实验一 神经网络分类任务

1. 问题描述

1.1 概述

- 基于华为AI框架MindSpore，构建神经网络对CIFAR-10数据集中的测试集进行分类。

1.2 实验平台及数据说明

- MindSpore是一款华为自研的适用于端边云场景的新型开源深度学习训练/推理框架，旨在实现易开发、高效执行、全场景覆盖三大目标，提供了友好的设计和高效的执行，提升了数据科学家和算法工程师的开发体验，并进行了软硬件协同优化。
- CIFAR-10数据集是由Alex Krizhevsky，Vinod Nair以及Geoffrey Hinton收集的一个计算机图像数据集，涵盖10种不同且互斥的物体类别，总计60,000张32×32的RGB彩色图像。该数据集由训练数据集和测试数据集两部分组成，训练数据集包含50,000张样本图像及其类别标签，测试数据集包含10,000张样本图像及其类别标签，数字标签与类别名称间的对应关系见“**class_digits.txt**”文件，每张图像仅包含一种类别。附件中已提供下载好的数据集压缩包“**cifar-10-binary.tar.gz**”，解压后可根据自己的需求进行处理。获取CIFAR-10数据集的详细说明或下载其他格式，请参考：https://www.cs.toronto.edu/~kriz/cifar.html。

1.3 任务说明

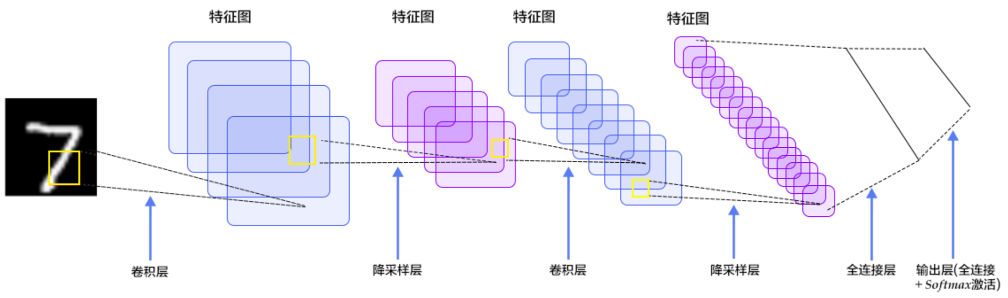
- 任务一：基于BP算法，在给定训练集上使用华为MindSpore框架自行设计并实现神经网络模型进行训练，随后对测试集进行分类，在实验报告中记录并分析所设计网络的分类准确率等性能指标。
- 任务二：记录神经网络在一个训练轮次（epoch）中训练损失值及分类准确率随训练步数（step的变化，绘制并保存为图表，可参考“sample_dynamics.png”，也可自行设计。
- 任务三：从测试集中随机选取若干图像，基于训练后的神经网络对该组图像的类别标签进行预测分类，将结果绘制并保存为图表，可参考“sample_predict.png”，也可自行设计。
- 在选择本实验并基本满足所有实验任务要求的前提下，实验总分额外加10分。

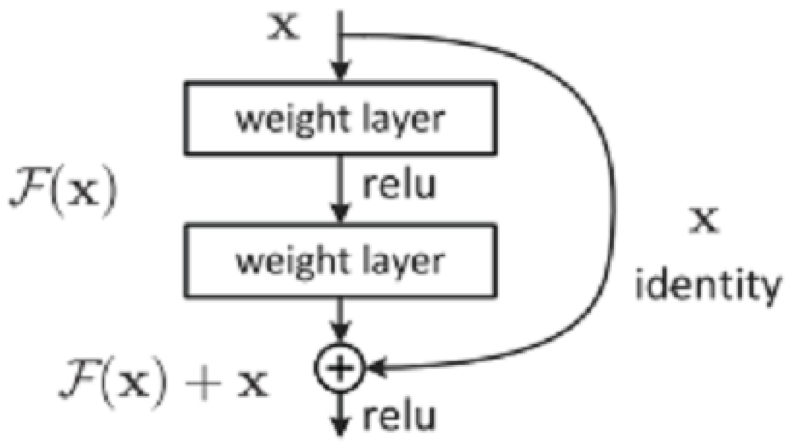
2. 实现步骤与流程

2.1 环境配置

- 在多次尝试后选用conda python3.7.5。虽然代码保留了GPU选项，但很难白嫖到x86的Ubuntu服务器所以最后还是用CPU训练。

2.2 代码编写

- 了解图片分类网络
    - LeNet
        - 输入的二维图像，先经过两次卷积层到池化层，再经过全连接层，最后使用softmax分类作为输出层
        - 

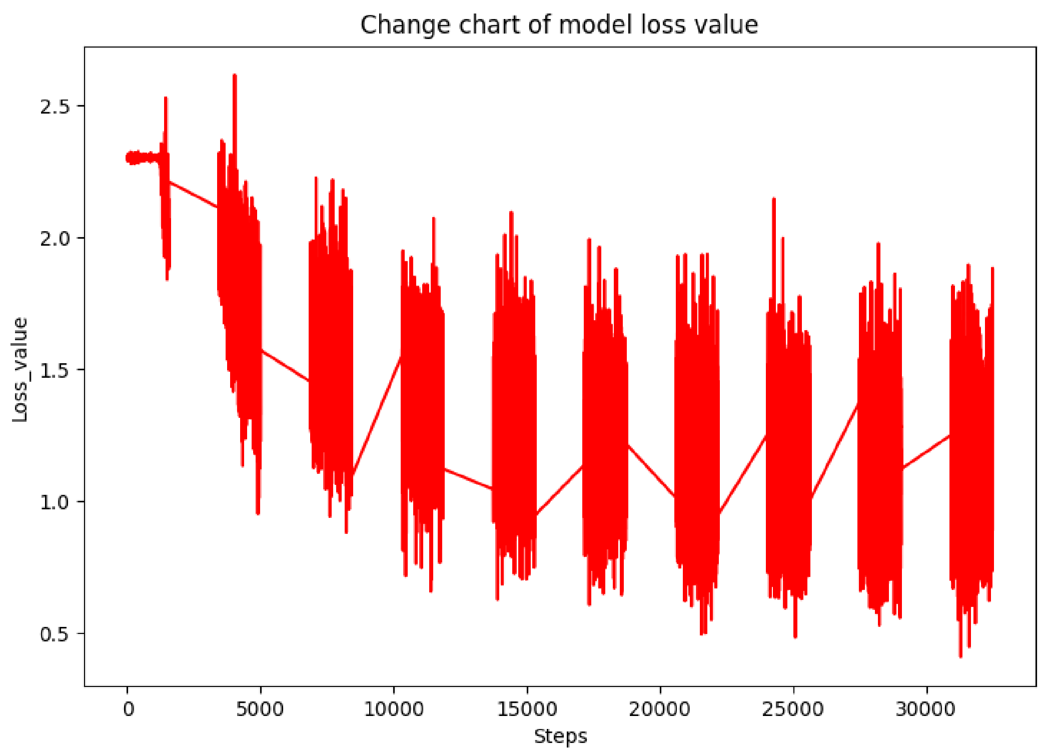
    - ResNet
        - 通过残差函数解决梯度弥散或爆炸的问题。以往神经网络由于梯度弥散层数不能过深，但通过在浅层网络基础上叠加$y = x$层可以使网络随深度加大不退化。
        - 
x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu
        - 由于过于难实现就嫁接了文档上的代码
- 通过阅读华为mindspore文档与实例编写代码，下面简述每块代码的作用，注释也已经标注没块代码的用途
    - 读入三个参数device_target、model和net分别对应设备，模式与网络
        - 其中设备不再赘述，模式包括训练和直接加载训练好的参数，为的是在只需要观看模型预测结果时跳过训练步骤
        - 网络包括ResNet和LeNet
    - 定义两个训练时用于传参的工具StepLossAccInfo和CrossEntropyLoss
    - 定义用于截取一定大小的数据集并强化数据特征的函数create_dataset
    - 定义同于训练和测试模型的函数train_net和test_net
    - 主函数

    - 定义网络net
    - 定义优化器net_opt和损失函数net_loss
    - 定义用于训练时保存模型的工具config_ck和ckpoint
    - 如果是训练模式则进行模型训练和测试
    - 加载训练完的网络参数并测试
    - 在测试集中选取32张图片用当前网络进行打标
    - 打印打标结果图片

3. 实现结果与分析

3.1 使用LeNet

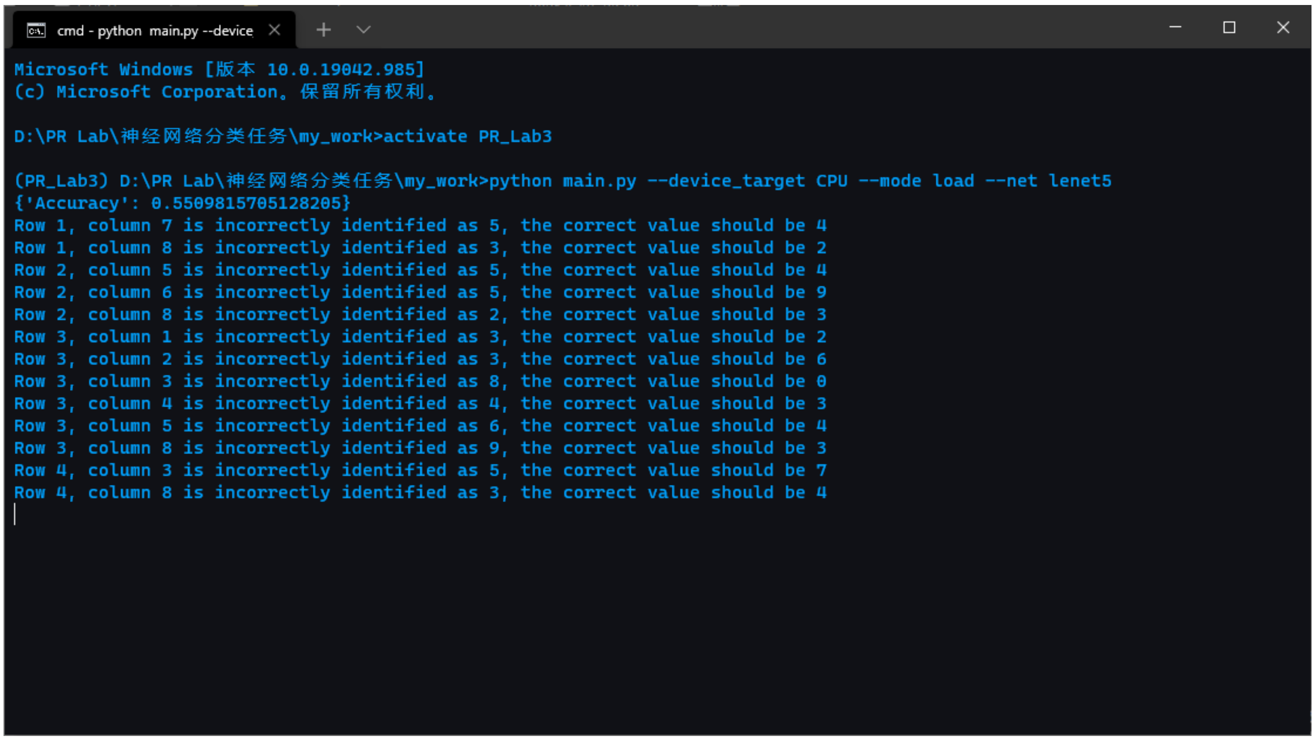
3.1.1训练过程

- 

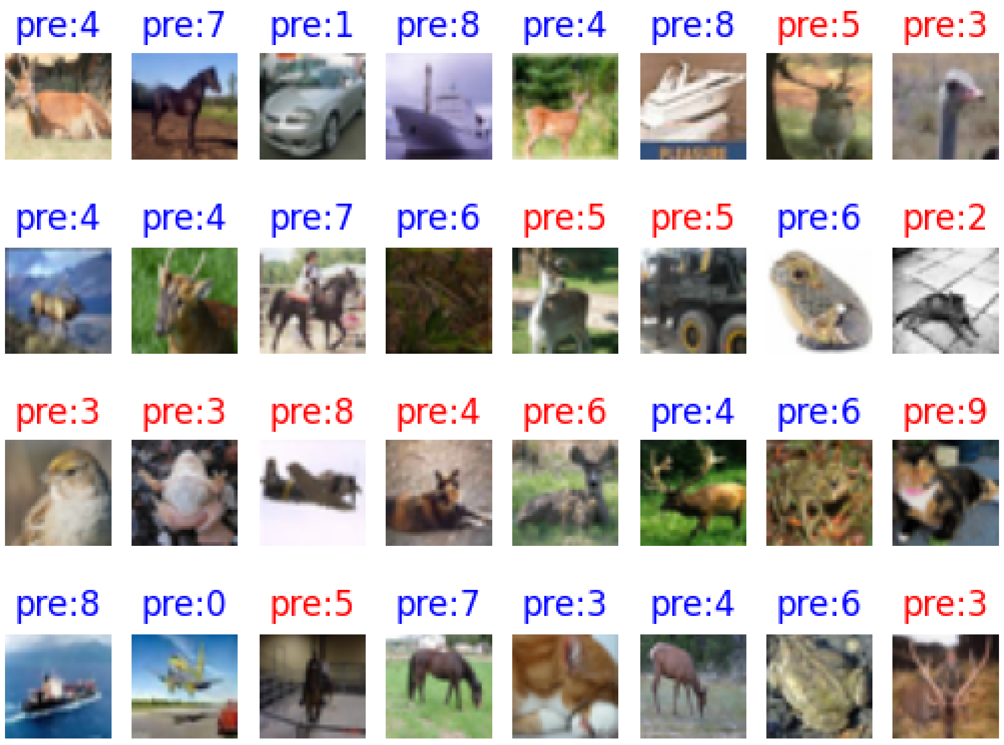
- 原本应该放一个epoch中loss的波动曲线，但一个epoch中收敛效果不明显，所以放10个epoch的step-loss曲线图，可见随着step增加，每个epoch的平均loss减少，可见模型在慢慢收敛。

3.1.2 训练结果

- 准确度在55%左右，图片打标结果有13个错的接近一半

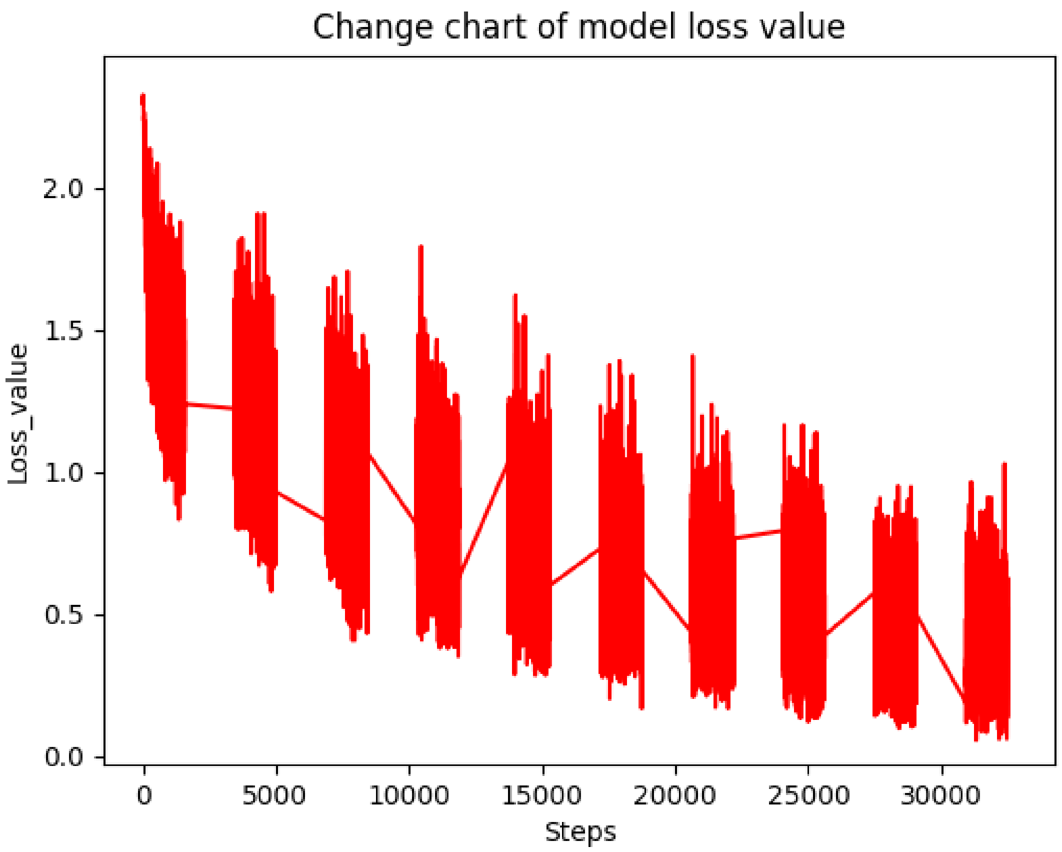


```
Microsoft Windows [版本 10.0.19042.985]
(c) Microsoft Corporation。保留所有权利。

D:\PR Lab\神经网络分类任务\my_work>activate PR_Lab3

(PR_Lab3) D:\PR Lab\神经网络分类任务\my_work>python main.py --device_target CPU --mode load --net lenet5
{'Accuracy': 0.5509815705128205}
Row 1, column 7 is incorrectly identified as 5, the correct value should be 4
Row 1, column 8 is incorrectly identified as 3, the correct value should be 2
Row 2, column 5 is incorrectly identified as 5, the correct value should be 4
Row 2, column 6 is incorrectly identified as 5, the correct value should be 9
Row 2, column 8 is incorrectly identified as 2, the correct value should be 3
Row 3, column 1 is incorrectly identified as 3, the correct value should be 2
Row 3, column 2 is incorrectly identified as 3, the correct value should be 6
Row 3, column 3 is incorrectly identified as 8, the correct value should be 0
Row 3, column 4 is incorrectly identified as 4, the correct value should be 3
Row 3, column 5 is incorrectly identified as 6, the correct value should be 4
Row 3, column 8 is incorrectly identified as 9, the correct value should be 3
Row 4, column 3 is incorrectly identified as 5, the correct value should be 7
Row 4, column 8 is incorrectly identified as 3, the correct value should be 4
```



3.2 使用ResNet

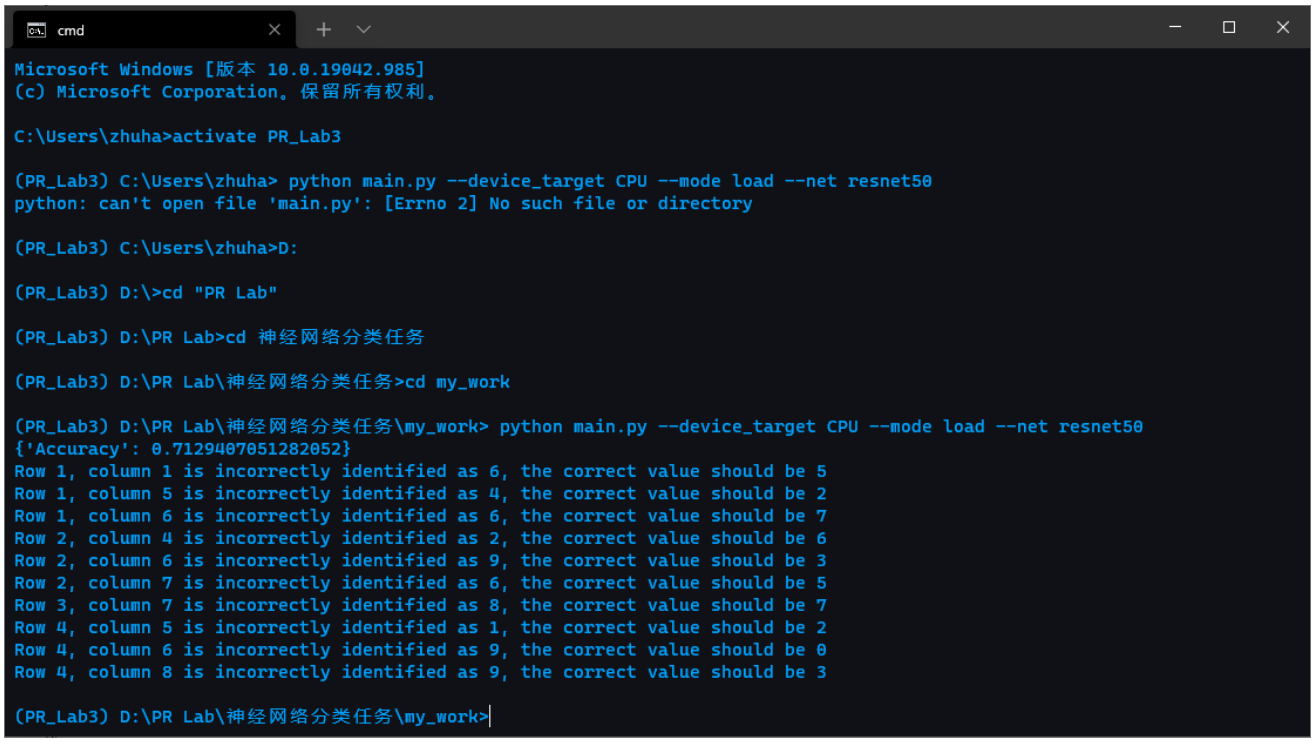
3.2.1 训练过程

- 

- 相比LeNet，ResNet收敛过程中loss波动更小，初始收敛更明显，并且可以看出如果继续训练可以得出更好的结果。但受限于设备与时间，只训练了10个epoch。

3.2.2 训练结果

- 准确度在70%左右，图片打标结果有10个错大致30%的错误率



```
Microsoft Windows [版本 10.0.19042.985]
(c) Microsoft Corporation。保留所有权利。

C:\Users\zhuha>activate PR_Lab3

(PR_Lab3) C:\Users\zhuha> python main.py --device_target CPU --mode load --net resnet50
python: can't open file 'main.py': [Errno 2] No such file or directory

(PR_Lab3) C:\Users\zhuha>D:

(PR_Lab3) D:\>cd "PR Lab"

(PR_Lab3) D:\PR Lab>cd 神经网络分类任务

(PR_Lab3) D:\PR Lab\神经网络分类任务>cd my_work

(PR_Lab3) D:\PR Lab\神经网络分类任务\my_work> python main.py --device_target CPU --mode load --net resnet50
{'Accuracy': 0.7129407051282052}
Row 1, column 1 is incorrectly identified as 6, the correct value should be 5
Row 1, column 5 is incorrectly identified as 4, the correct value should be 2
Row 1, column 6 is incorrectly identified as 6, the correct value should be 7
Row 2, column 4 is incorrectly identified as 2, the correct value should be 6
Row 2, column 6 is incorrectly identified as 9, the correct value should be 3
Row 2, column 7 is incorrectly identified as 6, the correct value should be 5
Row 3, column 7 is incorrectly identified as 8, the correct value should be 7
Row 4, column 5 is incorrectly identified as 1, the correct value should be 2
Row 4, column 6 is incorrectly identified as 9, the correct value should be 0
Row 4, column 8 is incorrectly identified as 9, the correct value should be 3

(PR_Lab3) D:\PR Lab\神经网络分类任务\my_work>
```

可见ResNet的分类准确度远高于LeNet，但训练时间也更久，迭代速度更慢

4. 代码附录

lenet.py

```
'''
lenet
'''
import mindspore.nn as nn
from mindspore.common.initializer import Normal

class LeNet5(nn.Cell):
    def __init__(self, num_class=10, num_channel=3):
        super(LeNet5, self).__init__()
        self.conv1 = nn.Conv2d(num_channel, 6, 5, pad_mode='valid')
        self.conv2 = nn.Conv2d(6, 16, 5, pad_mode='valid')
        self.fc1 = nn.Dense(16 * 5 * 5, 120, weight_init=Normal(0.02))
        self.fc2 = nn.Dense(120, 84, weight_init=Normal(0.02))
        self.fc3 = nn.Dense(84, num_class, weight_init=Normal(0.02))
        self.relu = nn.ReLU()
        self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
        self.flatten = nn.Flatten()

    def construct(self, x):
        x = self.conv1(x)
        x = self.relu(x)
        x = self.max_pool2d(x)
        x = self.conv2(x)
        x = self.relu(x)
        x = self.max_pool2d(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)
        return x


def lenet5():
    """create lenet5"""
    return LeNet5()
```

resnet.py

```
'''
resnet
'''
import numpy as np
import mindspore.nn as nn
from mindspore import Tensor
import mindspore.ops as ops

def weight_variable_0(shape):
    """weight_variable_0"""
    zeros = np.zeros(shape).astype(np.float32)
    return Tensor(zeros)


def weight_variable_1(shape):
    """weight_variable_1"""
    ones = np.ones(shape).astype(np.float32)
    return Tensor(ones)


def conv3x3(in_channels, out_channels, stride=1, padding=0):
    """3x3 convolution """
    return nn.Conv2d(in_channels, out_channels,
                     kernel_size=3, stride=stride, padding=padding,
weight_init='XavierUniform',
                     has_bias=False, pad_mode="same")
```

```
def conv1x1(in_channels, out_channels, stride=1, padding=0):
    """1x1 convolution"""
    return nn.Conv2d(in_channels, out_channels,
                     kernel_size=1, stride=stride, padding=padding,
weight_init='XavierUniform',
                     has_bias=False, pad_mode="same")


def conv7x7(in_channels, out_channels, stride=1, padding=0):
    """1x1 convolution"""
    return nn.Conv2d(in_channels, out_channels,
                     kernel_size=7, stride=stride, padding=padding,
weight_init='XavierUniform',
                     has_bias=False, pad_mode="same")


def bn_with_initialize(out_channels):
    """bn_with_initialize"""
    shape = (out_channels)
    mean = weight_variable_0(shape)
    var = weight_variable_1(shape)
    beta = weight_variable_0(shape)
    bn = nn.BatchNorm2d(out_channels, momentum=0.99, eps=0.00001,
gamma_init='Uniform',
                        beta_init=beta, moving_mean_init=mean,
moving_var_init=var)
    return bn


def bn_with_initialize_last(out_channels):
    """bn_with_initialize_last"""
    shape = (out_channels)
    mean = weight_variable_0(shape)
    var = weight_variable_1(shape)
    beta = weight_variable_0(shape)
    bn = nn.BatchNorm2d(out_channels, momentum=0.99, eps=0.00001,
gamma_init='Uniform',
                        beta_init=beta, moving_mean_init=mean,
moving_var_init=var)
    return bn


def fc_with_initialize(input_channels, out_channels):
    """fc_with_initialize"""
    return nn.Dense(input_channels, out_channels,
weight_init='XavierUniform', bias_init='Uniform')


class ResidualBlock(nn.Cell):
    """ResidualBlock"""
    expansion = 4

    def __init__(self,
                 in_channels,
                 out_channels,
                 stride=1):
        """init block"""
        super(ResidualBlock, self).__init__()

        out_chls = out_channels // self.expansion
        self.conv1 = conv1x1(in_channels, out_chls, stride=stride,
padding=0)
        self.bn1 = bn_with_initialize(out_chls)

        self.conv2 = conv3x3(out_chls, out_chls, stride=1, padding=0)
        self.bn2 = bn_with_initialize(out_chls)

        self.conv3 = conv1x1(out_chls, out_channels, stride=1,
padding=0)
        self.bn3 = bn_with_initialize_last(out_channels)

        self.relu = ops.ReLU()
        self.add = ops.Add()

    def construct(self, x):
        """construct"""
        identity = x

        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)
        out = self.relu(out)

        out = self.conv3(out)
        out = self.bn3(out)

        out = self.add(out, identity)
        out = self.relu(out)

        return out


class ResidualBlockWithDown(nn.Cell):
    """ResidualBlockWithDown"""
    expansion = 4
    def __init__(self,
                 in_channels,
                 out_channels,
                 stride=1,
                 down_sample=False):
        """init block with down"""
        super(ResidualBlockWithDown, self).__init__()

        out_chls = out_channels // self.expansion
        self.conv1 = conv1x1(in_channels, out_chls, stride=stride,
padding=0)
        self.bn1 = bn_with_initialize(out_chls)

        self.conv2 = conv3x3(out_chls, out_chls, stride=1, padding=0)
        self.bn2 = bn_with_initialize(out_chls)

        self.conv3 = conv1x1(out_chls, out_channels, stride=1,
padding=0)
        self.bn3 = bn_with_initialize_last(out_channels)

        self.relu = ops.ReLU()
        self.down_sample = down_sample

        self.conv_down_sample = conv1x1(in_channels, out_channels,
stride=stride, padding=0)
        self.bn_down_sample = bn_with_initialize(out_channels)
        self.add = ops.Add()

    def construct(self, x):
        """construct"""
        identity = x

        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)
        out = self.relu(out)

        out = self.conv3(out)
        out = self.bn3(out)

        identity = self.conv_down_sample(identity)
        identity = self.bn_down_sample(identity)

        out = self.add(out, identity)
        out = self.relu(out)
        return out


class MakeLayer0(nn.Cell):
    """MakeLayer0"""

    def __init__(self, block, in_channels, out_channels, stride):
        """init"""
        super(MakeLayer0, self).__init__()
        self.a = ResidualBlockWithDown(in_channels, out_channels,
stride=1, down_sample=True)
        self.b = block(out_channels, out_channels, stride=stride)
        self.c = block(out_channels, out_channels, stride=1)

    def construct(self, x):
        """construct"""
        x = self.a(x)
        x = self.b(x)
        x = self.c(x)

        return x


class MakeLayer1(nn.Cell):
    """MakeLayer1"""

    def __init__(self, block, in_channels, out_channels, stride):
        """init"""
        super(MakeLayer1, self).__init__()
        self.a = ResidualBlockWithDown(in_channels, out_channels,
stride=stride, down_sample=True)
        self.b = block(out_channels, out_channels, stride=1)
        self.c = block(out_channels, out_channels, stride=1)
        self.d = block(out_channels, out_channels, stride=1)

    def construct(self, x):
        """construct"""
        x = self.a(x)
        x = self.b(x)
        x = self.c(x)
        x = self.d(x)

        return x


class MakeLayer2(nn.Cell):
    """MakeLayer2"""
    def __init__(self, block, in_channels, out_channels, stride):
        """init"""
        super(MakeLayer2, self).__init__()
        self.a = ResidualBlockWithDown(in_channels, out_channels,
stride=stride, down_sample=True)
        self.b = block(out_channels, out_channels, stride=1)
        self.c = block(out_channels, out_channels, stride=1)
        self.d = block(out_channels, out_channels, stride=1)
        self.e = block(out_channels, out_channels, stride=1)
        self.f = block(out_channels, out_channels, stride=1)

    def construct(self, x):
        """construct"""
        x = self.a(x)
        x = self.b(x)
        x = self.c(x)
        x = self.d(x)
        x = self.e(x)
        x = self.f(x)

        return x


class MakeLayer3(nn.Cell):
    """MakeLayer3"""

    def __init__(self, block, in_channels, out_channels, stride):
        """init"""
        super(MakeLayer3, self).__init__()
        self.a = ResidualBlockWithDown(in_channels, out_channels,
stride=stride, down_sample=True)
        self.b = block(out_channels, out_channels, stride=1)
        self.c = block(out_channels, out_channels, stride=1)

    def construct(self, x):
        """construct"""
        x = self.a(x)
        x = self.b(x)
        x = self.c(x)

        return x


class ResNet(nn.Cell):
    """ResNet"""

    def __init__(self, block, num_classes=100, batch_size=32):
        """init"""
        super(ResNet, self).__init__()
        self.batch_size = batch_size
        self.num_classes = num_classes

        self.conv1 = conv7x7(3, 64, stride=2, padding=0)

        self.bn1 = bn_with_initialize(64)
        self.relu = ops.ReLU()
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2,
pad_mode="same")

        self.layer1 = MakeLayer0(block, in_channels=64,
out_channels=256, stride=1)
        self.layer2 = MakeLayer1(block, in_channels=256,
out_channels=512, stride=2)
        self.layer3 = MakeLayer2(block, in_channels=512,
out_channels=1024, stride=2)
        self.layer4 = MakeLayer3(block, in_channels=1024,
out_channels=2048, stride=2)

        self.pool = ops.ReduceMean(keep_dims=True)
        self.squeeze = ops.Squeeze(axis=(2, 3))
        self.fc = fc_with_initialize(512 * block.expansion, num_classes)

    def construct(self, x):
        """construct"""
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.maxpool(x)

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)

        x = self.pool(x, (2, 3))
        x = self.squeeze(x)
        x = self.fc(x)
        return x


def resnet50(batch_size, num_classes):
    """create resnet50"""
    return ResNet(ResidualBlock, num_classes, batch_size)
```

main.py

```
'''
env: python3.7.5
```

```
requirements: mindspore, numpy, matplotlib

usage: python main.py --device_target CPU/GPU/Ascend --mode load/train -
-net resnet50/lenet5
--device_target  according to the mindspore you use
--mode  use load when you want to skip the train step
--net  choose from the two nets.

Resnet takes more time to train while its accuracy is more than 70%
Accuracy of the Lenet fluctuate between 50% and 60% while it takes only
a few minutes to train
'''

import os
import argparse
import matplotlib.pyplot as plt
import numpy as np
import mindspore.dataset as ds
import mindspore.dataset.transforms.c_transforms as C
import mindspore.dataset.vision.c_transforms as CV
import mindspore.nn as nn
from mindspore import dtype as mstype
from mindspore import context, Tensor, Model, load_checkpoint,
load_param_into_net
from mindspore.train.callback import Callback, ModelCheckpoint,
CheckpointConfig, LossMonitor
from mindspore.dataset.vision import Inter
from mindspore.common.initializer import Normal
from mindspore.nn import Accuracy
import mindspore.ops as ops
from resnet50 import resnet50
from lenet5 import lenet5


# some changeable parameters
model_path = "./model"  # where to save model
mnist_path ="./cifar-10-binary/cifar-10-batches-bin"  # where the
dataset is


# some input arguments
parser = argparse.ArgumentParser(description='PR_Lab3')
parser.add_argument('--device_target', type=str, default="CPU", choices=
['Ascend', 'GPU', 'CPU'])
parser.add_argument('--mode', type=str, default="load", choices=
['load','train'])
parser.add_argument('--net', type=str, default="resnet50", choices=
['resnet50','lenet5'])
```

```
args = parser.parse_args()
context.set_context(mode=context.GRAPH_MODE,
device_target=args.device_target)
mode = args.mode
net_arg = args.net

#definition of callbacks
class StepLossAccInfo(Callback):
    def __init__(self, model, eval_dataset, steps_loss, steps_eval):
        self.model = model
        self.eval_dataset = eval_dataset
        self.steps_loss = steps_loss
        self.steps_eval = steps_eval

    def step_end(self, run_context):
        cb_params = run_context.original_args()
        cur_epoch = cb_params.cur_epoch_num
        cur_step = (cur_epoch-1)*1875 + cb_params.cur_step_num
        self.steps_loss["loss_value"].append(str(cb_params.net_outputs))
        self.steps_loss["step"].append(str(cur_step))
        if cur_step % 125 == 0:
            acc = self.model.eval(self.eval_dataset,
dataset_sink_mode=False)
            self.steps_eval["step"].append(cur_step)
            self.steps_eval["acc"].append(acc["Accuracy"])

class CrossEntropyLoss(nn.Cell):
    def __init__(self):
        super(CrossEntropyLoss, self).__init__()
        self.cross_entropy = ops.SoftmaxCrossEntropyWithLogits()
        self.mean = ops.ReduceMean()
        self.one_hot = ops.OneHot()
        self.one = Tensor(1.0, mstype.float32)
        self.zero = Tensor(0.0, mstype.float32)

    def construct(self, logits, label):
        label = self.one_hot(label, ops.shape(logits)[1], self.one,
self.zero)
        loss_func = self.cross_entropy(logits, label)[0]
        loss_func = self.mean(loss_func, (-1,))
        return loss_func


# create a dataset of optional size
def create_dataset(sample_num, data_path, batch_size=32, repeat_size=1,
                   num_parallel_workers=1):
    # load dataset
    # mnist_ds = ds.MnistDataset(data_path)
    mnist_ds = ds.Cifar10Dataset(data_path, num_samples=sample_num,
shuffle=True)
    resize_height, resize_width = 32, 32
    rescale = 1.0 / 255.0
    shift = 0.0
    rescale_nml = 1 / 0.3081
    shift_nml = -1 * 0.1307 / 0.3081

    # define resizers
    resize_op = CV.Resize((resize_height, resize_width),
interpolation=Inter.LINEAR)
    rescale_nml_op = CV.Rescale(rescale_nml, shift_nml)
    rescale_op = CV.Rescale(rescale, shift)
    hwc2chw_op = CV.HWC2CHW()
    type_cast_op = C.TypeCast(mstype.int32)

    # define maps and enhance dataset
    mnist_ds = mnist_ds.map(operations=type_cast_op,
input_columns="label", num_parallel_workers=num_parallel_workers)
    mnist_ds = mnist_ds.map(operations=resize_op, input_columns="image",
num_parallel_workers=num_parallel_workers)
    mnist_ds = mnist_ds.map(operations=rescale_op,
input_columns="image", num_parallel_workers=num_parallel_workers)
    mnist_ds = mnist_ds.map(operations=rescale_nml_op,
input_columns="image", num_parallel_workers=num_parallel_workers)
    mnist_ds = mnist_ds.map(operations=hwc2chw_op,
input_columns="image", num_parallel_workers=num_parallel_workers)

    # shuffle and batch
    buffer_size = 10000
    mnist_ds = mnist_ds.shuffle(buffer_size=buffer_size)
    mnist_ds = mnist_ds.batch(batch_size, drop_remainder=True)

    return mnist_ds


# the trainning and testing functions
steps_loss = {"step": [], "loss_value": []}
steps_eval = {"step": [], "acc": []}
def train_net(args, model, epoch_size, data_path, repeat_size,
ckpoint_cb, sink_mode):
    # load dataset
    ds_train = create_dataset(50000, os.path.join(data_path, "train"),
32, repeat_size)
    ds_eval = create_dataset(10000, os.path.join(data_path, "test"))

    # save the network model and parameters for subsequence fine-tuning
    config_ck = CheckpointConfig(save_checkpoint_steps=375,
keep_checkpoint_max=16)
```

```
    # group layers into an object with training and evaluation features
    ckpoint_cb = ModelCheckpoint(prefix="checkpoint_"+str(net_arg),
directory=model_path, config=config_ck)

    # collect the steps,loss and accuracy information
    step_loss_acc_info = StepLossAccInfo(model ,ds_eval, steps_loss,
steps_eval)

    model.train(epoch_size, ds_train, callbacks=[ckpoint_cb,
LossMonitor(per_print_times=1), step_loss_acc_info],
dataset_sink_mode=False)

def test_net(network, model, data_path):
    ds_eval = create_dataset(10000, os.path.join(data_path, "test"))
    acc = model.eval(ds_eval, dataset_sink_mode=False)
    print("{}".format(acc))


if __name__ == "__main__":
    # definition of the net
    net = resnet50(batch_size=32, num_classes=10) if net_arg ==
'resnet50' else lenet5()
    # definition of loss function
    net_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True,
reduction='mean')
    # definition of optimizer
    net_opt = nn.Momentum(net.trainable_params(), learning_rate=0.01,
momentum=0.9)
    # model savers
    config_ck = CheckpointConfig(save_checkpoint_steps=1875,
keep_checkpoint_max=10)
    ckpoint = ModelCheckpoint(prefix="checkpoint_lenet",
config=config_ck)

    if mode == 'train':
        # train and evaluate the model in train mode
        train_epoch = 10
        dataset_size = 1
        model = Model(net, net_loss, net_opt, metrics={"Accuracy":
Accuracy()})
        train_net(args, model, train_epoch, mnist_path, dataset_size,
ckpoint, False)
        test_net(net, model, mnist_path)

        # draw the step_loss chart
        steps = steps_loss["step"]
        loss_value = steps_loss["loss_value"]
        steps = list(map(int, steps))
        loss_value = list(map(float, loss_value))
        plt.plot(steps, loss_value, color="red")
        plt.xlabel("Steps")
        plt.ylabel("Loss_value")
        plt.title("Change chart of model loss value")
        plt.show()

    # load the model and evaluate, or you can just directly evaluate 
after training. The model loading step exists when you want to skip the 
trainning process and see the picture below.
    if net_arg == 'resnet50':
        load_checkpoint("checkpoint_resnet50_3-10_1562.ckpt", net=net)
    else:
        load_checkpoint("checkpoint_lenet5-10_1562.ckpt", net=net)
    net_loss = CrossEntropyLoss()
    model = Model(net, net_loss, metrics={"Accuracy": Accuracy()})
    ds_eval = create_dataset(10000, os.path.join(mnist_path, "test"))
    acc = model.eval(ds_eval, dataset_sink_mode=False)
    print("{}".format(acc))

    # randomly choose 32 pictures from the testset and predict them. The 
blue ones are correctly classified while the red means incorrect.
    ds = create_dataset(32, os.path.join(mnist_path, "test"))
    ds_test = ds.create_dict_iterator()
    data = next(ds_test)

    images = data["image"].asnumpy()
    labels = data["label"].asnumpy()

    output = model.predict(Tensor(data['image']))
    pred = np.argmax(output.asnumpy(), axis=1)

    images = np.add(images,1 * 0.1307 / 0.3081)
    images = np.multiply(images, 0.3081)

    index = 1
    for i in range(len(labels)):
        plt.subplot(4, 8, i+1)
        color = 'blue' if pred[i] == labels[i] else 'red'
        plt.title("pre:{}".format(pred[i]), color=color)
        img = np.squeeze(images[i]).transpose((1,2,0))
        plt.imshow(img)
        plt.axis("off")
        if color == 'red':
            index = 0
            print("Row {}, column {} is incorrectly identified as {}, 
the correct value should be {}".format(int(i/8)+1, i%8+1, pred[i], 
labels[i]))
    if index:
        print("All the figures in this group are predicted correctly!")
    plt.show()
```

实验二 贝叶斯分类任务

1. 问题描述

1.1 实验概述

- 利用贝叶斯分类算法对 wine 数据集中的测试集进行分类。

1.2 数据说明

- wine 葡萄酒数据集是 UCI 上的公开数据集。数据集包含由三种不同葡萄酿造的葡萄酒，通过化学分析确定了葡萄酒中含有的 13 种成分的含量。数据集的相关信息如表 1所示：

样例数量	特征维度	特征类型	类别数量
178	13	数值	3

- 数据集已被划分为训练集和测试集，分别存储于data文件夹中的train_data.csv和test_data.csv。其中，训练集包含 120 个样例，测试集包含 58 个样例，每个样例包含各个维度的特征值及样例标签（标签为 1、2 或 3），假定各维度的特征属性之间条件独立。

1.3 任务说明

- 基于贝叶斯分类原理，实现一个**贝叶斯分类器**。在**训练集**中进行训练，尽可能提高模型准确率，并在**测试集**上进行测试。在朴素贝叶斯分类模型中，当属性是连续型时，有两种方法可以计算属性的类条件概率：第一种方法是把一个连续的属性离散化，然后用相应的离散区间替换连续属性值，之后用频率去表示类条件概率，但这种方法不好控制离散区间划分的粒度；第二种方法是假设连续变量从某种概率分布，然后使用训练数据估计分布的参数，例如可以使用高斯分布来表示连续属性的类条件概率分布，通过高斯分布估计出类条件概率。
- 本实验规定采用**高斯分布**估计类条件概率。其中，均值和方差分别用训练集的**样本均值**和**样本方差**估计。
- 实验报告要求对**贝叶斯分类模型的过程**进行推导，并计算各个属性各个类别的**类条件密度**（高斯分布），同时，给出测试集的**预测准确率**。
- 测试集预测结果文件需要包含每个测试样例的**预测类别**及**分属于三个类别的概率值**。

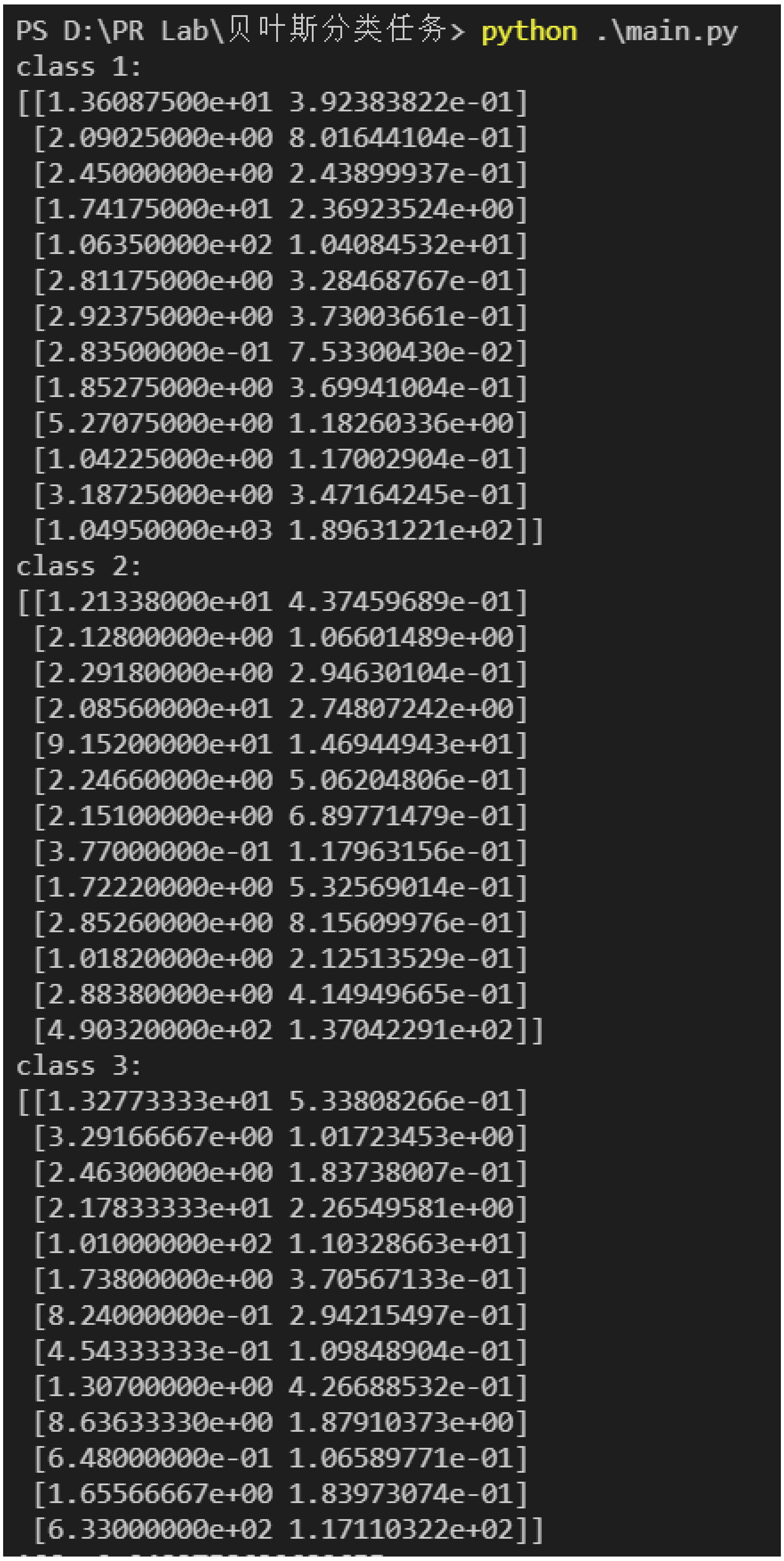
2. 实现步骤与流程

2.1 分析与假设

- 假设数据分布符合高斯函数，所以使用高斯函数拟合。运用极大似然可知，高斯分布的均值与方差为样本均值和方差
- 对于某个特征向量$\vec{x}$我们需要计算的是:$P(w_i|\vec{x}) = \frac{p(\vec{x}|w_i)P(w_i)}{p(\vec{x})}$
- 由于i不影响$p(\vec{x})$大小，所以对于一个x_i只要对比分子$p(\vec{x}|w_i)P(w_i)$的大小即可获得分类
    - 由于我们假设每个特征独立，则可以得到$p(\vec{x}|w_i) = \prod_{i=1}^{13} p(x_i|w_i)$。其中13为特征维度，并且由于假设数据服从高斯分布，所以可得公式 $p(x_i|w_i) = \frac{1}{\sqrt{2\pi}\delta_i} e^{-\frac{(x-\mu_i)^2}{2\delta_i^2}}$
    - $P(w_i)$相对更加好得到，此处可以直接得到$P(w_i) = \frac{|w_i|}{|D|}$，D为训练集。
- 由此对于验证集上每一个$\vec{x}$，都可以得到三个类的后验概率并比对大小。选取最大的后验概率作为分类选项

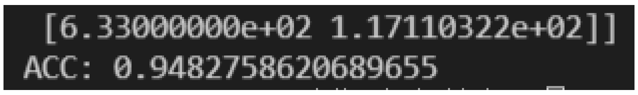
3. 实现结果与分析

- 每个类内对于13个特征的高斯分布参数，每一行前一个数字为均值，后一个为方差
-



```
PS D:\PR Lab\贝叶斯分类任务> python .\main.py
class 1:
[[1.36087500e+01 3.92383822e-01]
 [2.09025000e+00 8.01644104e-01]
 [2.45000000e+00 2.43899937e-01]
 [1.74175000e+01 2.36923524e+00]
 [1.06350000e+02 1.04084532e+01]
 [2.81175000e+00 3.28468767e-01]
 [2.92375000e+00 3.73003661e-01]
 [2.83500000e-01 7.53300430e-02]
 [1.85275000e+00 3.69941004e-01]
 [5.27075000e+00 1.18260336e+00]
 [1.04225000e+00 1.17002904e-01]
 [3.18725000e+00 3.47164245e-01]
 [1.04950000e+03 1.89631221e+02]]
class 2:
[[1.21338000e+01 4.37459689e-01]
 [2.12800000e+00 1.06601489e+00]
 [2.29180000e+00 2.94630104e-01]
 [2.08560000e+01 2.74807242e+00]
 [9.15200000e+01 1.46944943e+01]
 [2.24660000e+00 5.06204806e-01]
 [2.15100000e+00 6.89771479e-01]
 [3.77000000e-01 1.17963156e-01]
 [1.72220000e+00 5.32569014e-01]
 [2.85260000e+00 8.15609976e-01]
 [1.01820000e+00 2.12513529e-01]
 [2.88380000e+00 4.14949665e-01]
 [4.90320000e+02 1.37042291e+02]]
class 3:
[[1.32773333e+01 5.33808266e-01]
 [3.29166667e+00 1.01723453e+00]
 [2.46300000e+00 1.83738007e-01]
 [2.17833333e+01 2.26549581e+00]
 [1.01000000e+02 1.10328663e+01]
 [1.73800000e+00 3.70567133e-01]
 [8.24000000e-01 2.94215497e-01]
 [4.54333333e-01 1.09848904e-01]
 [1.30700000e+00 4.26688532e-01]
 [8.63633330e+00 1.87910373e+00]
 [6.48000000e-01 1.06589771e-01]
 [1.65566667e+00 1.83973074e-01]
 [6.33000000e+02 1.17110322e+02]]
```

- 从准确率来看贝叶斯确实在这一数据集上可以起到良好的预测效果



```
  [6.33000000e+02 1.17110322e+02]]
ACC: 0.9482758620689655
```

- 预测结果可见贝叶斯文件夹下的CSV文件

4. 代码附录

main.py

```
import numpy as np
import csv
import math

# load dataset
p = r'./data/train_data.csv'
with open(p,encoding = 'utf-8') as f:
    dataset = np.loadtxt(f,delimiter = ",")
k = r'./data/test_data.csv'
with open(k,encoding = 'utf-8') as f:
    testset = np.loadtxt(f,delimiter = ",")

# calculate mean and std
mean_train = [np.mean(dataset[np.where(dataset[:,0]==i)], axis=0)[1:]
for i in range(1,4)]
std_train = [np.std(dataset[np.where(dataset[:,0]==i)], axis=0, ddof=1)
[1:] for i in range(1,4)]
print("class 1:")
print(np.c_[mean_train[0],std_train[0]])
print("class 2:")
print(np.c_[mean_train[1],std_train[1]])
print("class 3:")
print(np.c_[mean_train[2],std_train[2]])

# gus function and p(x|w)
def gus(x, mean, sigma):
    return np.exp(-1*((x-mean)**2)/(2*(sigma**2)))/(math.sqrt(2*np.pi) *
sigma)
def prod13(x):
    return [np.prod([gus(x[i],mean_train[k][i],std_train[k][i]) for i in
range(13)]) for k in range(3)]

# calculate p(w)
p_w = [np.sum(np.where(dataset[:,0]==i,1,0))/len(dataset) for i in
range(1,4)]

# test and save file
predict = np.zeros(len(testset))
output = np.zeros((len(testset),4))
sum = 0
for i in range(len(testset)):
    label = testset[i][0]
    x = testset[i][1:]
    p = [p_w[i]*prod13(x)[i] for i in range(3)]
    output[i,0] = predict[i] = np.argmax(p)+1
```

```
        output[i,1:] = p/np.sum(p)
        if predict[i] == label:
            sum = sum + 1
print('ACC:',sum/len(testset))
np.savetxt('BYS_result.csv', output,delimiter=',')
```

实验三 KNN分类任务

1. 问题描述

1.1 概述

- 利用KNN算法对输血服务中心数据集中的测试集进行分类。

1.2 数据说明

- 输血服务中心数据集是UCI上的公开数据集。数据集包含多名献血者的信息如最近一次献血到现在的时间跨度，献血总次数，献血总量，以及首次献血 到现在的时间跨度。数据集的相关信息如表1所示:

表1 输血服务中心数据集信息

样例数量	特征维度	特征类型	类别数量
798	4	数值	2

- 数据集已被划分为训练集、验证集和测试集, 分别存储于data文件夹中的train_data.csv, val_data.csv, test_data.csv。train_data.csv和val data.csv文件包含data, label字段, 分别存储着特征$X \in \mathbb{R}^{N\times d}$ 和标记 $Y \in \mathbb{R}^{N\times 1}$ 。其中, N 是样例数量, $d = 4$ 为特征维度, 每个样例的标记 $y \in \{0, 1\}$。test data.csv 文件仅包含data字段。

1.3 任务说明

- 任务一: 利用**欧式距离、切比雪夫距离、曼哈顿距离**作为KNN算法的度量函数对**测试集**进行分类。**实验报告中，要求分析三种距离度量在该数据集上的优劣同时，要求在验证集上分析近邻数k对KNN算法分类精度的影响。**
- 任务二：利用**马氏距离**作为KNN算法的度量函数，对**测试集**进行分类。

2. 实现步骤与流程

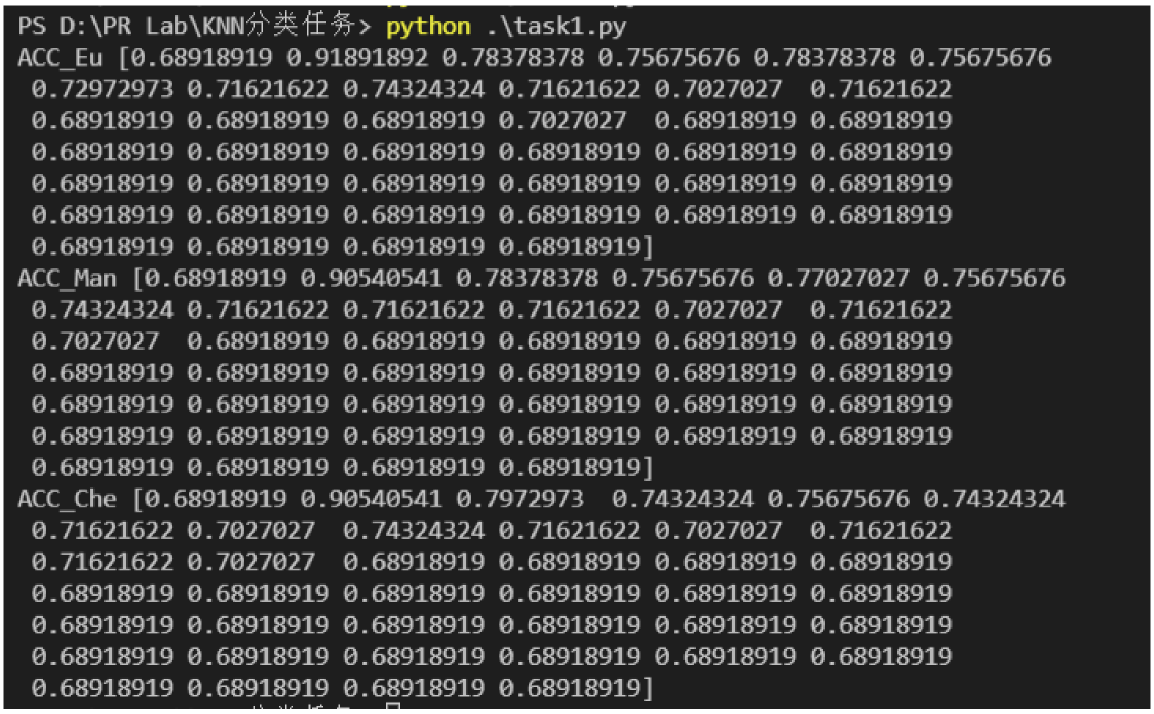
2.1 task1

- 读取数据集并对数据集归一化，由于KNN是通过衡量距离进行分类的算法，所以需要归一化，否则分类会产生对某个特征的偏好。此处归一化需要对测试集，数据集统一归一化不然会产生错误。
- 对于一个验证集上的x,求出其与训练集上所有点的距离，距离可以用不同方式求解
- 选择距离最短的k个点作为分类依据，选择这k个点中最多的类别作为x的分类
- 读取验证集所有数据进行预测与比对，得出预测准确率

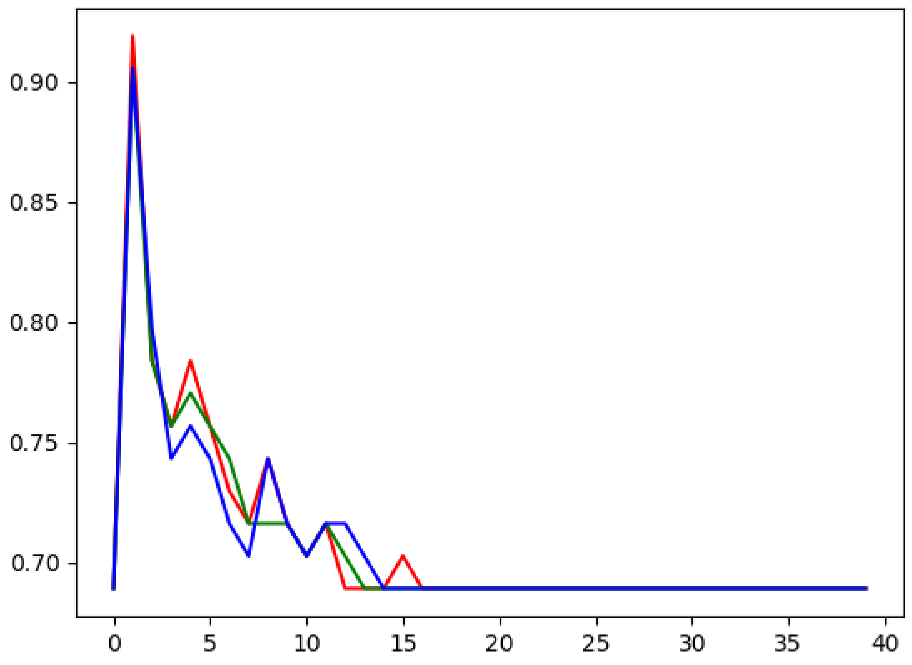
2.2 task2

- 用梯度下降逼近马氏距离中矩阵A的最优解
- 用得到的A计算验证集上的预测结果并进行比对，计算得出准确率

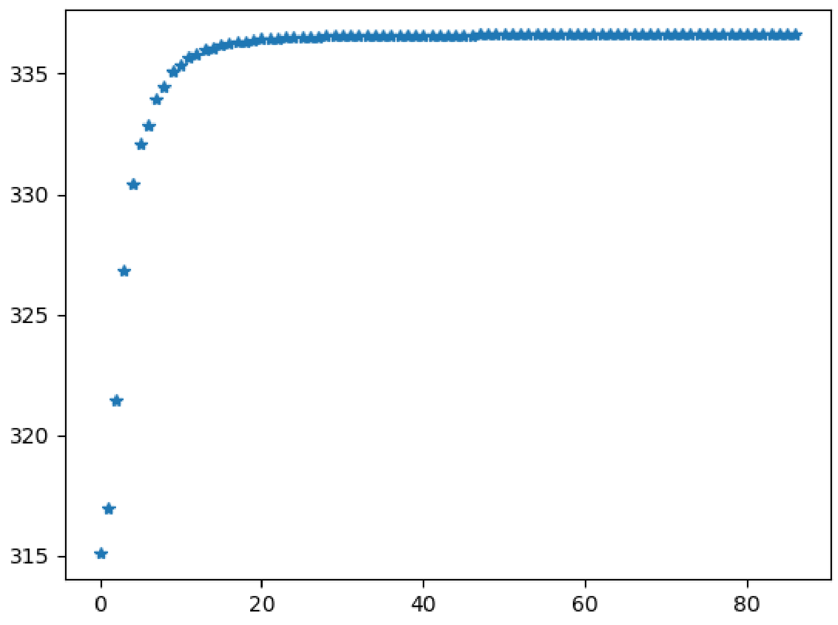
3. 实现结果与分析

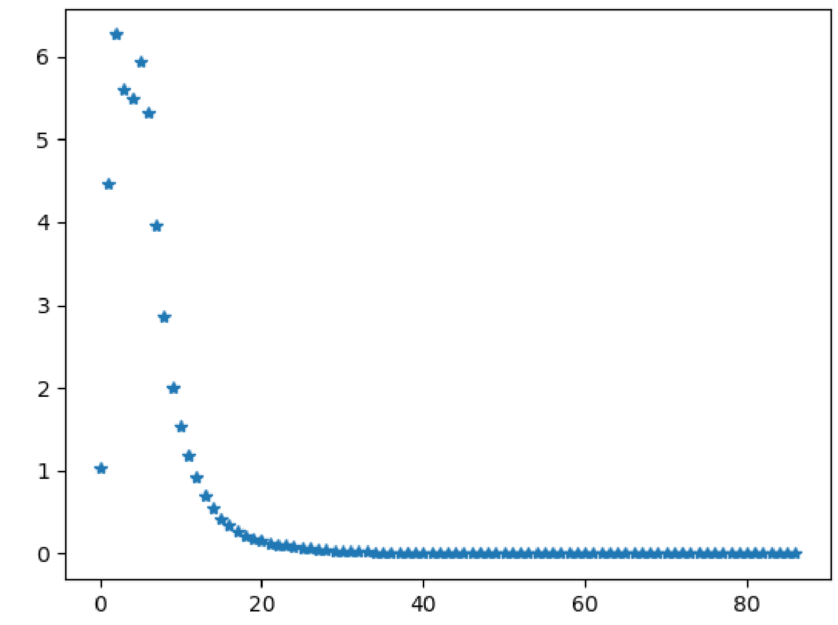
- task1
  - 用三种不同的衡量标准分别调整k的大小，观察分类情况，可以看到它们在k=1时其分类效果很好，大致在90%左右，其他的k值最高只能打到80%不到的准确率。
  - 

```
PS D:\PR Lab\KNN分类任务> python .\task1.py
ACC_Eu [0.68918919 0.91891892 0.78378378 0.75675676 0.78378378 0.75675676
 0.72972973 0.71621622 0.74324324 0.71621622 0.7027027  0.71621622
 0.68918919 0.68918919 0.68918919 0.7027027  0.68918919 0.68918919
 0.68918919 0.68918919 0.68918919 0.68918919 0.68918919 0.68918919
 0.68918919 0.68918919 0.68918919 0.68918919 0.68918919 0.68918919
 0.68918919 0.68918919 0.68918919 0.68918919 0.68918919 0.68918919
 0.68918919 0.68918919 0.68918919 0.68918919]
ACC_Man [0.68918919 0.90540541 0.78378378 0.75675676 0.77027027 0.75675676
 0.74324324 0.71621622 0.71621622 0.71621622 0.7027027  0.71621622
 0.7027027  0.68918919 0.68918919 0.68918919 0.68918919 0.68918919
 0.68918919 0.68918919 0.68918919 0.68918919 0.68918919 0.68918919
 0.68918919 0.68918919 0.68918919 0.68918919 0.68918919 0.68918919
 0.68918919 0.68918919 0.68918919 0.68918919 0.68918919 0.68918919
 0.68918919 0.68918919 0.68918919 0.68918919]
ACC_Che [0.68918919 0.90540541 0.7972973  0.74324324 0.75675676 0.74324324
 0.71621622 0.7027027  0.74324324 0.71621622 0.7027027  0.71621622
 0.71621622 0.7027027  0.68918919 0.68918919 0.68918919 0.68918919
 0.68918919 0.68918919 0.68918919 0.68918919 0.68918919 0.68918919
 0.68918919 0.68918919 0.68918919 0.68918919 0.68918919 0.68918919
 0.68918919 0.68918919 0.68918919 0.68918919 0.68918919 0.68918919
 0.68918919 0.68918919 0.68918919 0.68918919]
```

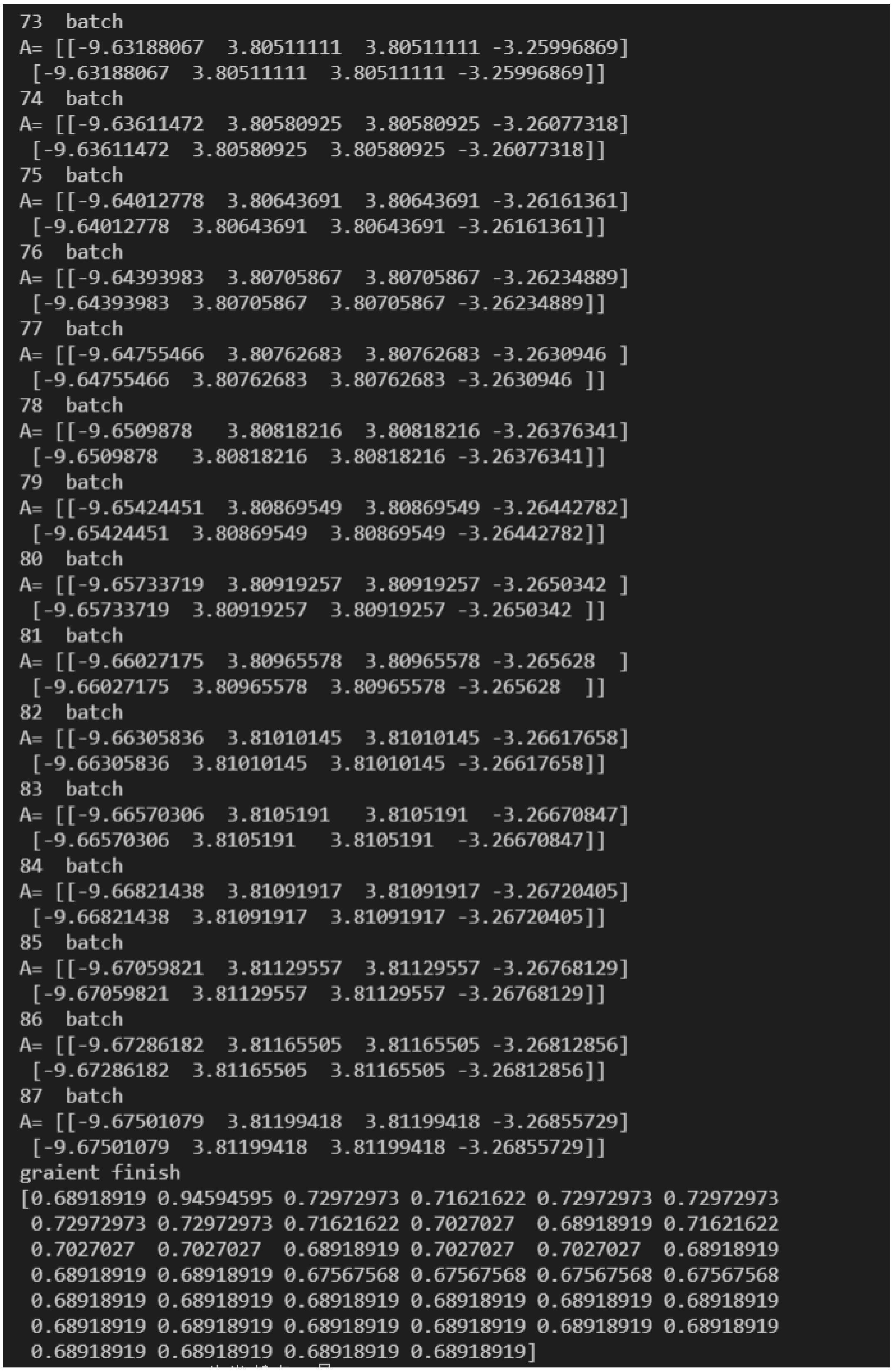
- 

- task2
  - 首先对马氏距离进行训练
    - 推导梯度计算公式
      - $$p_i = \sum_{j \in C_i} p_{ij}$$
        $C_i = \{j \mid y_j = y_i\}$
      - $f(\mathbf{A}) = \sum_i p_i$
      - $\frac{\partial f}{\partial \mathbf{A}} = 2\mathbf{A} \sum_i \left(p_i \sum_k p_{ik} \mathbf{x}_{ik} \mathbf{x}_{ik}^T - \sum_{j \in C_i} p_{ij} \mathbf{x}_{ij} \mathbf{x}_{ij}^T \right)$
    - 用梯度下降最大化目标函数得到结果A
    - 函数随着训练过程慢慢增大，梯度归于0并最终在86步停止，因为此时梯度已经小于0.00001

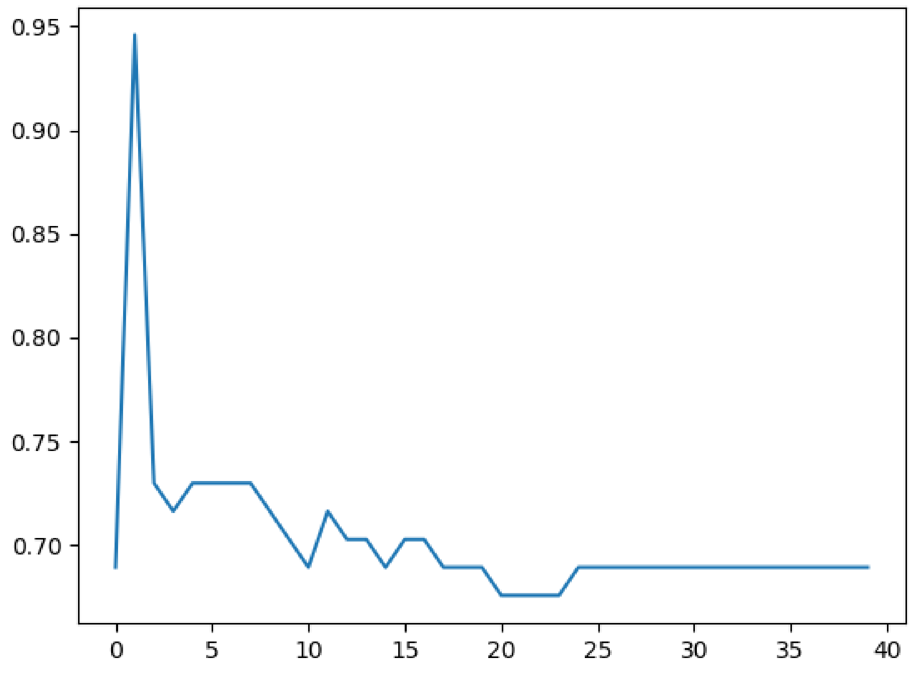
- 

- 

- 训练得出A后可以看到仍然是在k=1时准确率最高，可以打到94.5%左右



```
73  batch
A= [[-9.63188067  3.80511111  3.80511111 -3.25996869]
 [-9.63188067  3.80511111  3.80511111 -3.25996869]]
74  batch
A= [[-9.63611472  3.80580925  3.80580925 -3.26077318]
 [-9.63611472  3.80580925  3.80580925 -3.26077318]]
75  batch
A= [[-9.64012778  3.80643691  3.80643691 -3.26161361]
 [-9.64012778  3.80643691  3.80643691 -3.26161361]]
76  batch
A= [[-9.64393983  3.80705867  3.80705867 -3.26234889]
 [-9.64393983  3.80705867  3.80705867 -3.26234889]]
77  batch
A= [[-9.64755466  3.80762683  3.80762683 -3.2630946 ]
 [-9.64755466  3.80762683  3.80762683 -3.2630946 ]]
78  batch
A= [[-9.6509878   3.80818216  3.80818216 -3.26376341]
 [-9.6509878   3.80818216  3.80818216 -3.26376341]]
79  batch
A= [[-9.65424451  3.80869549  3.80869549 -3.26442782]
 [-9.65424451  3.80869549  3.80869549 -3.26442782]]
80  batch
A= [[-9.65733719  3.80919257  3.80919257 -3.2650342 ]
 [-9.65733719  3.80919257  3.80919257 -3.2650342 ]]
81  batch
A= [[-9.66027175  3.80965578  3.80965578 -3.265628  ]
 [-9.66027175  3.80965578  3.80965578 -3.265628  ]]
82  batch
A= [[-9.66305836  3.81010145  3.81010145 -3.26617658]
 [-9.66305836  3.81010145  3.81010145 -3.26617658]]
83  batch
A= [[-9.66570306  3.8105191   3.8105191  -3.26670847]
 [-9.66570306  3.8105191   3.8105191  -3.26670847]]
84  batch
A= [[-9.66821438  3.81091917  3.81091917 -3.26720405]
 [-9.66821438  3.81091917  3.81091917 -3.26720405]]
85  batch
A= [[-9.67059821  3.81129557  3.81129557 -3.26768129]
 [-9.67059821  3.81129557  3.81129557 -3.26768129]]
86  batch
A= [[-9.67286182  3.81165505  3.81165505 -3.26812856]
 [-9.67286182  3.81165505  3.81165505 -3.26812856]]
87  batch
A= [[-9.67501079  3.81199418  3.81199418 -3.26855729]
 [-9.67501079  3.81199418  3.81199418 -3.26855729]]
graient finish
[0.68918919 0.94594595 0.72972973 0.71621622 0.72972973 0.72972973
 0.72972973 0.72972973 0.71621622 0.7027027  0.68918919 0.71621622
 0.7027027  0.7027027  0.68918919 0.7027027  0.7027027  0.68918919
 0.68918919 0.68918919 0.67567568 0.67567568 0.67567568 0.67567568
 0.68918919 0.68918919 0.68918919 0.68918919 0.68918919 0.68918919
 0.68918919 0.68918919 0.68918919 0.68918919 0.68918919 0.68918919
 0.68918919 0.68918919 0.68918919 0.68918919]
```

▪ 

4. 代码附录

task1.py

```
import matplotlib
import  matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import random

# define distances
def Euclidean(a,b):
    return np.sqrt(np.sum(np.square(a-b)))

def Manhattan(a,b):
    return np.sum(np.abs(a-b))

def Chebyshev(a,b):
    return np.max(np.abs(a-b))

# read datasets
trainset_read = np.array(pd.read_csv('train_data.csv',header=0))
testset_read = np.array(pd.read_csv('test_data.csv',header=0))
evalset_read = np.array(pd.read_csv('val_data.csv',header=0))
trainset_without_label = trainset_read[:,:4]
train_labels = trainset_read[:,4]
evalset_without_label = evalset_read[:,:4]
eval_labels = evalset_read[:,4]
```

```
t_max = np.array([trainset_without_label.max(axis=0),
testset_read.max(axis=0),
evalset_without_label.max(axis=0)]).max(axis=0)
t_min = np.array([trainset_without_label.min(axis=0),
testset_read.min(axis=0),
evalset_without_label.min(axis=0)]).min(axis=0)
trainset_without_label = (trainset_without_label - t_min) / (t_max -
t_min)
evalset_without_label = (evalset_without_label - t_min) / (t_max -
t_min)
testset = (testset_read - t_min) / (t_max - t_min)

def calAllDistance(x, disFunc):
    return [disFunc(x,i) for i in trainset_without_label]
def nearest_k_label(x, disFunc, k):
    sort_args = np.argsort(calAllDistance(x, disFunc))
    return train_labels[sort_args[:k]]
nearest_k_label(testset[0], Chebyshev, 8)

def classify(X, disFunc, k):
    predict = np.zeros(len(X))
    for i in range(len(X)):
        label_k = nearest_k_label(X[i], disFunc, k)
        sum = np.sum(label_k)
        predict[i] = 1 if sum*np.sum(train_labels)>(len(label_k)-sum)*
(len(train_labels)-np.sum(train_labels)) else 0
    return predict

def cal_ACC(disFunc):
    allACC = np.zeros(40)
    for k in range(40):
        addon = classify(evalset_without_label, disFunc, k) +
eval_labels
        acc = np.sum(np.where(addon==1,0,1))/len(eval_labels)
        allACC[k] = acc
    return allACC
ACC_Eu, ACC_Man, ACC_Che = cal_ACC(Euclidean), cal_ACC(Manhattan),
cal_ACC(Chebyshev)
print("ACC_Eu", ACC_Eu)
print("ACC_Man", ACC_Man)
print("ACC_Che", ACC_Che)
plt.plot(ACC_Eu,c='r')
plt.plot(ACC_Man,c='g')
plt.plot(ACC_Che,c='b')
plt.show()

k_Eu, k_Man, k_Che = np.argmax(ACC_Eu), np.argmax(ACC_Man),
np.argmax(ACC_Che)
np.savetxt('task1_test_prediction_Euclidean.csv', np.c_[testset_read,
classify(testset, Euclidean, k_Eu)], delimiter=',')
np.savetxt('task1_test_prediction_Manhattan.csv', np.c_[testset_read,
classify(testset, Manhattan, k_Man)], delimiter=',')
np.savetxt('task1_test_prediction_Chebyshev.csv', np.c_[testset_read,
classify(testset, Chebyshev, k_Che)], delimiter=',')
```

task2.py

```
import matplotlib
import  matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import random
import math

# define distances
def Mahalanobis_distance(xi, A, xj):
    temp = np.dot(A,(xi-xj).T)
    return np.dot(temp.T,temp)

# read datasets
trainset_read = np.array(pd.read_csv('train_data.csv',header=0))
testset_read = np.array(pd.read_csv('test_data.csv',header=0))
evalset_read = np.array(pd.read_csv('val_data.csv',header=0))
trainset_without_label = trainset_read[:,:4]
train_labels = trainset_read[:,4]
evalset_without_label = evalset_read[:,:4]
eval_labels = evalset_read[:,4]
t_max = np.array([trainset_without_label.max(axis=0),
testset_read.max(axis=0),
evalset_without_label.max(axis=0)]).max(axis=0)
t_min = np.array([trainset_without_label.min(axis=0),
testset_read.min(axis=0),
evalset_without_label.min(axis=0)]).min(axis=0)
trainset_without_label = (trainset_without_label - t_min) / (t_max -
t_min)
evalset_without_label = (evalset_without_label - t_min) / (t_max -
t_min)
testset = (testset_read - t_min) / (t_max - t_min)

def f_gradient(A):
    data_divide = [trainset_without_label[np.where(train_labels==0)],
trainset_without_label[np.where(train_labels==1)]]
    len_train = len(trainset_without_label)
    exps = [[math.exp(-Mahalanobis_distance(trainset_without_label[i],
A, trainset_without_label[j])) for j in range (len_train)] for i in
range(len_train)]
    exps_sum = np.sum(exps,axis=1)
    p_ij = [exps[i]/exps_sum[i] for i in range(len(exps_sum))]
    for i in range(len(p_ij)):
        p_ij[i][i] = 0
    p_i = [np.sum(p_ij[i][np.where(train_labels==train_labels[i])]) for
i in range(len(train_labels))]
    sum = np.zeros((4,4))
    for i in range(len(train_labels)):
        sum1 = np.multiply(p_i[i],np.sum([np.multiply(p_ij[i]
[k],np.outer((trainset_without_label[i]-trainset_without_label[k]).T,
(trainset_without_label[i]-trainset_without_label[k]))) for k in
range(len(train_labels))], axis=0))
        sum2 = np.sum([np.multiply(p_ij[i]
[k],np.outer((trainset_without_label[i]-trainset_without_label[k]).T,
(trainset_without_label[i]-trainset_without_label[k]))) for k in
np.array(np.where(train_labels==train_labels[i]))[0]], axis=0)
        sum = sum + (sum1-sum2)
    return np.sum(p_i), np.dot(np.multiply(2,A),sum)

def Gredient_Descent_batch(A, lr = 1 ):
    print("graient begin")
    epoch = 100
    histroy = []
    history_f = []
    for j in range(epoch):
        f, gd = f_gradient(A)
        sum = np.sum(np.square(gd))
        # lr /= ((j+1)**0.5)
        if sum >= 0.00001:
            A += lr * gd
            histroy.append(sum)
            history_f.append(f)
        else:
            break
        print(j+1," batch")
        print("A=", A)
    print("graient finish")
    return A, histroy, history_f
A = 0.1*np.ones([2, 4])
A_better, history, history_f = Gredient_Descent_batch(A)
plt.plot(history_f, '*')
plt.show()
plt.plot(history,'*')
plt.show()

def Mahalanobis_distance_better(xi,xj):
    # [[-9.67501079,3.81199418,3.81199418,-3.26855729],
[-9.67501079,3.81199418,3.81199418,-3.26855729]]
    # ACC=0.945
    A = A_better
    temp = np.dot(A,(xi-xj).T)
    return np.dot(temp.T,temp)

def calAllDistance(x, disFunc):
    return [disFunc(x,i) for i in trainset_without_label]
def nearest_k_label(x, disFunc, k):
    sort_args = np.argsort(calAllDistance(x, disFunc))
    return train_labels[sort_args[:k]]

def classify(X, disFunc, k):
    predict = np.zeros(len(X))
    for i in range(len(X)):
        label_k = nearest_k_label(X[i], disFunc, k)
        sum = np.sum(label_k)
        predict[i] = 1 if sum*np.sum(train_labels)>(len(label_k)-sum)*
(len(train_labels)-np.sum(train_labels)) else 0
    return predict

def cal_ACC(disFunc):
    allACC = np.zeros(40)
    for k in range(40):
        addon = classify(evalset_without_label, disFunc, k) +
eval_labels
        acc = np.sum(np.where(addon==1,0,1))/len(eval_labels)
        allACC[k] = acc
    return allACC
ACC_Mah = cal_ACC(Mahalanobis_distance_better)
print(ACC_Mah)
plt.plot(ACC_Mah)
plt.show()

k_Mah = np.argmax(ACC_Mah)
np.savetxt('task2_test_prediction_Mahalanobis.csv', np.c_[testset_read,
classify(testset_read, Mahalanobis_distance_better,
k_Mah)].astype(np.uint8), delimiter=',')
```