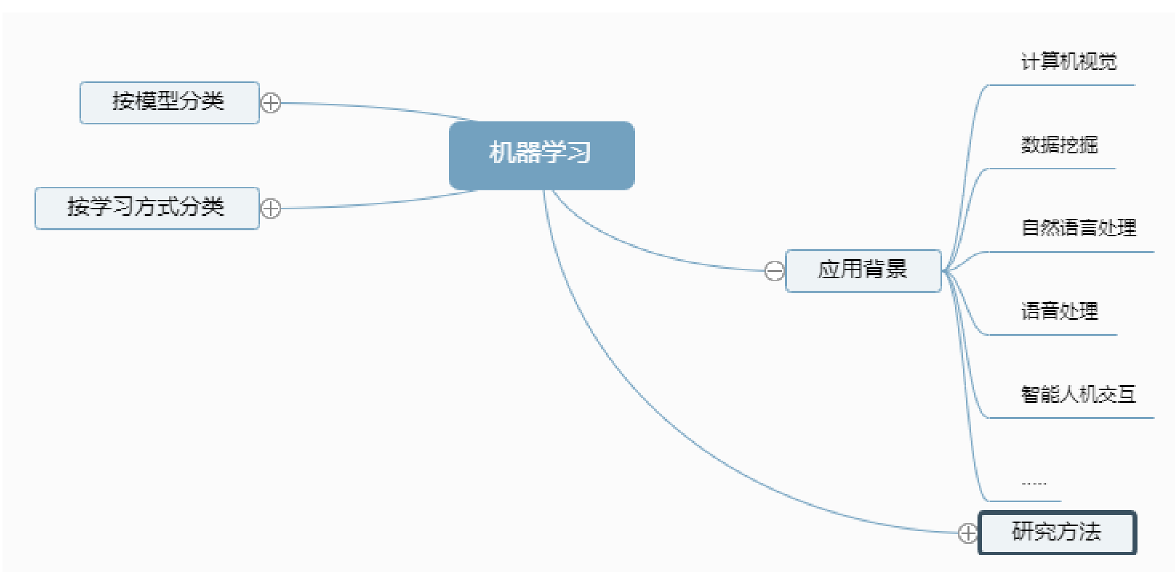
任务： 根据本学期所学《机器学习导论》的内容，制作一颗知识树，它能够反应“机器学习”的重要任务和重要概念。具体要求如下：

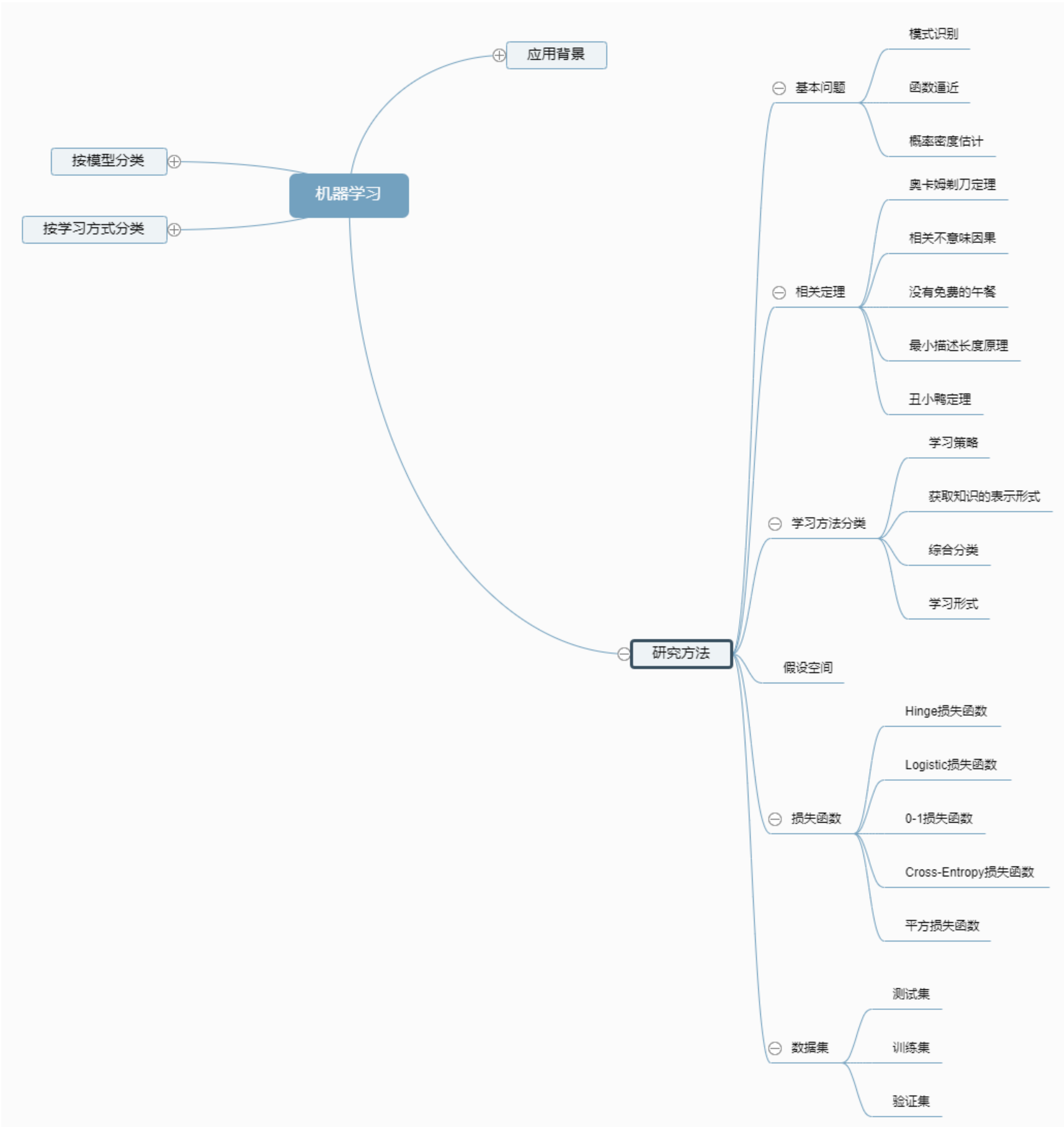
具体要求 1： 知识树的层次要清晰；

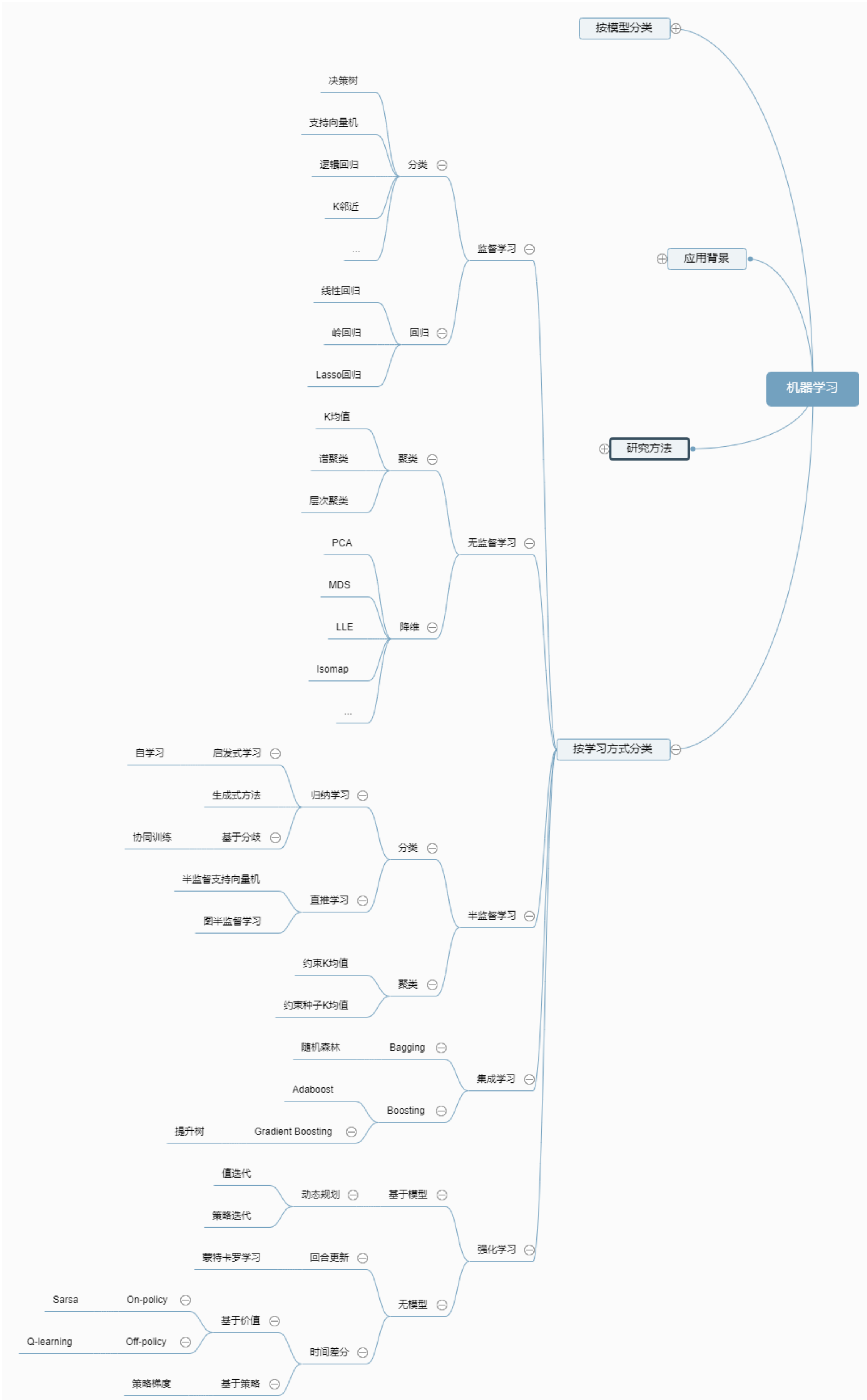
具体要求 2： 应能体现本学期课程内容。

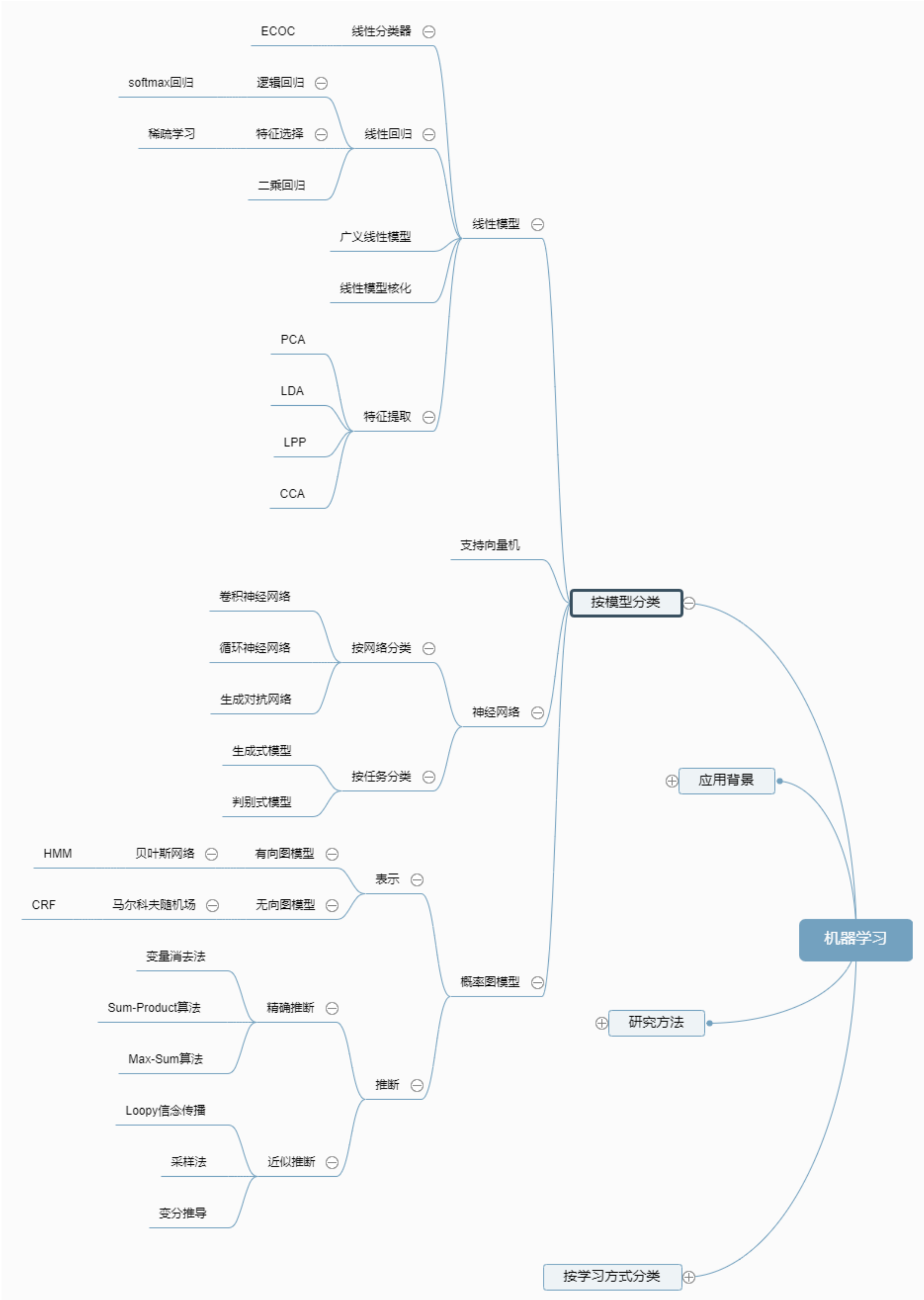
答：

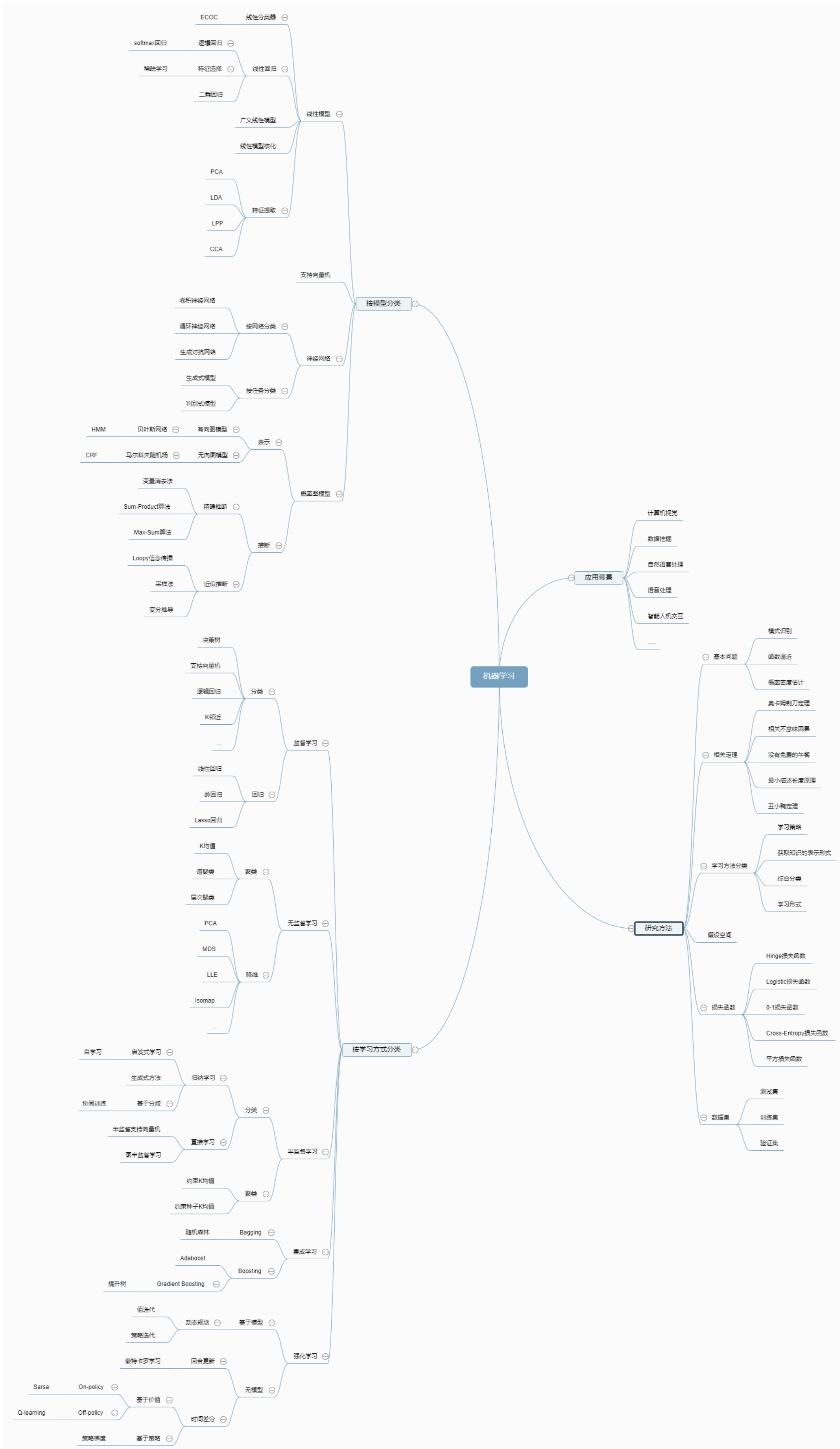





机器学习
应用背景
按模型分类
按学习方式分类
研究方法
基本问题
模式识别
函数逼近
概率密度估计
相关定理
奥卡姆剃刀定理
相关不意味因果
没有免费的午餐
最小描述长度原理
丑小鸭定理
学习方法分类
学习策略
获取知识的表示形式
综合分类
学习形式
假设空间
损失函数
Hinge损失函数
Logistic损失函数
0-1损失函数
Cross-Entropy损失函数
平方损失函数
数据集
测试集
训练集
验证集


机器学习
按模型分类
应用背景
研究方法
按学习方式分类
监督学习
分类
决策树
支持向量机
逻辑回归
K邻近
...
回归
线性回归
岭回归
Lasso回归
无监督学习
聚类
K均值
谱聚类
层次聚类
降维
PCA
MDS
LLE
Isomap
...
半监督学习
分类
归纳学习
启发式学习
自学习
生成式方法
基于分歧
协同训练
直推学习
半监督支持向量机
图半监督学习
聚类
约束K均值
约束种子K均值
集成学习
Bagging
随机森林
Boosting
Adaboost
Gradient Boosting
提升树
强化学习
基于模型
动态规划
值迭代
策略迭代
无模型
回合更新
蒙特卡罗学习
时间差分
基于价值
On-policy
Sarsa
Off-policy
Q-learning
基于策略
策略梯度


机器学习
按模型分类
线性模型
线性分类器
ECOC
线性回归
逻辑回归
softmax回归
特征选择
稀疏学习
二乘回归
广义线性模型
线性模型核化
特征提取
PCA
LDA
LPP
CCA
支持向量机
神经网络
按网络分类
卷积神经网络
循环神经网络
生成对抗网络
按任务分类
生成式模型
判别式模型
概率图模型
表示
有向图模型
贝叶斯网络
HMM
无向图模型
马尔科夫随机场
CRF
推断
精确推断
变量消去法
Sum-Product算法
Max-Sum算法
近似推断
Loopy信念传播
采样法
变分推导
应用背景
研究方法
按学习方式分类



机器学习
按模型分类
线性模型
线性分类器
ECOC
线性回归
逻辑回归
softmax回归
特征选择
稀疏学习
二乘回归
广义线性模型
线性模型核化
特征提取
PCA
LDA
LPP
CCA
支持向量机
神经网络
按网络分类
卷积神经网络
循环神经网络
生成对抗网络
按任务分类
生成式模型
判别式模型
概率图模型
表示
有向图模型
贝叶斯网络
HMM
无向图模型
马尔科夫随机场
CRF
推断
精确推断
变量消去法
Sum-Product算法
Max-Sum算法
近似推断
Loopy信念传播
采样法
变分推导
应用背景
计算机视觉
数据挖掘
自然语言处理
语音处理
智能人机交互
.....
研究方法
基本问题
模式识别
函数逼近
概率密度估计
相关定理
奥卡姆剃刀定理
相关不意味因果
没有免费的午餐
最小描述长度原理
丑小鸭定理
学习方法分类
学习策略
获取知识的表示形式
综合分类
学习形式
假设空间
损失函数
Hinge损失函数
Logistic损失函数
0-1损失函数
Cross-Entropy损失函数
平方损失函数
数据集
测试集
训练集
验证集
按学习方式分类
监督学习
分类
决策树
支持向量机
逻辑回归
K邻近
...
回归
线性回归
岭回归
Lasso回归
无监督学习
聚类
K均值
谱聚类
层次聚类
降维
PCA
MDS
LLE
Isomap
...
半监督学习
分类
归纳学习
启发式学习
自学习
生成式方法
基于分歧
协同训练
直推学习
半监督支持向量机
图半监督学习
聚类
约束K均值
约束种子K均值
集成学习
Bagging
随机森林
Boosting
Adaboost
Gradient Boosting
提升树
强化学习
基于模型
动态规划
值迭代
策略迭代
无模型
回合更新
蒙特卡罗学习
时间差分
基于价值
On-policy
Sarsa
Off-policy
Q-learning
基于策略
策略梯度

编程实践

编程实践所涉及的数据集如表 1 所示。

Table 1: 编程实践所需的数据集

数据集	类别数	特征维数	训练样本数目	测试样本数目
MNIST	10	784	$60,000$	$10,000$
Letter Recognition	26	16	$16,000$	4000

请完成以下两个任务：

任务 1： 编程实现一个分类器算法，该分类器由主成分分析 (Principal Component Analysis, PCA)、线性判别分析 (Linear Discriminant Analysis, LDA) 和 K 邻近 (K-Nearest Neighbor, KNN) 分类器共同完成，即 PCA+LDA+K-NN。具体过程如下：首先，对数据的原始特征采用 PCA 进行降维；然后，以 PCA 的降维结果作为输入，采用 LDA 提取判别特征；最后，以 LDA 提取的鉴别特征为输入，采用 K 邻近分类器完成最后的分类。

具体要求 1： 写出 PCA 的基本原理、核心公式和主要计算过程；

具体要求 2： 写出 LDA 的基本原理、核心公式和主要计算过程；

具体要求 3： 在实验过程中，在采用 PCA 对原始数据进行降维时，需按表 2 所示将数据降至不同的维度。进一步，针对 PCA 降维所获得的不同维度的数据，分别执行“LDA+K-NN”，并报告所获得的识别精度。另外，在实验过程中，K-NN 分类器所需要采用最近邻 (1-NN) 分类器和 3 近邻 (3-NN) 分类器来分别进行实验。最后，写出上述实验过程的主要步骤；

具体要求 4： 对实验结果进行分析，并以附录 A 的形式提交源代码。

Table 2: PCA 降维数

数据集	PCA 降维数
MNIST	50 维、100 维、200 维、300 维、400 维
Letter Recognition	3 维、5 维、7 维、9 维、11 维

答： PCA(Principal Component Analysis) 即主成分分析，基本思想是仅用一个超平面就能从整体上对所有样本 $\{x_1, x_2, \ldots, x_n \in \mathbb{R}^m\}$ 进行恰当的表示。通常包含两种思路求解，可重构性和可区分性。前者表示样本到这个超平面的距离都足够近，后者表示样本点在这个超平面的投影能够尽可能地分开。
对高维空间中的样本 x 进行线性变换，

$$y = W^T x,\ where\, x \in \mathbb{R}^m,\, W \in \mathbb{R}^{m\times d},\, y \in \mathbb{R}^d,\, d < m$$

变换矩阵 $W = [w_1, w_2, \ldots, w_d]$ 可视为 m 维空间中由 d 个基向量组成的矩阵。$y = W^T x$ 可视为样本 x 与 d 个基向量分别做内积运算而得，即 x 在新坐标系下的坐标。

可重构性： 假定投影变换时正交变换，即新坐标系由 $W = [w_1, w_2, \ldots, w_d]$ 表示 $d < m$，w_i 的模等于 1，w_i 与 w_j 两两正交。

设样本点 x_i 在新坐标系下的坐标为：

$$y_i = [y_{i1}, y_{i2}, \ldots, y_{id}]^T \in \mathbb{R}^d$$

在正交坐标系下，对样本点 x_i，有新坐标：

$$y_{ij} = w_j^T x_i,\ w_j \in \mathbb{R}^m,\ j = 1, 2, \ldots, d$$

在新坐标系下，可得 x_i 的新表示：

$$\hat{x_i} = \sum_{j=1}^{d} y_{ij} w_j,\ i = 1, 2, \ldots, n$$

这样，重构误差为：

$$\sum_{i=1}^{n} \|x - \hat{x}_i\|_2^2 = \sum_{i=1}^{n} \left\| x_i - \sum_{j=1}^{d} y_{ij} w_j \right\|_2^2 = -tr\left(W^T \sum_{i=1}^{n} x_i x_i^T W \right) + const$$

进一步，假定数据已经零均值化，令 $X = [x_1, x_2, \ldots, x_n] \in \mathbb{R}^{m\times n}$。于是，得主成分分析的最优模型：

$$\max_{W\in\mathbb{R}^{m\times d}} tr\left(W^T X X^T W\right),\ s.t. W^T W = I$$

可区分性：使所有样本点的投影尽可能地分开，则需最大化投影点的方差，设投影后获得的样本点为：

$$y_i = W^T x_i \in \mathbb{R}^d,\ i = 1, 2, \ldots, n$$

由于数据点零均值化，有 $\sum_{i=1}^{n} y_i = W^T \sum_{i=1}^{n} x_i = 0$。因此，投影后的样本点的协方差为，

$$\sum_{i=1}^{n} W_T x_i x_i^T W = W^T X X^T W$$

要使得数据具有最大可分性，就应使其方差最大，考虑多维情形，由此有：

$$\max_{W\in\mathbb{R}^{m\times d}} tr\left(W^T X X^T W\right),\ s.t. W^T W = I$$

PCA 的求解可以采用拉格朗日乘子法，因此有：

$$X X^T W = \lambda W$$

然后对协方差矩阵 XX^T 进行特征值分解，并对特征值进行排序：$\lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_d$，取前 d 个特征值对应的特征向量构成变换矩阵 W。

LDA(Linear Discriminant Analysis)，即线性判别分析。基本思想是对于二分类问题，给定训练集，设法将样例投影到一条直线上，使得同类样例的投影点尽可能接近，不同类样例的投影点尽可能相互远离。对于新样本进行分类时，将其投影到这条直线上，再根据投影点的位置来判断其类别。

对于样本集 $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}, y_i \in \{0, 1\}$，令 X_i, μ_i, Σ_i 分别表示第 $i \in \{0, 1\}$ 类的示例集合、均值向量以及协方差矩阵。考虑线性变换 $y = w^T x$，对两类样本

的中心点，在直线上的投影将分别为 $w^T\mu_0$ 和 $w^T\mu_1$。要使得同类样本的投影点尽可能接近，可以让同类样本投影点的协方差尽可能小，即 $w^T\Sigma_0 w + w^T\Sigma_1 w$。要使得异类样本的投影点尽可能远离，则可让类中心点之间的距离尽可能大，即 $\|w^T\mu_0 - w^T\mu_1\|^2$ 尽可能大。因此，目标为最大化如下函数：

$$J(w) = \frac{\|w^T\mu_0 - w^T\mu_1\|_2^2}{w^T\Sigma_0 w + w^T\Sigma_1 w} = \frac{w^T(\mu_0-\mu_1)(\mu_0-\mu_1)^T w}{w^T(\Sigma_0+\Sigma_1)w}$$

设类内散度矩阵为：

$$S_w = \Sigma_0 + \Sigma_1 = \sum_{x\in X_0}(x-\mu_0)(x-\mu_0)^T + \sum_{x\in X_1}(x-\mu_1)(x-\mu_1)^T$$

类间散度矩阵：

$$S_b = (\mu_0-\mu_1)(\mu_0-\mu_1)^T$$

因为 $J(w)$ 与向量长度无关，不失一般性令 w 为单位向量。因此，目标函数可重写为：

$$J(w) = \frac{w^T S_b w}{w^T S_w w},\ s.t.\, w^T w = 1$$

更进一步，令 $w^T S_w w = 1$，得：

$$\max\ w^T S_b w,\ s.t.\, w^T S_w w = 1$$

根据拉格朗日乘子法可得：

$$S_b w = \lambda S_w w \Rightarrow S_w^{-1} S_b w = \lambda w$$

最终，

$$w = S_w^{-1}(\mu_0 - \mu_1)$$

对于多类别的 LAD 算法，假设有 c 个类别。定义全局散度矩阵为：

$$S_t = S_w + S_b = \sum_{i=1}^{n}(x_i-\mu)(x_i-\mu)^T,\ \mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

类内散度矩阵：

$$S_w = \sum_{j=1}^{c} S_{wj},\ S_{wj} = \sum_{x\in X_j}(x-\mu_j)(x-\mu_j)^T,\ \mu_j = \frac{1}{n_j}\sum_{x\in X_j} x$$

类间散度矩阵：

$$S_b = S_t - S_w = \sum_{j=1}^{c} n_j(\mu_j-\mu)(\mu_j-\mu)^T$$

其中 n_j 为属于第 j 类的样本个数。
因此，可得两个目标函数，
迹比值最大化目标函数：

$$\max \frac{tr(W^T S_b W)}{tr(W^T S_b W)},\ s.t.\, W^T W = I$$

行列式比值最大化目标函数：

$$\max \frac{|W^T S_b W|}{|W^T S_w W|},\ s.t.\, W^T W = I$$

实验报告： 完成本次实验的程序包含 3 个文件，分别是 main.py, model.py, utils.py。
main.py： 主函数，实现一个配置类，用于初始化实验参数以及实现一个 assignment_1 成员函数，用于初始化、训练、测试本次实验相应的模型，并输出结果；
model.py： 实现 PCA+LDA+KNN 模型，基于 sklearn 库实现本次实验的模型，实现一个 PCA_LDA_KNN 类，主要包括两个成员函数 train 和 acc，分别用于训练模型和测试模型；
utils.py： 一个辅助文件，实现数据集的导入以及一些辅助函数，load_dataset 函数用于导入数据集，load_MNIST_img 和 load_MNIST_label 函数用于辅助导入、预处理 MNIST 数据集相应的图像和标签，mkdir 函数用于创建保存结果的文件夹，randomcolor 函数用于随机颜色，output_assignment_1 函数用于输出实验结果。
数据集保存在该项目当前目录下的 dataset 文件夹中，并分别存在以两个数据集名称命名的两个文件夹中。
初始化实验相关参数，如下图所示：

```python
class config():
    def __init__(self, save_result):
        self.save_path = mkdir() if save_result else None
        self.dataset = ['Letter Recognition', 'MNIST']

    def config_1(self):
        self.PCA_components = {'MNIST': [50, 100, 200, 300, 400], \
                    'Letter Recognition': [3, 5, 7, 9, 11]}
        self.KNN_neighbors = [1, 3]
```

模型的实现，如下图所示：

```python
class PCA_LDA_KNN():
    def __init__(self, n_components, n_neighbors):
        self.model_pca = PCA(n_components=n_components)
        self.model_lda = LDA()
        self.model_knn = KNN(n_neighbors=n_neighbors)

    def train(self, x_train, y_train):
        data_pca = self.model_pca.fit_transform(x_train)
        data_lda = self.model_lda.fit_transform(data_pca, y_train)
        self.model_knn.fit(data_lda, y_train)

    def acc(self, x_test, y_test):
        x_pca = self.model_pca.transform(x_test)
        x_lda = self.model_lda.transform(x_pca)
        return self.model_knn.score(x_lda, y_test)

    def pred(self):
        pass
```

导入 Letter Recognition 数据集，并对数据即进行预处理。即，将字母 A~Z 转换为 $0 \sim 25$，如下图所示：

```python
if dataset_name.title() == 'Letter Recognition':
    print('loading dataset: Letter Recognition')
    data_path = path.join(PATH, dataset_name.title(), 'letter-recognition.data')

    x_data = np.loadtxt(fname=data_path, dtype=float, delimiter=',', usecols=range(1,17))
    y_data = np.loadtxt(fname=data_path, dtype=str, delimiter=',', usecols=0)

    y_data = np.array([float(ord(y_data[i])-ord('A')) for i in range(len(y_data))])

    x_train = x_data[:-4000,:]
    y_train = y_data[:-4000]
    x_test = x_data[-16000:,:]
    y_test = y_data[-16000:]

    return x_train, y_train, x_test, y_test
```

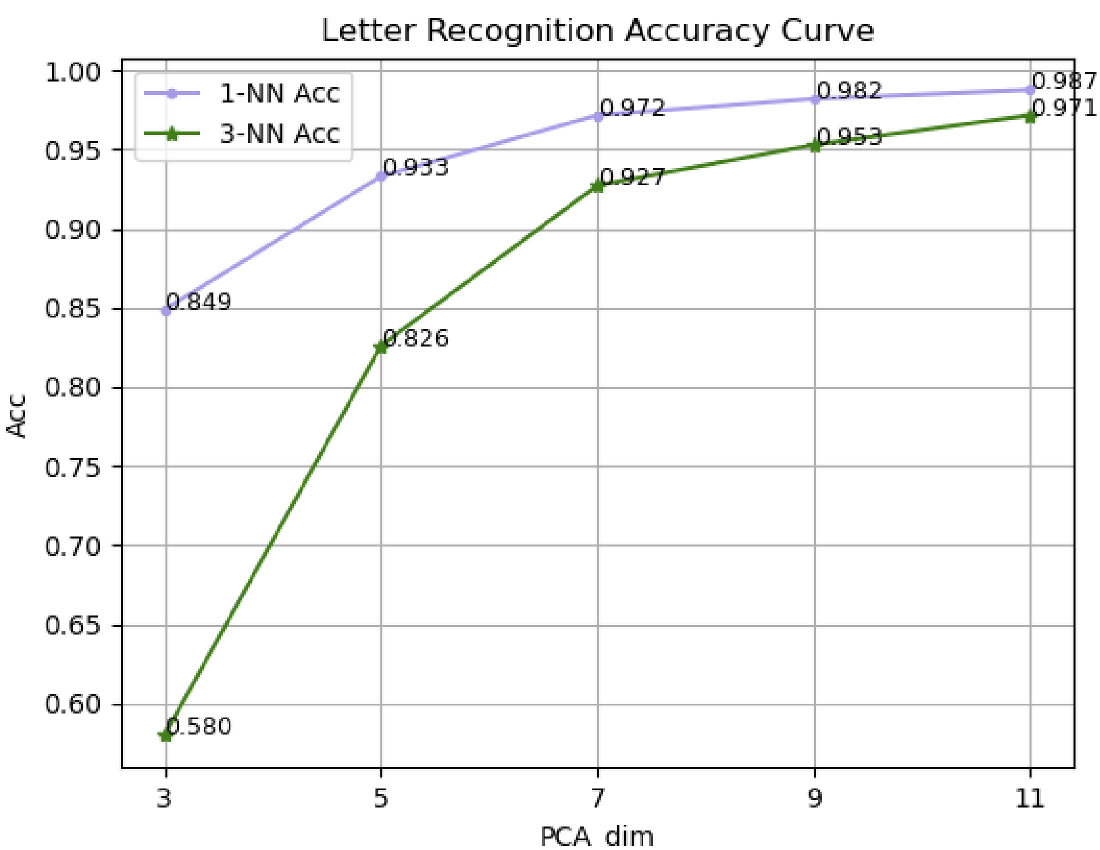
导入 MNIST 数据集，并对数据即进行预处理。即，将图像像素归一化 $0 \sim 1$ 之间，如下图所示：

```python
    elif dataset_name.upper() == 'MNIST':
        print('loading dataset: MNIST')

        TRAIN_IMAGES = path.join(PATH, dataset_name.upper(), 'train-images-idx3-ubyte.gz')
        TRAIN_LABELS = path.join(PATH, dataset_name.upper(), 'train-labels-idx1-ubyte.gz')
        TEST_IMAGES = path.join(PATH, dataset_name.upper(), 't10k-images-idx3-ubyte.gz')
        TEST_LABELS = path.join(PATH, dataset_name.upper(), 't10k-labels-idx1-ubyte.gz')

        x_train = load_MNIST_img(TRAIN_IMAGES)
        y_train = load_MNIST_label(TRAIN_LABELS)
        x_test = load_MNIST_img(TEST_IMAGES)
        y_test = load_MNIST_label(TEST_LABELS)

        return x_train, y_train, x_test, y_test

def load_MNIST_img(file_path):
    with gzip.open(file_path, 'rb') as bytestream:
        img_data=np.frombuffer(bytestream.read(), np.uint8, offset=16).astype(np.float32)
    #normalize : 将图像的像素归一化为 0.0~1.0
        img_data /= 255.0
    return img_data.reshape(-1, 784)

def load_MNIST_label(file_path):
    with gzip.open(file_path, 'rb') as bytestream:
        label_data=np.frombuffer(bytestream.read(), np.uint8, offset=8)
    return label_data
```

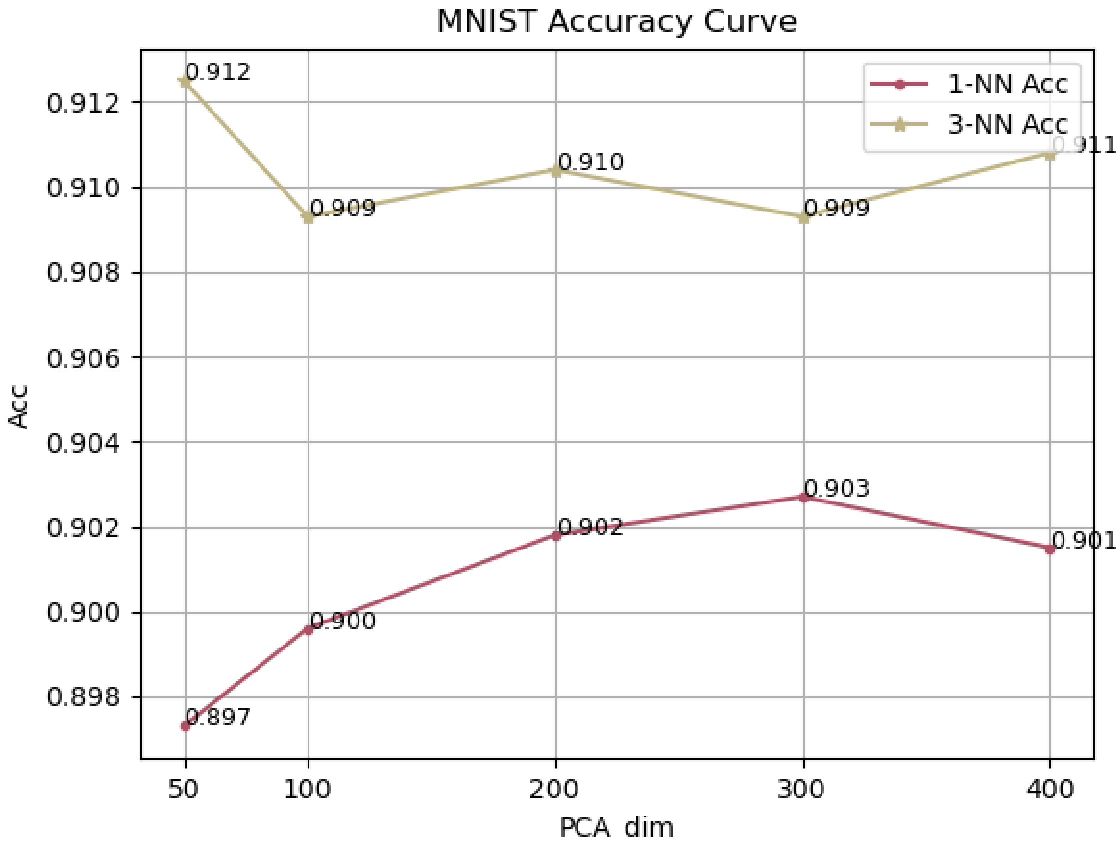
实验结果： Terminal 输出结果如下图所示，矩阵第一行表示 K-NN 不同的 K 值，第一列表示 PCA 的降维数，中间对应的单元表示相应的准确度。

```
start assignment_1: PCA+LDA+KNN
loading dataset: Letter Recognition
              1         3
3   0.848688  0.580000
5   0.932875  0.825937
7   0.971500  0.927125
9   0.981812  0.952812
11  0.987437  0.971437

loading dataset: MNIST
          1       3
50   0.8973  0.9125
100  0.8996  0.9093
200  0.9018  0.9104
300  0.9027  0.9093
400  0.9015  0.9108
```

Letter Recognition 数据集在不同参数得到的准确率图像，如下图所示：



MNIST 数据集在不同参数得到的准确率图像，如下图所示：



任务 2： 请采用前向神经网络方法编程实现对上述两个数据集的分类。

具体要求 1： 简述网络结构设计和网络训练步骤，给出误差反向传播算法的细节；

具体要求 2： 报告在不同学习率、不同隐含层结点个数等情形下的分类精度；

具体要求 3： 对实验结果进行分析，并以附录 B 的形式提交源码。

答： 本次实验设计的神经网络包括输入层、隐藏层和输出层，其中隐藏层只有一层网络，激活函数使用 ReLu，优化算法是随机梯度下降，损失函数为交叉熵损失函数。

训练过程，首先将训练集和测试集转换为 pytorch 相应的小批量形式的张量，定义好相应的优化算法和损失函数。接着开始训练网络，将训练集特征张量传入模型，得到预测输出值。再将该预测输出值与真实值传入损失函数，得到损失值。最后通过误差反向传播算法优化网络，这样完成一次训练迭代。完成一次训练迭代之后，利用测试集测试一次模型准确率。按照上述步骤不断迭代训练、测试网络。

包括输入层和输出层，本次实验总共是三层的神经网络。

对于三层前向神经网络，第 k 个样本，从输入层到输出层节点 j 的输出有：

$$z_j^k = f\left(\sum_h w_{hj} f\left(\sum_i w_{ih} x_i^k\right)\right)$$

其中，$f(*)$ 为激活函数，i 为遍历输入层结点，h 为遍历隐含层结点。w 为下标所连接两个结点的边的权重，x_i 为输入层输入数据。假设第 k 个样本输出层结点 j 输出的真实值为

t_j^k，则误差函数有：

$$E(w)^k = \frac{1}{2}\sum_{j,k}\left\{t_j^k - f\left(\sum_h w_{hj} f\left(\sum_i w_{ih} x_i^k\right)\right)\right\}^2$$

从而有隐含层到输出层的连接权重调节量：

$$\Delta w_{hj} = -\eta \frac{\partial E}{\partial w_{hj}} = \eta \sum_k \delta_j^k y_h^k$$

$$\delta_j^k = -\frac{\delta E}{\delta net_j^k} = f'(net_j^k)\Delta_j^k$$

$$net_j^k = \sum_h w_{hj} f(\sum_i w_{ih} x_i^k)\ , \quad \Delta_j^k = t_j^k - z_j^k$$

输入层到隐含层的连接权重调节量：

$$\Delta w_{ih} = -\eta \frac{\partial E}{\partial w_{ih}} = \eta \sum_k \delta_h^k x_i^k$$

$$\delta_h^k = f'(net_h^k)\Delta_h^k$$

$$net_h^k = \sum_i w_{ih} x_i^k\ , \quad \Delta_h^k = \sum_j w_{hj}\delta_j^k$$

其中 η 为学习率，激活函数为：

$$f(x) = \begin{cases} 0, & x \leqslant 0 \\ \text{x}, & x > 0 \end{cases}$$

实验报告： 完成本次实验的代码实现部分总共包含三个文件，分别是 main.py, model.py, utils.py。

main.py： 主函数，实现一个配置类，用于初始化实验参数以及实现一个 assignment_2 成员函数，用于初始化、训练、测试本次实验相应的模型，并输出结果；

model.py： 基于 pytorch 框架实现前向神经网络模型，主要包括两个成员函数 train 和 evaluate_accuracy，分别用于训练模型和测试模型；

utils.py： 一个辅助文件，同上个实验，不同在于实现 output_assignment_2 函数用于输出本次实验结果。

数据集保存在该项目当前目录下的 dataset 文件夹中，并分别存在以两个数据集名称命名的两个文件夹中。

网络实现，如下图所示：

```
class ForwardNeuralNetwork():
    def __init__(self, input_dim, num_hidden, output_dim):
        self.device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

        self.net = torch.nn.Sequential(
            torch.nn.Linear(input_dim, int(num_hidden*input_dim)),
            torch.nn.ReLU(),
            torch.nn.Linear(int(num_hidden*input_dim), output_dim)
        ).to(self.device)
```

训练模型核心代码：

```
optimizer = torch.optim.SGD(self.net.parameters(), lr=lr)

criterion = torch.nn.CrossEntropyLoss().to(self.device)

loss_list = []
acc_list = []
for epoch in range(num_epochs):
    count, loss_acc, start = 0, 0, time.time()
    # train
    for _, (feature, label) in enumerate(train_loader):
        X = Variable(feature).float().to(self.device)
        y_true = Variable(label).long().to(self.device)

        optimizer.zero_grad()

        y_hat = self.net(X)
        loss = criterion(y_hat, y_true.view(-1))

        loss.backward()
        optimizer.step()
```

测试模型函数代码：

```
def evaluate_accuracy(self, x_test, y_test, batch_size):

    x_test = torch.Tensor(x_test)
    y_test = torch.Tensor(y_test)
    test_set = TensorDataset(x_test, y_test)
    test_loader = DataLoader(test_set, shuffle=True, batch_size=batch_size)

    count, acc_count = 0, 0
    for _, (X, y_true) in enumerate(test_loader):

        output = self.net(X.to(self.device))
        output = torch.nn.functional.softmax(output, dim=1)

        y_pred = output.argmax(dim=1)
        result = torch.eq(y_pred, y_true.to(self.device)).float()

        count += 1
        acc_count += torch.mean(result).item()
    return acc_count/count
```

设置参数和待测试的超参数：“num_labels”表示类别数，“feature_dim”表示输入特征

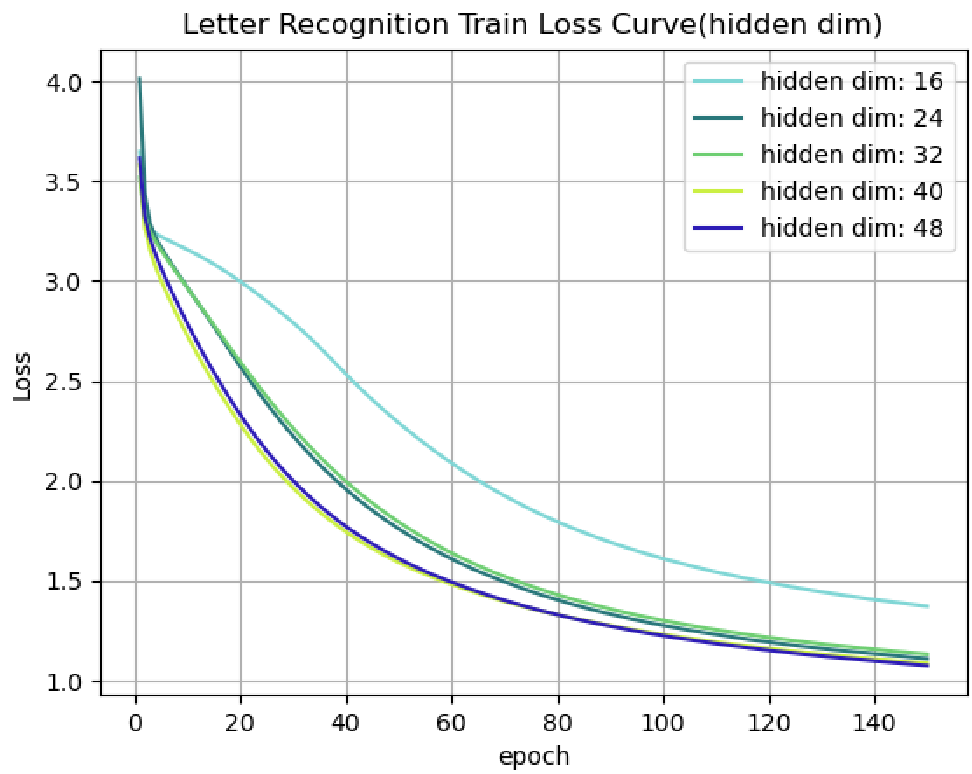
维数，“lr”表述学习率，“batch_size”表示小批量导入数据的大小，“num_epochs”表示迭代次数，“num_hidden”表示隐藏层维数，为对应数据集输入维数相应的倍数。

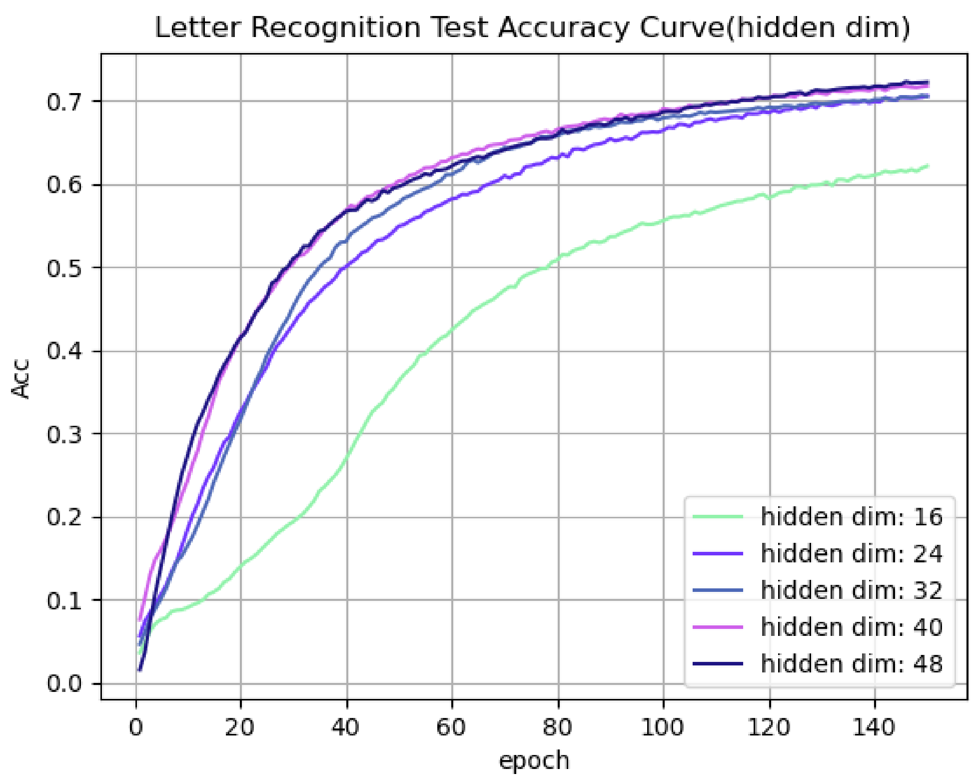
```
def config_2(self):
    self.num_labels = {'MNIST': 10, 'Letter Recognition': 26}
    self.feature_dim = {'MNIST': 784, 'Letter Recognition': 16}
    self.lr = [0.001, 0.01, 0.03, 0.05, 0.1]
    self.batch_size = [32, 64, 128, 256, 512]
    self.num_epochs = 150
    self.num_hidden = [1, 1.5, 2, 2.5, 3]
```

实验结果： 最大迭代次数为 150，默认学习率、小批量大小、隐含层维数分别为 0.001, 128 以及数据集输入特征维度的两倍。

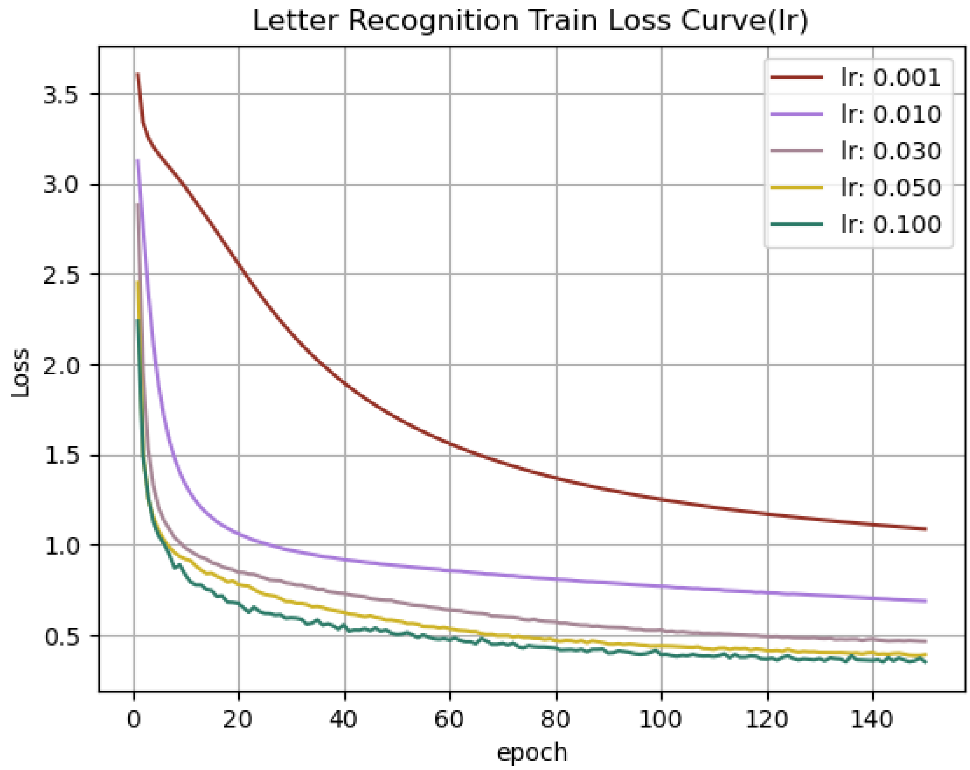
Letter Recognition 数据集：

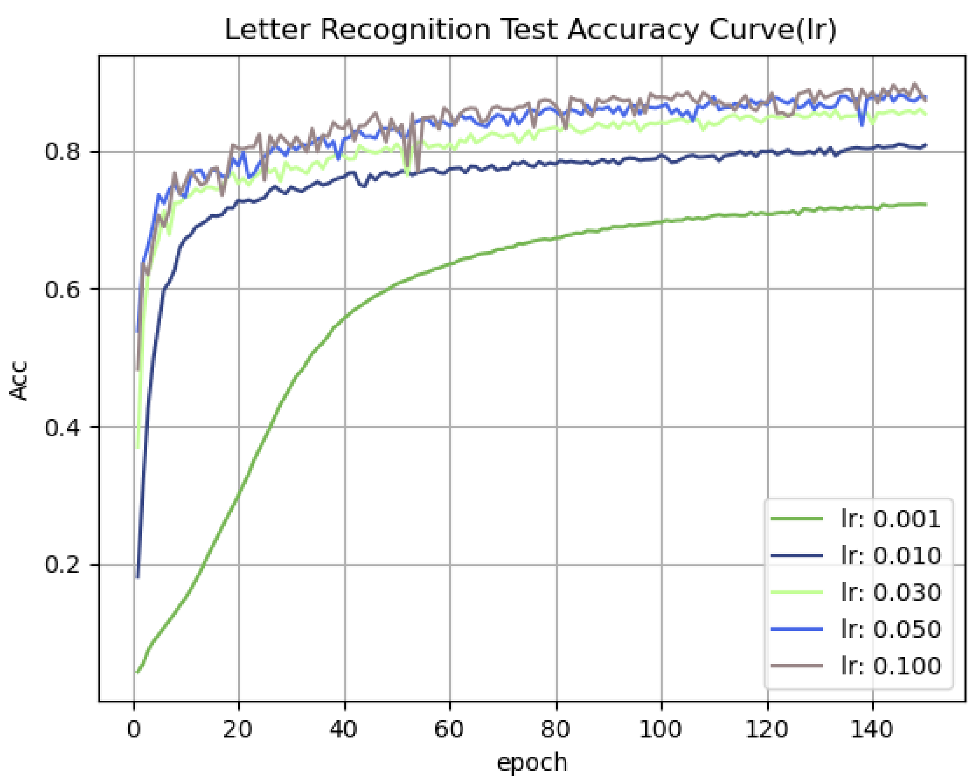
在默认学习率、小批量大小的条件下，隐藏层结点分别设置为 1×16, 1.5×16, 2×16, 2.5×16, 3×16 得到的训练集损失曲线以及测试集准确率曲线：


Letter Recognition Train Loss Curve(hidden dim)
Loss
epoch
hidden dim: 16
hidden dim: 24
hidden dim: 32
hidden dim: 40
hidden dim: 48
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
20
40
60
80
100
120
140

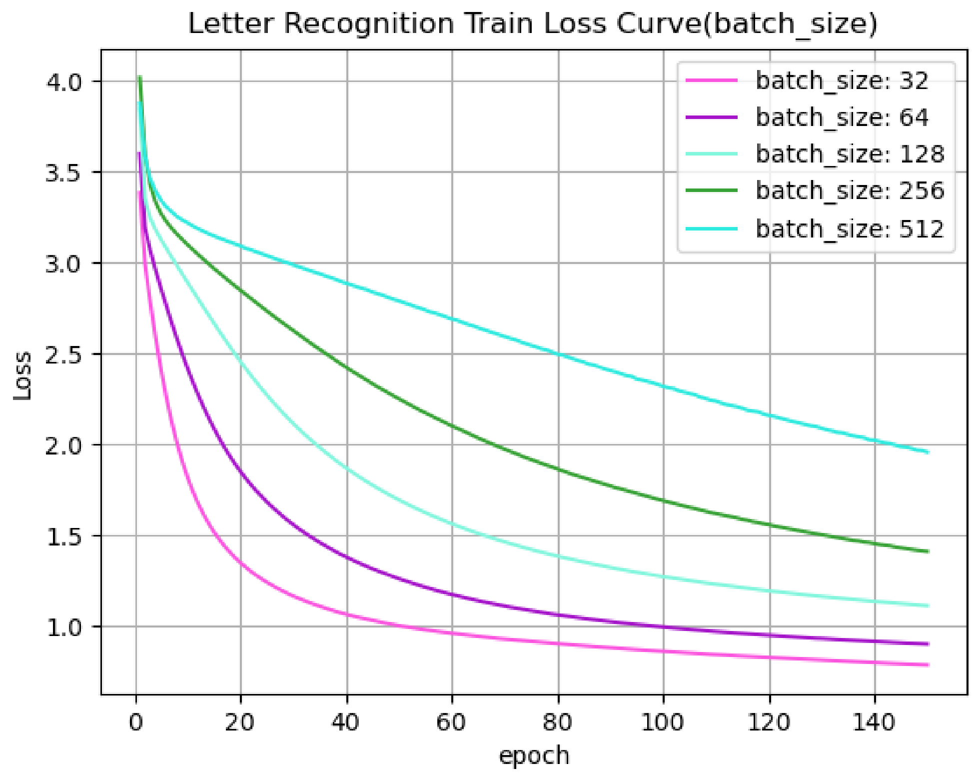

Letter Recognition Test Accuracy Curve(hidden dim)
Acc
epoch
hidden dim: 16
hidden dim: 24
hidden dim: 32
hidden dim: 40
hidden dim: 48
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
20
40
60
80
100
120
140

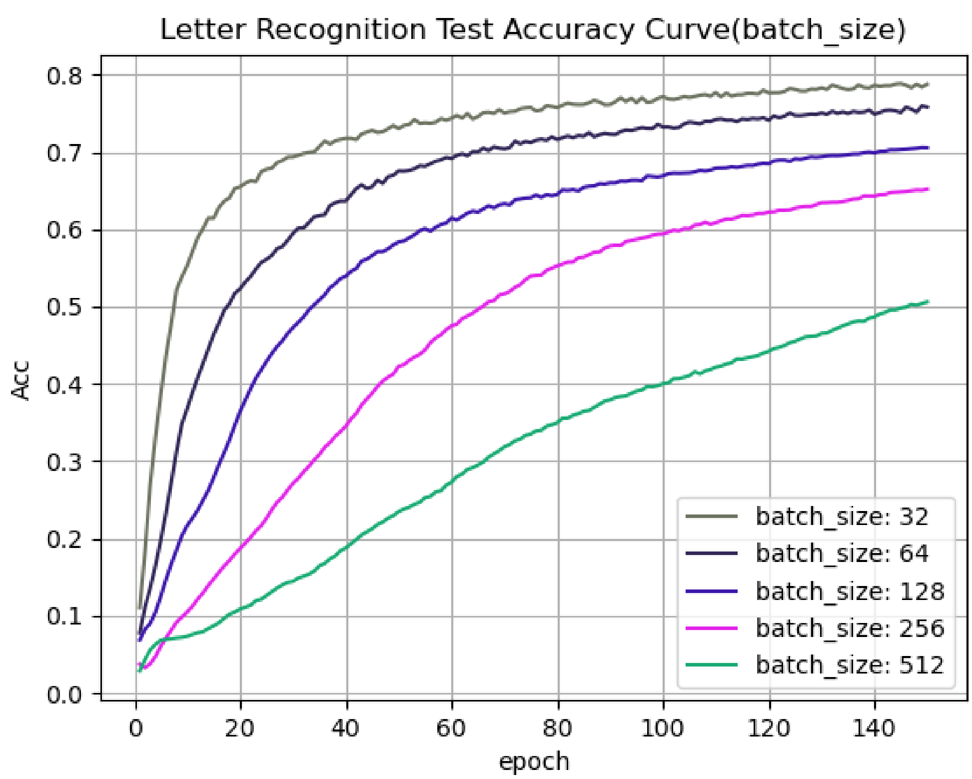
在默认小批量大小，隐藏层节点数 (2×16) 的条件下，学习率分别设置为 0.001, 0.01, 0.03, 0.05, 0.1 得到的训练集损失曲线以及测试集准确率曲线：





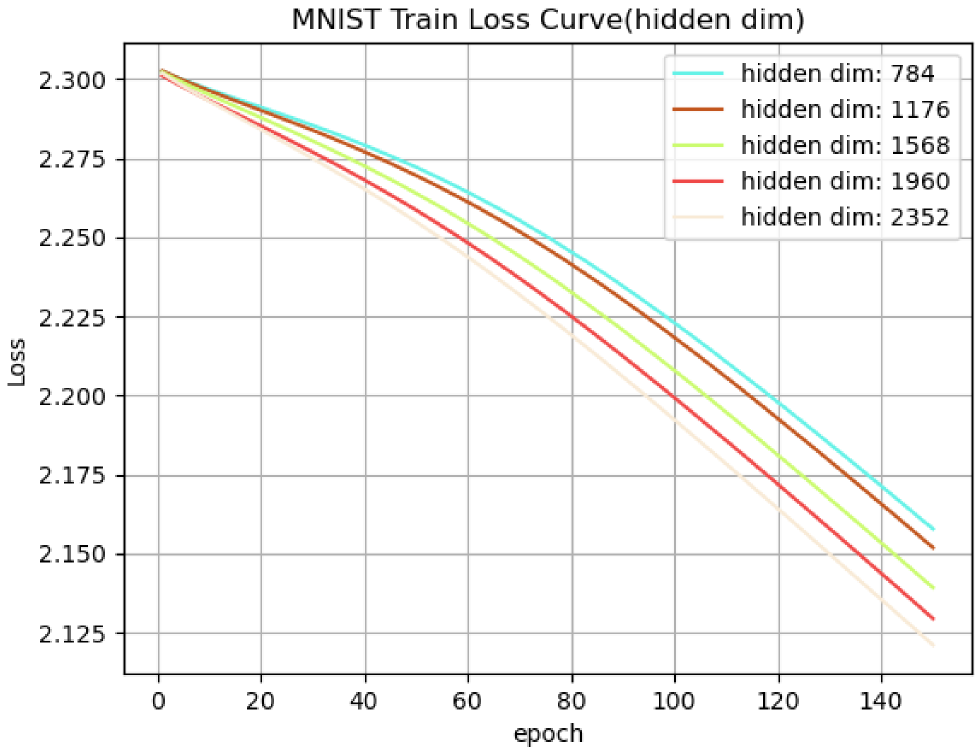
在默认学习率，隐藏层节点数 (2×16) 的条件下，小批量大小分别设置为 32, 64, 128, 256, 512 得到的训练集损失曲线以及测试集准确率曲线：

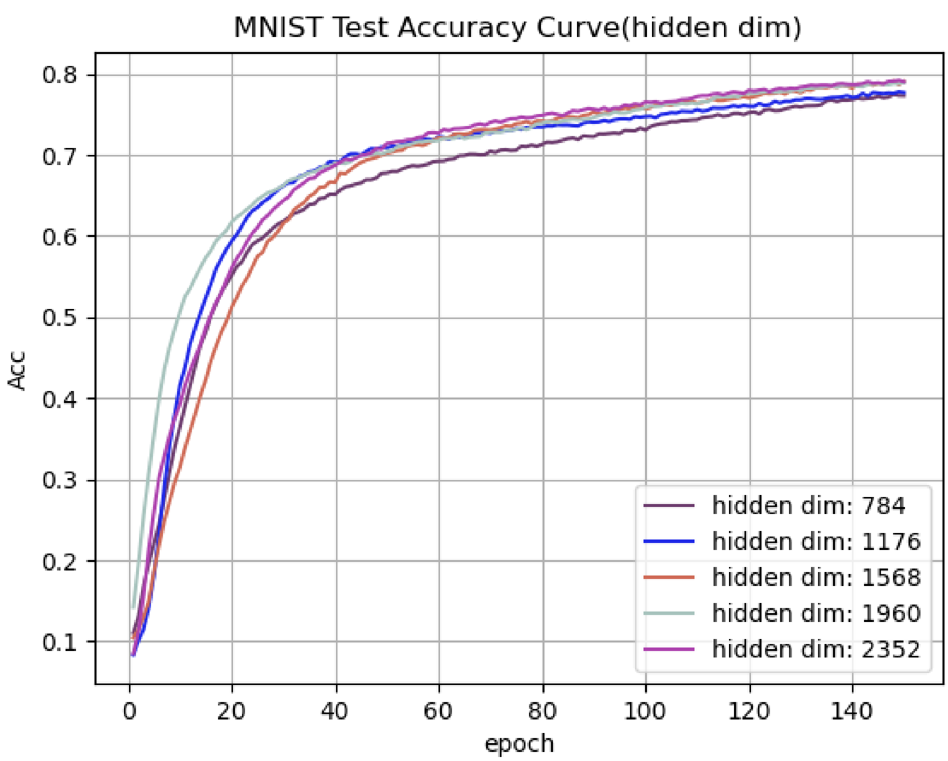




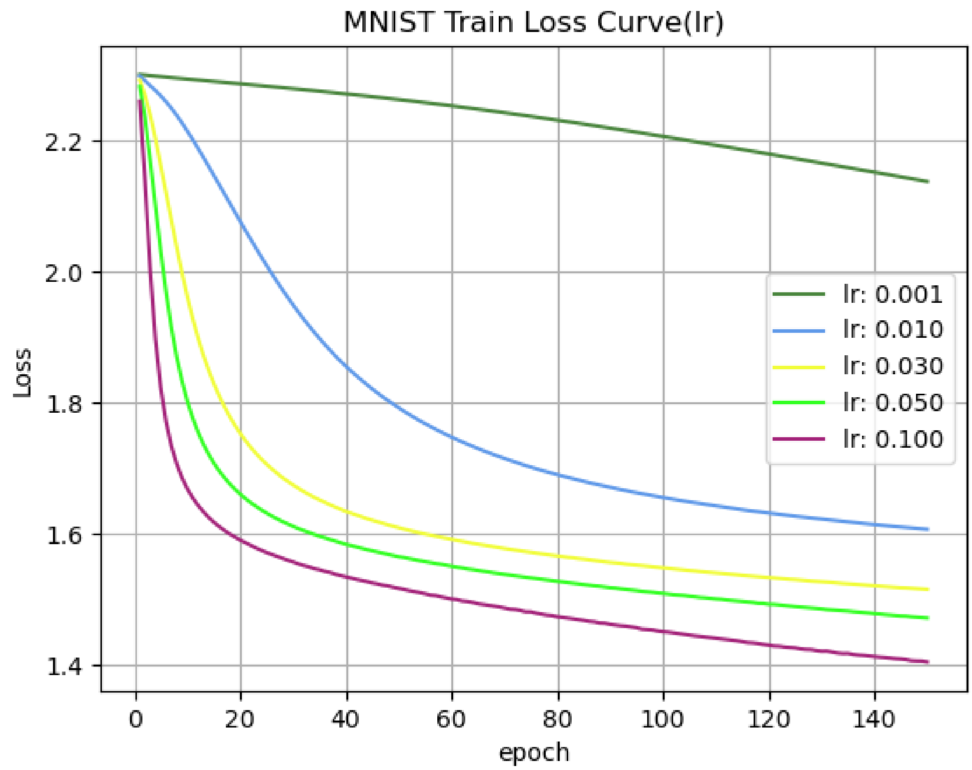
MNIST 数据集：

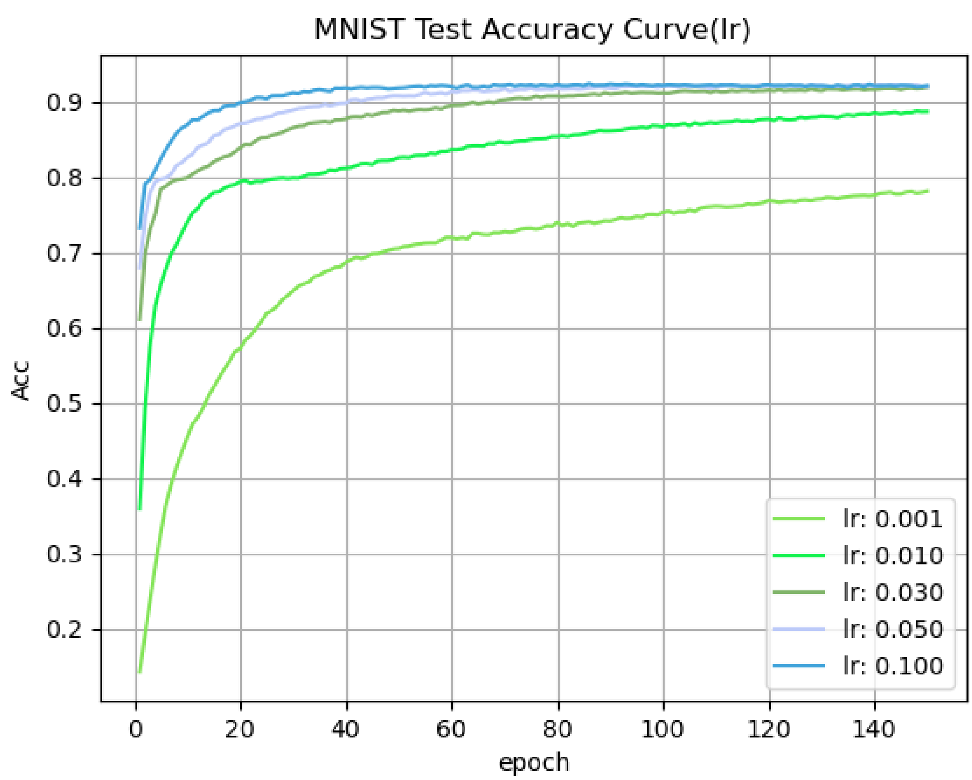
在默认学习率、小批量大小的条件下，隐藏层结点分别设置为 1×784, 1.5×784, 2×784, 2.5×784, 3×784 得到的训练集损失曲线以及测试集准确率曲线：



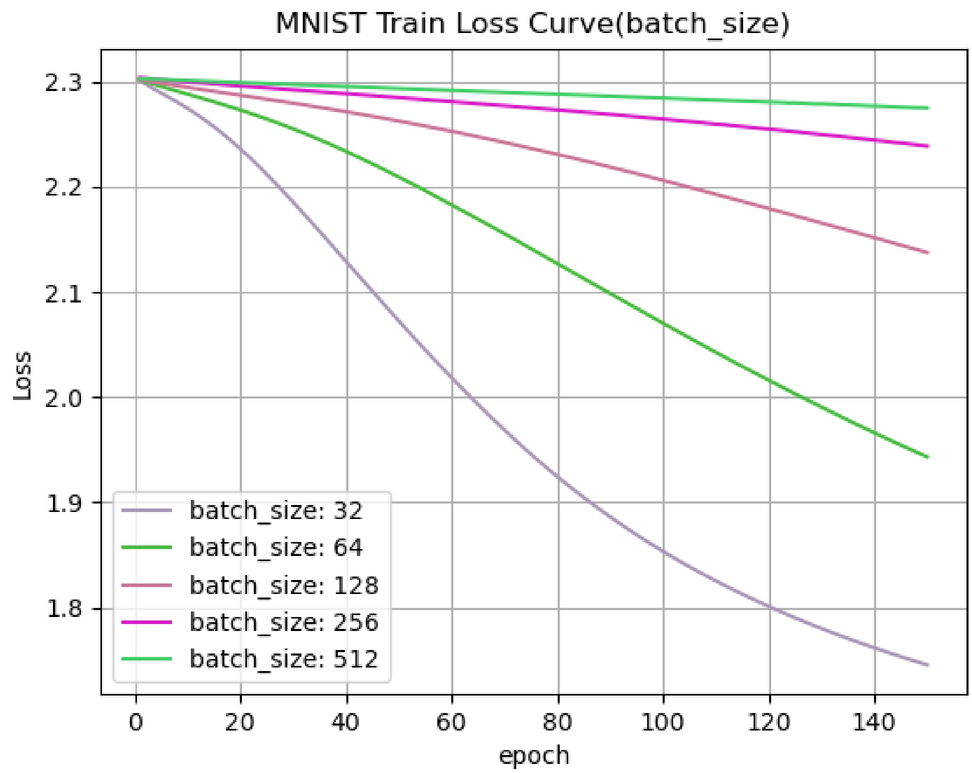


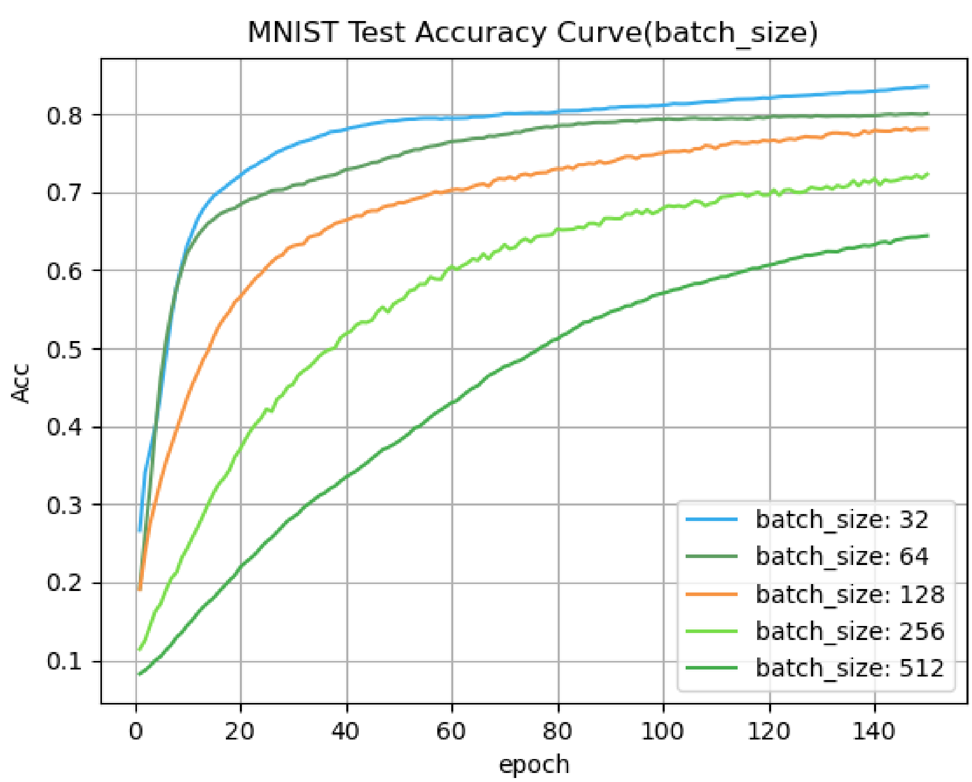
在默认小批量大小，隐藏层节点数 (2×784) 的条件下，学习率分别设置为 0.001, 0.01, 0.03, 0.05, 0.1 得到的训练集损失曲线以及测试集准确率曲线：





在默认学习率，隐藏层节点数 (2×784) 的条件下，小批量大小分别设置为 32, 64, 128, 256, 512 得到的训练集损失曲线以及测试集准确率曲线：





附录 A

main.py:

```
from utils import mkdir, load_dataset, output_assignment_1
from model import PCA_LDA_KNN

class config():
    def __init__(self, save_result):
        self.save_path = mkdir() if save_result else None
        self.dataset = ['Letter Recognition', 'MNIST']

    def config_1(self):
        self.PCA_components = {'MNIST': [50, 100, 200, 300, 400], \
                  'Letter Recognition': [3, 5, 7, 9, 11]}
        self.KNN_neighbors = [1, 3]

    def assignment_1(self):
        '''
        PCA+LDA+KNN parameters:
        PCA n_components,
            Letter Recognition: 3, 5, 7, 9, 11;
            MNIST: 50, 100, 200, 300, 400.
        K-NN n_neighbors: 1, 3.
        '''
        print('start assignment_1: PCA+LDA+KNN')
        self.config_1()

        for dataset_name in self.dataset:
            x_train, y_train, x_test, y_test = load_dataset(dataset_name)

            result = []

            for n_components in self.PCA_components[dataset_name]:
                acc_list = []
                for n_neighbors in self.KNN_neighbors:
                    model = PCA_LDA_KNN(n_components, n_neighbors)
                    model.train(x_train, y_train)
                    acc = model.acc(x_test, y_test)
                    acc_list.append(acc)

                result.append(acc_list)

            output_assignment_1(result, dataset_name, self.PCA_components[dataset_name], \
                                self.KNN_neighbors, save_path=self.save_path)
if __name__ == '__main__':
    run = config(save_result=True)
    run.assignment_1()
```

model.py:

```
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.neighbors import KNeighborsClassifier as KNN

class PCA_LDA_KNN():
```

```
    def __init__(self, n_components, n_neighbors):
        self.model_pca = PCA(n_components=n_components)
        self.model_lda = LDA()
        self.model_knn = KNN(n_neighbors=n_neighbors)

    def train(self, x_train, y_train):
        data_pca = self.model_pca.fit_transform(x_train)
        data_lda = self.model_lda.fit_transform(data_pca, y_train)
        self.model_knn.fit(data_lda, y_train)

    def acc(self, x_test, y_test):
        x_pca = self.model_pca.transform(x_test)
        x_lda = self.model_lda.transform(x_pca)
        return self.model_knn.score(x_lda, y_test)

    def pred(self):
        pass
```

utils.py:

```
import gzip, time, random, os, os.path as path
import pandas as pd, numpy as np
from matplotlib import pyplot as plt

def load_dataset(dataset_name):
    '''
    return numpy: x_train, y_train, x_test, y_test
    MNIST: http://yann.lecun.com/exdb/mnist/
    Letter Recognition: http://archive.ics.uci.edu/ml/datasets/Letter+Recognition
    '''
    PATH=path.abspath(path.join(path.dirname("__file__"),'dataset'))

    if dataset_name.title() == 'Letter Recognition':
        print('loading dataset: Letter Recognition')
        data_path = path.join(PATH, dataset_name.title(), 'letter-recognition.data')

        x_data = np.loadtxt(fname=data_path, dtype=float, delimiter=',', usecols=range(1,17))
        y_data = np.loadtxt(fname=data_path, dtype=str, delimiter=',', usecols=0)

        y_data = np.array([float(ord(y_data[i])-ord('A')) for i in range(len(y_data))])

        x_train = x_data[:-4000,:]
        y_train = y_data[:-4000]
        x_test = x_data[-16000:,:]
        y_test = y_data[-16000:]

        return x_train, y_train, x_test, y_test

    elif dataset_name.upper() == 'MNIST':
        print('loading dataset: MNIST')

        TRAIN_IMAGES = path.join(PATH, dataset_name.upper(), 'train-images-idx3-ubyte.gz')
        TRAIN_LABELS = path.join(PATH, dataset_name.upper(), 'train-labels-idx1-ubyte.gz')
        TEST_IMAGES = path.join(PATH, dataset_name.upper(), 't10k-images-idx3-ubyte.gz')
        TEST_LABELS = path.join(PATH, dataset_name.upper(), 't10k-labels-idx1-ubyte.gz')
```

```python
        x_train = load_MNIST_img(TRAIN_IMAGES)
        y_train = load_MNIST_label(TRAIN_LABELS)
        x_test = load_MNIST_img(TEST_IMAGES)
        y_test = load_MNIST_label(TEST_LABELS)

        return x_train, y_train, x_test, y_test

def load_MNIST_img(file_path):
    with gzip.open(file_path, 'rb') as bytestream:
        img_data=np.frombuffer(bytestream.read(), np.uint8, offset=16).astype(np.float32)
    #normalize : 将图像的像素归一化为 0.0~1.0
        img_data /= 255.0
    return img_data.reshape(-1, 784)

def load_MNIST_label(file_path):
    with gzip.open(file_path, 'rb') as bytestream:
        label_data=np.frombuffer(bytestream.read(), np.uint8, offset=8)
    return label_data

def randomcolor():
    '''生成随机颜色

    '''
    colorArr = ['1','2','3','4','5','6','7','8','9','A','B','C','D','E','F']
    color = ''
    for _ in range(6):
        color += colorArr[random.randint(0,14)]
    return "#" + color

def mkdir():
    '''创建以当前时间为文件名的文件夹，并返回该文件夹地址

    '''
    current_time = time.strftime("%Y-%m-%d %H%M%S", time.localtime())
    PATH = path.abspath(path.join(path.dirname("__file__"), path.pardir)) + '\\output'
    if not path.exists(PATH):
        print('creating folder...')
        os.mkdir(PATH)
        print(PATH)
        PATH += '\\' + current_time
        os.mkdir(PATH)
        print(PATH)
        print('Done.')
    else:
        PATH += '\\' + current_time
        if not path.exists(PATH):
            print('creating folder...')
            os.mkdir(PATH)
            print(PATH)
            print('Done.')
        else:
            print('the folder has been created.')
            print(PATH)
    return PATH
```

```
def output_assignment_1(data, dataset_name, n_components, n_neighbors, save_path):

    data_np = np.array(data)
    result_pd = pd.DataFrame(data_np, index=n_components, columns=n_neighbors)

    print(result_pd,'\n')

    title = dataset_name + ' Accuracy Curve'

    plt.title(title)
    plt.plot(n_components, data_np[:,0], c=randomcolor(), marker='.')
    plt.plot(n_components, data_np[:,1], c=randomcolor(), marker='*')
    plt.xlabel('PCA_dim')
    plt.ylabel('Acc')

    plt.xticks(n_components)
    plt.legend(['1-NN Acc','3-NN Acc'])
    plt.grid()

    for x, y in zip(n_components, data_np[:,0]):
        plt.text(x, y, '%.3f'%(y), fontdict={'fontsize': 9})
    for x, y in zip(n_components, data_np[:,1]):
        plt.text(x, y, '%.3f'%(y), fontdict={'fontsize': 9})

    if save_path != None:
        save_name_fig = title + '.png'
        save_name_excel = dataset_name + '_PCA_LDA_KNN.xlsx'
        result_pd.to_excel(path.join(save_path, save_name_excel))
        plt.savefig(path.join(save_path, save_name_fig))
    else:
        plt.show()
    plt.close()
```

附录 B

main.py:

```
from utils import mkdir, load_dataset, output_assignment_2
from model import ForwardNeuralNetwork

class config():
    def __init__(self, save_result):
        self.save_path = mkdir() if save_result else None
        self.dataset = ['Letter Recognition', 'MNIST']

    def config_2(self):
        self.num_labels = {'MNIST': 10, 'Letter Recognition': 26}
        self.feature_dim = {'MNIST': 784, 'Letter Recognition': 16}
        self.lr = [0.001, 0.01, 0.03, 0.05, 0.1]
        self.batch_size = [32, 64, 128, 256, 512]
        self.num_epochs = 150
        self.num_hidden = [1, 1.5, 2, 2.5, 3]

    def assignment_2(self):
        self.config_2()

        print('start assignment_2: Feed Forward Neural Network')

        for dataset_name in self.dataset:
            x_train, y_train, x_test, y_test = load_dataset(dataset_name)

            acc_hidden, loss_hidden = [], []
            print('num_hidden...')
            for num_hidden in self.num_hidden:
                model = ForwardNeuralNetwork(self.feature_dim[dataset_name], num_hidden, \
                                            self.num_labels[dataset_name])

                loss_list, acc_list = model.train(x_train, y_train, x_test, y_test, \
                                lr=1e-3, batch_size=128, num_epochs=self.num_epochs)
                acc_hidden.append(acc_list)
                loss_hidden.append(loss_list)

            loss_lr, acc_lr  = [], []
            print('lr...')
            for lr in self.lr:
                model = ForwardNeuralNetwork(self.feature_dim[dataset_name], 2, \
                                            self.num_labels[dataset_name])

                loss_list, acc_list = model.train(x_train, y_train, x_test, y_test, \
                                lr=lr, batch_size=128, num_epochs=self.num_epochs)
                acc_lr.append(acc_list)
                loss_lr.append(loss_list)

            loss_batch_size, acc_batch_size = [], []
            print('batch_size...')
            for batch_size in self.batch_size:
                model = ForwardNeuralNetwork(self.feature_dim[dataset_name], 2, \
                                            self.num_labels[dataset_name])
```

```
                loss_list, acc_list = model.train(x_train, y_train, x_test, y_test, \
                                lr=1e-3, batch_size=batch_size, num_epochs=self.num_epochs)
                acc_batch_size.append(acc_list)
                loss_batch_size.append(loss_list)

            output_assignment_2(acc_hidden, loss_hidden, acc_lr, loss_lr, \
                                acc_batch_size, loss_batch_size, 2, 1e-3, 128, dataset_name, \
                                self.feature_dim[dataset_name], self.num_labels[dataset_name], \
                                self.num_hidden, self.lr, self.batch_size, self.save_path)

if __name__ == '__main__':
    run = config(save_result=True)
    run.assignment_2()
```

model.py:

```
import time
import torch, numpy as np
from torch.nn import init
from torch.autograd import Variable
from torch.utils.data import DataLoader, TensorDataset

class ForwardNeuralNetwork():
    def __init__(self, input_dim, num_hidden, output_dim):
        self.device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

        self.net = torch.nn.Sequential(
            torch.nn.Linear(input_dim, int(num_hidden*input_dim)),
            torch.nn.ReLU(),
            torch.nn.Linear(int(num_hidden*input_dim), output_dim)
        ).to(self.device)

    def train(self, x_train, y_train, x_test, y_test, lr, batch_size, num_epochs):

        x_train = torch.Tensor(x_train).long()
        y_train = torch.Tensor(y_train).long()
        train_set = TensorDataset(x_train, y_train)
        train_loader = DataLoader(train_set, shuffle=True, batch_size=batch_size)


        optimizer = torch.optim.SGD(self.net.parameters(), lr=lr)

        criterion = torch.nn.CrossEntropyLoss().to(self.device)

        loss_list = []
        acc_list = []
        for epoch in range(num_epochs):
            count, loss_acc, start = 0, 0, time.time()
            # train
            for _, (feature, label) in enumerate(train_loader):
                X = Variable(feature).float().to(self.device)
                y_true = Variable(label).long().to(self.device)

                optimizer.zero_grad()
```

```
                y_hat = self.net(X)
                loss = criterion(y_hat, y_true.view(-1))

                loss.backward()
                optimizer.step()

                count += 1
                loss_acc += loss.item()

            # test
            test_acc = self.evaluate_accuracy(x_test, y_test, batch_size)

            # output
            loss_list.append(loss_acc/count)
            acc_list.append(test_acc)
            print('lr=%.3f, batch_size=%3d, epoch: [%d/%d]\ttrain loss: %.3f, test acc: %.3f,
                                                  elapse: %.2f sec;' \
                    %(lr, batch_size, epoch+1, num_epochs, loss_acc/count, test_acc, time.time()-
                                                          start))

        return loss_list, acc_list

    def evaluate_accuracy(self, x_test, y_test, batch_size):

        x_test = torch.Tensor(x_test)
        y_test = torch.Tensor(y_test)
        test_set = TensorDataset(x_test, y_test)
        test_loader = DataLoader(test_set, shuffle=True, batch_size=batch_size)

        count, acc_count = 0, 0
        for _, (X, y_true) in enumerate(test_loader):

            output = self.net(X.to(self.device))
            output = torch.nn.functional.softmax(output, dim=1)

            y_pred = output.argmax(dim=1)
            result = torch.eq(y_pred, y_true.to(self.device)).float()

            count += 1
            acc_count += torch.mean(result).item()
        return acc_count/count

    def pred(self):
        pass
```

utils.py:

```
import gzip, time, random, os, os.path as path
import numpy as np
from matplotlib import pyplot as plt

def load_dataset(dataset_name):
    '''
    return numpy: x_train, y_train, x_test, y_test
    MNIST: http://yann.lecun.com/exdb/mnist/
    Letter Recognition: http://archive.ics.uci.edu/ml/datasets/Letter+Recognition
```

```python
    '''
    PATH=path.abspath(path.join(path.dirname("__file__"),'dataset'))

    if dataset_name.title() == 'Letter Recognition':
        print('loading dataset: Letter Recognition')
        data_path = path.join(PATH, dataset_name.title(), 'letter-recognition.data')

        x_data = np.loadtxt(fname=data_path, dtype=float, delimiter=',', usecols=range(1,17))
        y_data = np.loadtxt(fname=data_path, dtype=str, delimiter=',', usecols=0)

        y_data = np.array([float(ord(y_data[i])-ord('A')) for i in range(len(y_data))])

        x_train = x_data[:-4000,:]
        y_train = y_data[:-4000]
        x_test = x_data[-16000:,:]
        y_test = y_data[-16000:]

        return x_train, y_train, x_test, y_test

    elif dataset_name.upper() == 'MNIST':
        print('loading dataset: MNIST')

        TRAIN_IMAGES = path.join(PATH, dataset_name.upper(), 'train-images-idx3-ubyte.gz')
        TRAIN_LABELS = path.join(PATH, dataset_name.upper(), 'train-labels-idx1-ubyte.gz')
        TEST_IMAGES = path.join(PATH, dataset_name.upper(), 't10k-images-idx3-ubyte.gz')
        TEST_LABELS = path.join(PATH, dataset_name.upper(), 't10k-labels-idx1-ubyte.gz')

        x_train = load_MNIST_img(TRAIN_IMAGES)
        y_train = load_MNIST_label(TRAIN_LABELS)
        x_test = load_MNIST_img(TEST_IMAGES)
        y_test = load_MNIST_label(TEST_LABELS)

        return x_train, y_train, x_test, y_test

def load_MNIST_img(file_path):
    with gzip.open(file_path, 'rb') as bytestream:
        img_data=np.frombuffer(bytestream.read(), np.uint8, offset=16).astype(np.float32)
    #normalize : 将图像的像素归一化为 0.0~1.0
        img_data /= 255.0
    return img_data.reshape(-1, 784)

def load_MNIST_label(file_path):
    with gzip.open(file_path, 'rb') as bytestream:
        label_data=np.frombuffer(bytestream.read(), np.uint8, offset=8)
    return label_data

def randomcolor():
    '''生成随机颜色

    '''
    colorArr = ['1','2','3','4','5','6','7','8','9','A','B','C','D','E','F']
    color = ''
    for _ in range(6):
        color += colorArr[random.randint(0,14)]
    return "#" + color
```

```python
def mkdir():
    '''创建以当前时间为文件名的文件夹，并返回该文件夹地址

    '''
    current_time = time.strftime("%Y-%m-%d %H%M%S", time.localtime())
    PATH = path.abspath(path.join(path.dirname("__file__"), path.pardir)) + '\\output'
    if not path.exists(PATH):
        print('creating folder...')
        os.mkdir(PATH)
        print(PATH)
        PATH += '\\' + current_time
        os.mkdir(PATH)
        print(PATH)
        print('Done.')
    else:
        PATH += '\\' + current_time
        if not path.exists(PATH):
            print('creating folder...')
            os.mkdir(PATH)
            print(PATH)
            print('Done.')
        else:
            print('the folder has been created.')
            print(PATH)
    return PATH

def output_assignment_2(acc_hidden, loss_hidden, acc_lr, loss_lr, \
                        acc_batch_size, loss_batch_size, \
                        default_num_hidden, default_lr, default_batch_size, \
                        dataset_name, feature_dim, \
                        num_labels_list, num_hidden_list, lr_list, batch_size_list, save_path):

    num_epochs = len(acc_hidden[0])

    string = str(num_hidden_list)
    #hidden dim acc
    title = dataset_name + ' Test Accuracy Curve(hidden dim)'

    plt.title(title)
    for i in range(len(num_hidden_list)):
        plt.plot(range(1, num_epochs+1), acc_hidden[i], c=randomcolor(), \
                    label='hidden dim: %d'%(num_hidden_list[i]*feature_dim))
    plt.xlabel('epoch')
    plt.ylabel('Acc')
    plt.grid()
    plt.legend()
    if save_path != None:
        save_name_fig = title + '.png'
        save_name_txt = title + '.txt'
        plt.savefig(path.join(save_path, save_name_fig))

        head = title + '\nnum_hidden: '+ string[1:-1] + '\n' + \
            'num_epochs=%d, batch_size=%d, lr=%.3f'%(num_epochs, default_batch_size, default_lr)
        np.savetxt(path.join(save_path, save_name_txt), np.array(acc_hidden), \
```

```
                fmt='%.5f', delimiter=',', header=head)
else:
    plt.show()
plt.close()

#hidden dim loss
title = dataset_name + ' Train Loss Curve(hidden dim)'
plt.title(title)
for i in range(len(num_hidden_list)):
    plt.plot(range(1, num_epochs+1), loss_hidden[i], c=randomcolor(), \
                label='hidden dim: %d'%(num_hidden_list[i]*feature_dim))
plt.xlabel('epoch')
plt.ylabel('Loss')
plt.grid()
plt.legend()
if save_path != None:
    save_name_fig = title + '.png'
    save_name_txt = title + '.txt'
    plt.savefig(path.join(save_path, save_name_fig))

    head = title + '\nnum_hidden: '+ string[1:-1] + '\n' + \
        'num_epochs=%d, batch_size=%d, lr=%.3f'%(num_epochs, default_batch_size, default_lr)
    np.savetxt(path.join(save_path, save_name_txt), np.array(acc_hidden), \
                fmt='%.5f', delimiter=',', header=head)
else:
    plt.show()
plt.close()

string = str(lr_list)
#lr acc
title = dataset_name + ' Test Accuracy Curve(lr)'
plt.title(title)
for i in range(len(lr_list)):
    plt.plot(range(1, num_epochs+1), acc_lr[i], c=randomcolor(), \
                label='lr: %.3f'%(lr_list[i]))
plt.xlabel('epoch')
plt.ylabel('Acc')
plt.grid()
plt.legend()
if save_path != None:
    save_name_fig = title + '.png'
    save_name_txt = title + '.txt'
    plt.savefig(path.join(save_path, save_name_fig))

    head = title + '\nlr: '+ string[1:-1] + '\n' + \
        'num_epochs=%d, batch_size=%d, hidden_dim=%d'%(num_epochs, default_batch_size, \
            int(default_num_hidden*feature_dim))
    np.savetxt(path.join(save_path, save_name_txt), np.array(acc_hidden), \
                fmt='%.5f', delimiter=',', header=head)
else:
    plt.show()
plt.close()

#lr loss
title = dataset_name + ' Train Loss Curve(lr)'
```

```
plt.title(title)
for i in range(len(lr_list)):
    plt.plot(range(1, num_epochs+1), loss_lr[i], c=randomcolor(), \
                label='lr: %.3f'%(lr_list[i]))
plt.xlabel('epoch')
plt.ylabel('Loss')
plt.grid()
plt.legend()
if save_path != None:
    save_name_fig = title + '.png'
    save_name_txt = title + '.txt'
    plt.savefig(path.join(save_path, save_name_fig))

    head = title + '\nlr: '+ string[1:-1] + '\n' + \
        'num_epochs=%d, batch_size=%d, hidden_dim=%d'%(num_epochs, default_batch_size, \
            int(default_num_hidden*feature_dim))

    np.savetxt(path.join(save_path, save_name_txt), np.array(acc_hidden), \
                fmt='%.5f', delimiter=',', header=head)
else:
    plt.show()
plt.close()

string = str(num_hidden_list)
#batch_size acc
title = dataset_name + ' Test Accuracy Curve(batch_size)'
plt.title(title)
for i in range(len(batch_size_list)):
    plt.plot(range(1, num_epochs+1), acc_batch_size[i], c=randomcolor(), \
                label='batch_size: %d'%(batch_size_list[i]))
plt.xlabel('epoch')
plt.ylabel('Acc')
plt.grid()
plt.legend()
if save_path != None:
    save_name_fig = title + '.png'
    save_name_txt = title + '.txt'
    plt.savefig(path.join(save_path, save_name_fig))

    head = title + '\nlr: '+ string[1:-1] + '\n' + \
        'num_epochs=%d, lr=%.3f, hidden_dim=%d'%(num_epochs,  default_lr, \
            int(default_num_hidden*feature_dim))
    np.savetxt(path.join(save_path, save_name_txt), np.array(acc_hidden), \
                fmt='%.5f', delimiter=',', header=head)
else:
    plt.show()
plt.close()

string = str(batch_size_list)
#batch_size loss
title = dataset_name + ' Train Loss Curve(batch_size)'
plt.title(title)
for i in range(len(batch_size_list)):
    plt.plot(range(1, num_epochs+1), loss_batch_size[i], c=randomcolor(), \
                label='batch_size: %d'%(batch_size_list[i]))
```

```python
    plt.xlabel('epoch')
    plt.ylabel('Loss')
    plt.grid()
    plt.legend()
    if save_path != None:
        save_name_fig = title + '.png'
        save_name_txt = title + '.txt'
        plt.savefig(path.join(save_path, save_name_fig))

        head = title + '\nlr: '+ string[1:-1] + '\n' + \
            'num_epochs=%d, lr=%.3f, hidden_dim=%d'%(num_epochs,  default_lr, \
                int(default_num_hidden*feature_dim))
        np.savetxt(path.join(save_path, save_name_txt), np.array(acc_hidden), \
                    fmt='%.5f', delimiter=',', header=head)
    else:
        plt.show()
    plt.close()
```