模式识别实验报告

专业：	吴健雄学院 人工智能
学号：	61518218
年级：	18
姓名：	沈书杨

签名：

时间：

实验一 线性分类器分类任务

1. 问题描述

1.1 问题背景

利用线性分类器对 MNIST 数据集中的测试集进行分类。

MNIST 是著名的手写体数字识别数据集。该数据集由训练数据集和测试数据集两部分组成，其中训练数据集包含了 60000 张样本图片及其对应标签，每张图片由 28 × 28 的像素点构成；训练数据集包含了 10000 张样本图片及其对应标签，每张图片由 28 × 28 的像素点构成。

1.2 问题介绍

任务一：对 MNIST 数据集做恰当预处理，在处理后的训练集上学习一个多类线性分类器，并对处理后的测试集进行分类。

任务二：利用 LDA 降维技术进行对 MNIST 数据集的降维工作，在降维后的数据集上完成多类线性分类器的训练和测试。要求比较应用 LDA 降维技术前后，分类器准确率的变化。（对于降维后的数据，可以尝试利用可视化方法展示结果。）

1.3 问题分析

本问题是对 0-9 的数字进行识别，针对这种问题有几个基本思路：

a. 使用一对一的线性分类器并对于不同的类别给出置信度，选出置信度最高的作为结果。需要训练 10 个模型。
b. 使用一对其余的线性分类器同样需要给出置信度，选出置信度最高的作为结果。需要训练 10 个模型。
c. 使用一对多的线性分类器，以 softmax 作为激活函数并以交叉熵作为损失。需要训练一个模型。

在本问题中，如果使用 a, b 思路会导致无法进行矩阵运算训练优化，增加训练时间，而且最后还是要通过归一化变成类似 softmax 结果的形式。因此在本问题中，我选择用 c 解决问题。

2. 实现步骤与流程

2.1 数据读取

对于 mnist 数据的读取使用了网络资源。

2.2 数据预处理

对 mnist 数据进行了标准化：即对所有数据减去它们的均值后除以它们的方差。对标签进行了热点化，以方便在之后的矩阵运算中进行求导优化。

2.3 线性层类

需要构建一个可以反向传播并根据上一曾输入的梯度更新的线性层。

对 weight 和 bias 的初始化均使用均值为 0，方差为 1 的正态分布。

线性层输出的向量维度为 1 by #of_category。

反向传播：（softmax 和 cross entropy 的求导相对简单，此处略去）

$\frac{\partial L}{\partial W}$的理论结果是 1 by 10 by shape_of_W 的复杂张量，但是从$\frac{\partial L}{\partial W_i}$ (此处 Wi 指 W 权重矩阵的列向量)观察可以发现，结果对应的列只与 Wi 有关。即对 weight 求导结果只与输入的列向量有关，且对任意 Wi 都相同，当输入向量的维度增加时可以推广。对 bias 的求导就是一个单位矩阵，这里不再赘述。

2. 4 Softmax 层类

Softmax 与交叉熵的组合求损失函数实际上可以通过一个式子表示，可以在代码中清晰体现，即 $\hat{y} - y$ 。在这里 $\hat{y} - y$ 的维度为 10 by 1，与线性层的 BP 结果可以进行链式组合，在形式上具有一致性。

2. 5 LDA

在本题中 LDA 并不能直接使用，因为经过实践发现计算得到的 Sb 矩

阵是奇异阵。解决这个问题有两种思路：

a. 先将数据进行 PCA 降维，在进行 LDA

b. 对奇异阵 Sb 加上一个很小的单位阵使其可逆

实践证明，两种方案下都会出现虚数根，后者在训练 20 个 epoch 后正确率依然只有 10%。

对方案 b 尝试参数筛选，选出单位阵的系数 1e-10 使其非奇异，尝试 scale 相差不大的前 12 个特征值及对应的特征向量，依然效果不佳，训练 100 个 epoch 后准确率在 20%左右。尝试加大降维后的维度，结果相似。

尝试方案 a，先用 PCA 降至 700 维。发现正确率还是很低。仔细观察 PCA 时的特征值：

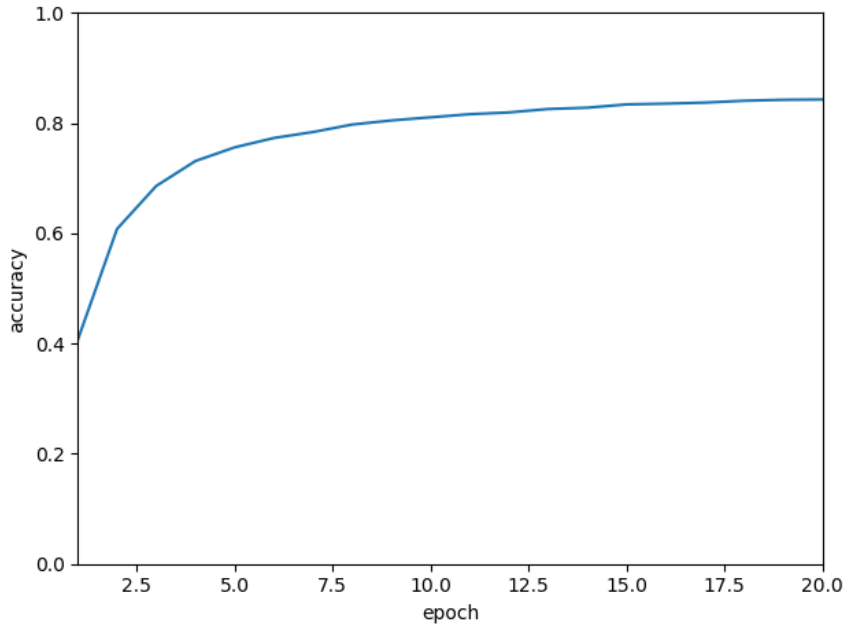
```
8.73199512e-03  6.38472099e-03  5.55077960e-03  4.84925423e-03
4.38081218e-03  3.88002938e-03  2.94399428e-03  2.59485130e-03
2.48519952e-03  2.12076549e-03  1.89779190e-03  1.82023279e-03
1.54384692e-03  1.52251621e-03  1.42041883e-03  1.33432075e-03
1.19180420e-03  1.14891780e-03  1.06826716e-03  1.03715369e-03
9.59307584e-04  9.05813527e-04  8.57999548e-04  8.21082238e-04
7.94863704e-04  7.55196715e-04  7.31136486e-04  7.07550983e-04
6.70090750e-04  6.21616401e-04  5.92135019e-04  5.83185964e-04
5.42216743e-04  5.27790824e-04  5.12889941e-04  4.89141599e-04
4.55092969e-04  4.38962007e-04  4.33177131e-04  4.24932497e-04
4.10968555e-04  4.00251440e-04  3.76556203e-04  3.58303416e-04
3.46389763e-04  3.37507906e-04  3.25725796e-04  3.16352344e-04
3.05975197e-04  2.89613960e-04  2.87042934e-04  2.81453443e-04
2.66317218e-04  2.59993552e-04  2.55652865e-04  2.42508131e-04
2.44230619e-04  2.32566622e-04  2.28335933e-04  2.20247302e-04
2.16400928e-04  2.15282302e-04  2.07315341e-04  1.99328634e-04
1.92300223e-04  1.86455679e-04  1.82692317e-04  1.77059688e-04
1.73523560e-04  1.69726414e-04  1.68236918e-04  1.62933582e-04
1.57368567e-04  1.59765588e-04  1.49144773e-04  1.47467329e-04
1.45279253e-04  1.39569572e-04  1.32818171e-04  1.28825960e-04
1.27852552e-04  1.27005831e-04  1.26125156e-04  1.22131899e-04
1.20432047e-04  1.19126648e-04  1.17111981e-04  1.13257209e-04
1.10517464e-04  1.09398439e-04  1.05304451e-04  1.03360157e-04
1.01894060e-04  9.97719066e-05  9.80768864e-05  9.62066408e-05
9.35825261e-05  9.37527262e-05  9.11076685e-05  9.04517125e-05
8.85392207e-05  8.54502039e-05  8.46993098e-05  8.24341795e-05
8.06981447e-05  8.16859543e-05  7.69460799e-05  7.78655737e-05
7.60866005e-05  7.40059373e-05  7.12246151e-05  7.05967941e-05
7.07166429e-05  6.91773885e-05  6.87440526e-05  6.77608585e-05
6.62937396e-05  6.54253688e-05  6.47523130e-05  6.35974220e-05
6.25724146e-05  6.22787894e-05  6.14817918e-05  6.06501495e-05
6.00037762e-05  5.80592699e-05  5.71890464e-05  5.68338278e-05
5.60498919e-05  5.44623828e-05  5.43092796e-05  5.34897937e-05
5.29343427e-05  5.27734921e-05  5.19017267e-05  5.23074847e-05
5.08703872e-05  4.99162107e-05  4.81531813e-05  4.72464704e-05
4.73213454e-05  4.59115420e-05  4.52559545e-05  4.48728345e-05
4.50856339e-05  4.41851440e-05  4.36873612e-05  4.34452103e-05
4.26504845e-05  4.21409860e-05  4.19833150e-05  4.16889928e-05
4.13260101e-05  3.98338446e-05  4.05239663e-05  4.03824785e-05
3.92907784e-05  3.84606631e-05  3.82384835e-05  3.79715370e-05
3.73867866e-05  3.68580373e-05  3.60650418e-05  3.58003742e-05
3.51355576e-05  3.54583213e-05  3.47179741e-05  3.40923243e-05
3.41712692e-05  3.36460531e-05  3.32389260e-05  3.19097507e-05
```

PCA 特征值

可以看到特征值的 scale 有很大差异，最小的特征值数量级为 e-23。再次尝试，使用 e-3~e-4 数量级的特征值对应的特征向量降维后使用 LDA，训练 20 个 epoch，准确率达到 80%以上。得出了可以接受的结果。

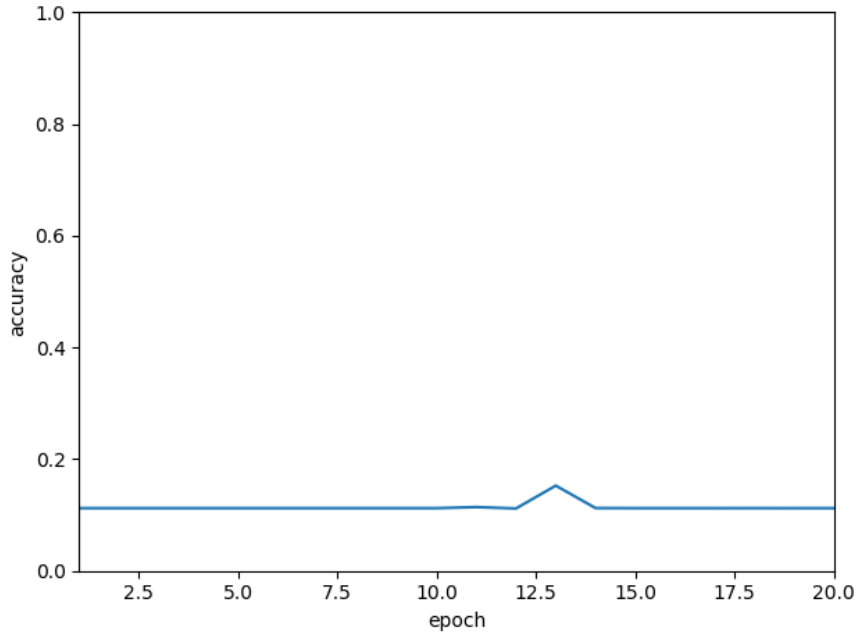
3. 实验结果与分析

3.1 任务 1



上图为训练 20 个 epoch 的实验结果，可以看到模型已经接近收敛，说明单层的线性层还是不能很好的执行该类任务。但从实验结果上说，可以证明代码的正确性及数学推导的正确性。

3.2 任务 2

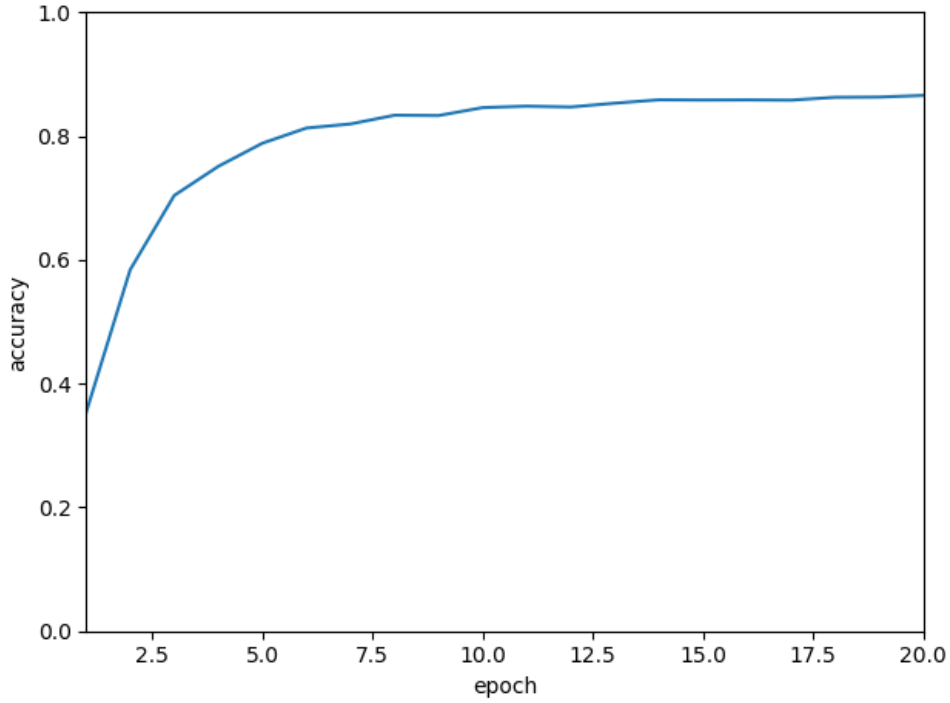


图一

[0.0993, 0.09871666666666666, 0.0993, 0.10441666666666667, 0.10441666666666667, 0.11236666666666667, 0.15505, 0.09915, 0.2101, 0.10388333333333333, 0.09871666666666666, 0.12408333333333334, 0.1989, 0.10441666666666667, 0.11306666666666666, 0.11671666666666666, 0.11236666666666667, 0.09756666666666666, 0.10218333333333333, 0.3049, 0.0993, 0.18186666666666668, 0.10218333333333333, 0.10218333333333333, 0.11236666666666667, 0.11236666666666667, 0.09915, 0.15528333333333333, 0.1236, 0.17276666666666668, 0.11236666666666667, 0.24215, 0.13156666666666667, 0.15321666666666667, 0.0993, 0.09736666666666667, 0.10441666666666667, 0.11236666666666667, 0.1003, 0.1816, 0.20981666666666668, 0.20263333333333333, 0.13618333333333332, 0.1132, 0.19668333333333332, 0.21995, 0.11236666666666667, 0.14653333333333332, 0.10718333333333334, 0.16891666666666666]

图二

图一为单位阵+LDA 训练 20 个 epoch 的实验结果，图二为单位阵+LDA 训练 100 个 epoch 的实验结果，由于在服务器上跑，没有进行可视化，但是如上文所说，效果没有太大提升，只到了 20%左右。



上图为 PCA+LDA 训练 20 个 epoch 的结果（PCA 至 80 维，LDA 至 20 维）。

4. 代码附录

```
import numpy as np
from struct import unpack
import gzip,math,random,copy
import matplotlib.pyplot as plt

def read_image(path):
    with gzip.open(path, 'rb') as f:
        magic, num, rows, cols = unpack('>4I', f.read(16))
```

```
        img = np.frombuffer(f.read(), dtype=np.uint8).reshape(num, 28 *
28)
    return img

def read_label(path):
    with gzip.open(path, 'rb') as f:
        magic, num = unpack('>2I', f.read(8))
        lab = np.frombuffer(f.read(), dtype=np.uint8)
    return lab

def normalize_image(img):
    mean=np.mean(img)
    std=np.sum((img-mean)**2)/(img.shape[0]*img.shape[1])
    return (img-mean)/std

def one_hot_label(label):
    lab = np.zeros((label.size, 10))
    for i, row in enumerate(lab):
        row[label[i]] = 1
    return lab

class Linear:
    def __init__(self,shape,lr=0.003):
        self.shape=shape
        self.lr=lr
        self.weight=np.random.normal(0, 1,
(self.shape[0],self.shape[1]))
        self.bias=np.random.normal(0, 1, (1,self.shape[1]))

    def __call__(self,x):
        self.input=x
        self.output=np.matmul(x,self.weight)+self.bias
        return self.output

    def __backward__(self):
        return [self.input.transpose(),np.identity(self.shape[1])]

    def update(self,par):
        par_w=np.matmul(self.__backward__()[0],par)
        par_b=np.matmul(par,self.__backward__()[1])
        self.weight-=self.lr*par_w.astype(np.float)
        self.bias-=self.lr*par_b

class Softmax:
```

```
    def __init__(self,x,y):
        self.input = x.flatten()
        self.input-=self.input.max()
        self.label= y
        dnome=0
        for data in self.input:
            dnome+=math.pow(math.e,data)
        self.output=np.array([math.pow(math.e,ele)/dnome for ele in
self.input])

    def __call__(self, *args, **kwargs):
        return self.output

    def __backward__(self):
        return np.array([self.output-self.label])


train_data_dir = './线性分类器分类任务/train-images-idx3-ubyte.gz'
train_label_dir = './线性分类器分类任务/train-labels-idx1-ubyte.gz'
test_data_dir = './线性分类器分类任务/t10k-images-idx3-ubyte.gz'
oh_train_label = one_hot_label(read_label(train_label_dir))
train_data = normalize_image(read_image(train_data_dir))
test_data = normalize_image(read_image(test_data_dir))


#************task 1************#
my_iter = [[train_data[i], oh_train_label[i]] for i in
range(len(train_data))]
dataloader=copy.deepcopy(my_iter)
epoch=20
l=Linear([784,10])
res=[]
for i in range(epoch):
    random.shuffle(dataloader)
    for x,y in dataloader:
        l_out=l(np.array([x]))
        l.update(Softmax(l_out,y).__backward__())
    count, total = 0, 0
    for x, y in my_iter:
        total += 1
        count += l(np.array([x])).argmax() == np.array(y).argmax()
    res.append(count / total)
x=range(1,epoch+1,1)
```

```
plt.plot(x,res)
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.xlim(1,epoch)
plt.ylim(0,1)
plt.show()



#************task 2************#

#------------PCA-------------#
pca_mean=np.mean(train_data,axis=0)

temp = train_data - pca_mean
S=np.cov(temp.T)

eigenvalue,featurevector=np.linalg.eig(S)
#print(eigenvalue)
pca_num=80
pca_data=np.matmul(train_data-pca_mean,featurevector[:,:pca_num])
#----------------------------#

##想使用PCA注释下面这行，取消注释上面这行

#pca_data=train_data

#------------LDA-------------#
by_class=[[] for _ in range(10)]
train_label = read_label(train_label_dir)
globle_mean=np.mean(pca_data,axis=0)
by_mean=[]
for i in range(len(pca_data)):
    by_class[train_label[i]].append(pca_data[i])
by_class=np.array(by_class)
for clas in by_class:
    by_mean.append(np.mean(clas,axis=0))

S_w=None
for i in range(10):
    temp = np.array(by_class[i]) - by_mean[i]
    try:
        S_w+=np.cov(temp.T)
    except:
```

```
        S_w=np.cov(temp.T)

S_b=None
for i in range(10):
    temp = (np.array(by_mean[i]) - globle_mean)[None]
    try:
        S_b+=len(by_class[i])*np.matmul(temp.T,temp)
    except:
        S_b=len(by_class[i])*np.matmul(temp.T,temp)

##寻找单位阵系数
'''
scale=10
flag=True
while flag==True:
    for i in range(1,10):
        try:
            np.linalg.inv(S_w + i*np.identity(784)/scale)
            print(i,scale)
            flag=False
            break
        except:
            continue
    scale*=10
'''

S_w_inverse=np.linalg.inv(S_w)
eigenvalue,featurevector=np.linalg.eig(np.matmul(S_w_inverse,S_b))

fnum=20
lda_train_data=np.matmul(pca_data,featurevector[:,:fnum].astype(np.float
64))
#----------------------------#


my_iter_lda = [[lda_train_data[i], oh_train_label[i]] for i in
range(len(pca_data))]
dataloader_lda=copy.deepcopy(my_iter_lda)
epoch=20
l=Linear([fnum,10])
res=[]
for i in range(epoch):
    random.shuffle(dataloader_lda)
    for x,y in dataloader_lda:
```

```
        l_out=l(np.array([x]))
        l.update(Softmax(l_out,y).__backward__())
    count, total = 0, 0
    for x, y in my_iter_lda:
        total += 1
        count += l(np.array([x])).argmax() == np.array(y).argmax()
    res.append(count/total)

x=range(1,epoch+1,1)

plt.plot(x,res)
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.xlim(1,epoch)
plt.ylim(0,1)
plt.show()
```

实验二 KNN 分类任务

1. 问题描述

1.1 问题背景

利用 KNN 算法对 wine 数据集中的测试集进行分类。

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.mat, val_data.mat, test_data.mat。train_data.mat 和 val_data.mat 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 = 13 为特征维度，每个样例的标记 y $\in$ {1,2,3}。test_data.mat 文件仅包含 data 字段。

训练集中包含部分特征值缺失的样例。

1.2 问题介绍

任务一：利用欧式距离作为 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。

马氏距离定义：

$$d_M(\boldsymbol{x}_i,\boldsymbol{x}_j)=\sqrt{(\boldsymbol{x}_i-\boldsymbol{x}_j)^T\boldsymbol{M}(\boldsymbol{x}_i-\boldsymbol{x}_j)}$$

其中，$\boldsymbol{M}\in\mathbb{R}^{d\times d}$是一个半正定矩阵，是可以学习的参数。由于$\boldsymbol{M}$的半正定性质，可将上述定义表述为：

$$d_M(\boldsymbol{x}_i,\boldsymbol{x}_j)=\sqrt{(\boldsymbol{x}_i-\boldsymbol{x}_j)^T\boldsymbol{A}^T\boldsymbol{A}(\boldsymbol{x}_i-\boldsymbol{x}_j)}=\|\boldsymbol{A}\boldsymbol{x}_i-\boldsymbol{A}\boldsymbol{x}_j\|_2$$

其中，矩阵$\boldsymbol{A}\in\mathbb{R}^{e\times d}$。故，马氏距离可以理解为对原始特征进行线性映射，然后计算欧式距离。

给定以下目标函数，在**训练集**上利用**梯度下降法**对马氏距离进行学习：

$$f(\boldsymbol{A})=\underset{\boldsymbol{A}}{argmax}\sum_{i=1}^{N}\sum_{j\in C_i}p_{ij}$$

其中，C_i表示与样例$\boldsymbol{x}_i$同类的样例集合，p_{ij}定义为：

$$p_{ij}=\begin{cases}\dfrac{exp\left(-d_M(\boldsymbol{x}_i,\boldsymbol{x}_j)^2\right)}{\sum_{k\neq i}exp(-d_M(\boldsymbol{x}_i,\boldsymbol{x}_k)^2)} & j\neq i\\ 0 & j=i\end{cases}$$

实验中，矩阵$\boldsymbol{A}$的维度e可任意设置为一合适值，例如$e=2$。

1.3 问题分析

本次训练集中包含部分特征值缺失的样例，针对这种问题有几个基本思路：

a. 使用神经网络生成模型对样本进行学习，并生成缺失处参数。

b. 使用同一列非空的特征均值作为缺失值。

在本问题中，我选择用 b 解决问题。

2. 实现步骤与流程

2.1 KNN 优化

由于本次的 KNN 任务数据集中有三个样本类，不能通过取 k 为奇数的方式避免同类别数相同带来的问题。如果直接用 np.max 或 np.argmax 等取最大函数，返回结果只会是第一个取到的最大值或最大 index。这样的 KNN 是不健康的。为此，我提出的方法是：在选取前 k 个最近的点后，同时统计各个类别到目标点的总距离和总个数。先选出总个数最大的类别，再选出总距离最近的，这样达到的效果最好。我也尝试过使用随机对总数相同的类别进行随机选取，这样做获得的结果平均只有 60%，远不如前一种方法。

2.2 代码的健壮性

对 knn()函数进行了设计，使其可以适用于不同的环境下。可以在第四部分的代码中看到，即使我建立了一个马式距离的类，而且并没有在其中单独设计 knn 方法成员函数，依然能很好的使用普通函数 knn()。（具体详见第 15 页）

2.2 马式距离的求导

$$\frac{\partial f(\boldsymbol{A})}{\partial \boldsymbol{A}}$$

$$f(\boldsymbol{A}) = argmax \sum_{i=1}^{N} \sum\nolimits_{j \in C_i} p_{ij}$$

转化为 Pij 对 A 的求导：

$$p_{ij} = \begin{cases} \dfrac{exp(-d_M(x_i,\ x_j)^2)}{\sum_{k\neq i} exp(-d_M(x_i,\ x_k)^2)} & i \neq j \\ 0 & i = j \end{cases}$$

由公式定义可以看出，求导需要使用分式求导法则。为了简化求导过程，对一些式子进行代换：

$$D = \sum_{k\neq i} exp(-d_M(x_I, x_k)^2)$$

$$d = exp(-d_M(x_i, x_j)^2)$$

则 Pij 对 A 的求导表示为：

$$\frac{\partial p_{ij}}{\partial \boldsymbol{A}} = \frac{\frac{\partial d}{\partial \boldsymbol{A}} D - d\frac{\partial D}{\partial \boldsymbol{A}}}{D^2}$$

现在问题化简为 d 和 D 对 A 的求导，同时，由于 D 可以看作是很多个下标不同的 d 的和，所以主要问题再次化简，下面主要进行 d 对 A 的求导：

首先，

$$\frac{\partial d}{\partial \boldsymbol{A}} = \frac{\partial d}{\partial d_M(x_i, x_j)} \frac{\partial d_M(x_i, x_j)}{\partial \boldsymbol{A}}$$

简化求导，设，

$$l = d_M(x_i, x_j)$$

则上式可以化简为：

$$\frac{\partial d}{\partial \boldsymbol{A}} = \frac{\partial d}{\partial d_M(x_i, x_j)} \frac{\partial d_M(x_i, x_j)}{\partial \boldsymbol{A}} = -2l \cdot exp(-l^2) \frac{\partial l}{\partial \boldsymbol{A}}$$

现在关注 *l* 对 A 的求导，由于马式距离在取欧氏距离化为标量前，形式为：

$$\boldsymbol{A} \cdot (x_i - x_j)$$

输出的是一个 e by 1 的向量，设该向量为 a：

$$a = \boldsymbol{A} \cdot (x_i - x_j)$$

现在，*l* 对 A 的求导变为：

$$\frac{\partial l}{\partial \boldsymbol{A}} = \frac{\partial l}{\partial a} \frac{\partial a}{\partial \boldsymbol{A}}$$

化简到此处，求导的方式已经和问题一中线性分类器的求导几乎一样了，需要注意的是由于马式距离最后取了欧式距离，要注意根式的求导。即：

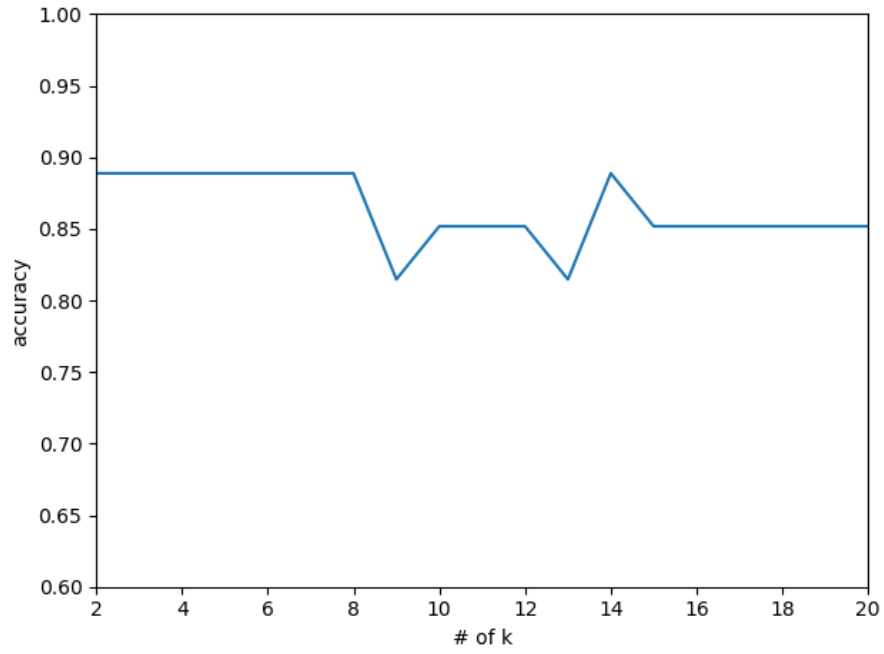
$$\frac{\partial l}{\partial a} = \frac{a}{l}$$

至此，a 对 A 的求导已经可以直接得到了，即$x_i - x_j$，维度的差异直接通过 numpy 的广播机制轻松解决。若对改处有疑问，可以阅读**实验一**的 **2.3** 部分。

求导过程毕。

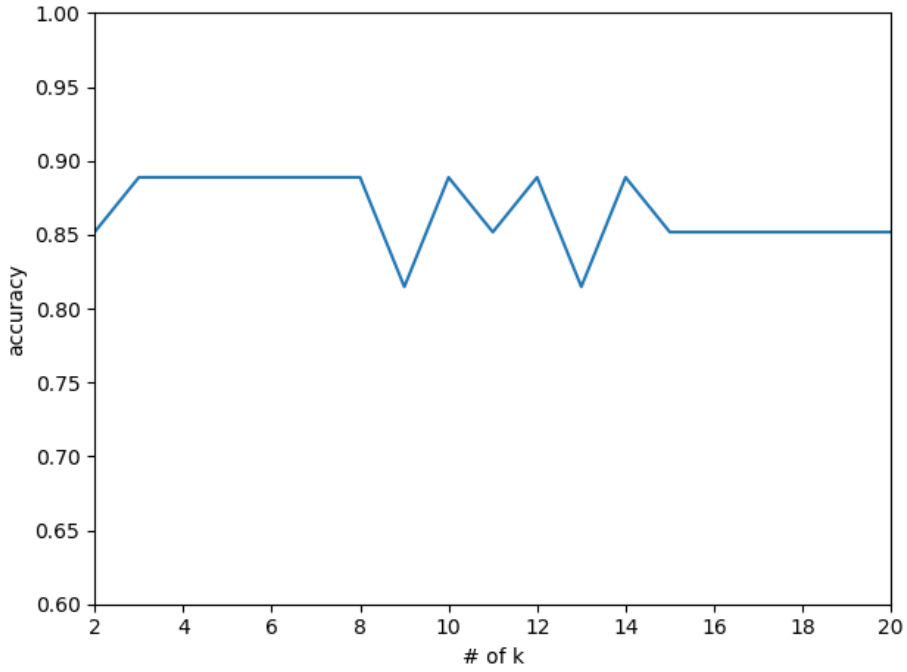
3. 实验结果与分析

3.1 任务 1



上图为 k 的数量由 2 到 20 的实验结果，可以看到结果还是比较稳定的，根本是由于数据集数量太少，包括训练集和测试集。

3.2 任务 2



上图为训练 5 个 epoch 后 k 的数量由 2 到 20 的实验结果，可以看到结果和任务一是十分相近的，其根本原因还是数据集数量太少，包括训练集和测试集。

4. 代码附录

```
import numpy as np
import random,math
from matplotlib import pyplot as plt
from scipy.io import loadmat

##row sample  ---   n by d
##return 1 by n mat
def euclideanDistance(vec,mat=None)->float:
    if isinstance(mat,np.ndarray):
        return np.sum((vec-mat)**2,axis=1)**0.5
    else:
        return np.sum(vec**2)**0.5

def e_neg(x):
    return math.pow(math.e,-x)

def processNan(data:np.ndarray)->np.ndarray:
    def evenColumn(data:np.ndarray,col_num:int)->float:
```

```
        res=0
        num=0
        for row in data:
            if not np.isnan(row[col_num]):
                res+=row[col_num]
                num+=1
        return res/num

    for row in data:
        for i in range(len(row)):
            if np.isnan(row[i]):
                row[i]=evenColumn(data,i)
    return data/100

def findMinIndex(vector,num:int):

    def bubble_sort(indexes,vec):
        flag = True
        while flag:
            flag = False
            for i in range(len(indexes) - 1):
                if vec[indexes[i]] < vec[indexes[i + 1]]:
                    indexes[i], indexes[i + 1] = indexes[i + 1],
indexes[i]
                    flag = True
        return indexes

    index_list=num*[-1]
    for i in range(len(vector)):
        for j in range(len(index_list)):
            if index_list[j]==-1:
                index_list[j]=i
                break
        bubble_sort(index_list,vector)
        if vector[i]<vector[index_list[0]]:
            index_list[0]=i
    return index_list

def knn(metric,train,test,category:int=3,k:int=5):  ## 引入 metric 符号,
增强函数复用性，二个任务都用同一 knn 函数
    res=[]
    data,label=train
    random.seed()
    for i in range(test.shape[0]):
```

```
            dist_list=metric(test[i],data)
            nn=category*[0]
            sumd=category*[0]
            for j in findMinIndex(dist_list,k):
                nn[label[j]-1]+=1
                sumd[label[j]-1]+=dist_list[j]
            max=np.max(np.array(nn))
            quesera=[]
            for j in range(category):
                if nn[j]<max:
                    continue
                else:
                    quesera.append(j)
            ind=quesera[0]
            d=sumd[ind]
            for j in quesera:
                if sumd[j]<d:
                    ind=j
                    d=sumd[j]
            res.append(ind+1)
        return res

    def check(res_hat:list,res_truth:list)->float:
        total=len(res_hat)
        count=0
        for i in range(total):
            if res_truth[i]==res_hat[i]:
                count+=1
        return count/total

    class Mahalanobis:
        def __init__(self,train,category_num:int,e:int=2):
            self.data=train[0]
            self.label=train[1]
            self.cat=category_num
            self.e=e
            self.n,self.d=self.data.shape[0],self.data.shape[1]
            self.A = np.random.normal(0, 1, (self.e,self.d))

        def __call__(self):
            out=0
            for i in range(self.n):
                for j in range(self.n):  ## i=j p_ij=0 && p_ij = p_ji
                    if self.label[j] == self.label[i]:
```

```
                    d = e_neg(self.mahalanobisDistance(self.data[i],
self.data[j]) ** 2)
                    D = 0
                    for k in range(self.n):
                        if k != i:
                            D +=
e_neg(self.mahalanobisDistance(self.data[i], self.data[k]) ** 2)
                    out += d / D
                else:
                    continue
        print('loss: %.4f' % out)

    def __backward__(self):
        partial_A=np.zeros([self.e,self.d])
        for i in range(self.n):
            for j in range(self.n):  ## i=j p_ij=0 && p_ij = p_ji
                if self.label[j] == self.label[i] and i!=j:
                    lm_ij=self.mahalanobisDistance(self.data[i],
self.data[j])
                    diff_ij=self.data[i]-self.data[j]
                    d = e_neg(lm_ij ** 2)
                    e = np.matmul(self.A, diff_ij[None].T)
                    e=e/euclideanDistance(e)
                    second=None
                    D = 0
                    for k in range(self.n):
                        if k != i:
                            lm_ik=self.mahalanobisDistance(self.data[i],
self.data[k])
                            diff_ik=self.data[i] - self.data[k]
                            temp_d=e_neg(lm_ik ** 2)
                            temp_e = np.matmul(self.A, diff_ik[None].T)
                            temp_e = temp_e / euclideanDistance(temp_e)
                            D += e_neg(lm_ik ** 2)
                            temp=-
2*lm_ik*temp_d*D*np.matmul(temp_e,diff_ik[None])
                            try:
                                second+=temp
                            except:
                                second=temp
                    second*=d
                    first=-2*lm_ij*d*np.matmul(e,diff_ij[None])*D
                    partial_A+=(first-second)/(D**2)
                else:
```

```
                    continue
        return partial_A

    def mahalanobisDistance(self,vec1,vec2):
        vec=np.matmul(vec1-vec2,self.A.T)**2
        try:
            vec=np.sum(vec,axis=1)
        except:
            vec=np.sum(vec)
        return vec**0.5

    def train(self,test,epoch:int=5,lr:float=1e-2,k_rangge=[2,20]):
        test_data,test_label=test
        k_s,k_e=k_rangge
        self.lr=lr
        acc=[[] for _ in range(epoch)]
        for i in range(epoch):
            partial=self.__backward__()
            ori_A=self.A.copy()
            self.A+=partial*self.lr
            diff=np.sum((ori_A-self.A)**2)
            if diff<1e-8:
                break
            else:
                for k in range(k_s,k_e+1):

pre=check(knn(self.mahalanobisDistance,[self.data,self.label],test_data,
self.cat,k),test_label)
                    acc[i].append(pre)
                    #print('epoch: %d    accuracy: %.6f    K: %d' %
(i+1,pre,k))
        print('************Train Finished*************')
        return acc

test_data = processNan(loadmat('./KNN 分类任务
/data/test_data.mat')['data'])
val_data,val_label = processNan(loadmat('./KNN 分类任务
/data/val_data.mat')['data']),np.squeeze(loadmat('./KNN 分类任务
/data/val_data.mat')['label'])
train_data,train_label = processNan(loadmat('./KNN 分类任务
/data/train_data.mat')['data']),np.squeeze(loadmat('./KNN 分类任务
/data/train_data.mat')['label'])
train=[train_data,train_label]
test=[val_data,val_label]
```

```
#************task 1************#

start,end= 2,20
x=range(start,end+1)
y=[]
for i in range(start,end+1):
    y.append(check(knn(euclideanDistance,train,val_data,k=i),val_label))
plt.xlabel('# of k')
plt.ylabel('accuracy')
plt.title='Euclidean KNN'
plt.plot(x,y)
plt.xlim(start,end)
plt.ylim(0.6,1)
plt.show()

#************task 2************#
k_s,k_e=2,20
epoch=5
m=Mahalanobis(train,category_num=3,e=5)
y_vec=m.train(test,epoch=epoch,k_rangge=[k_s,k_e])

x=range(k_s,k_e+1)

plt.xlabel('# of k')
plt.ylabel('accuracy')
plt.title='Mahalanobis KNN'
plt.plot(x,y_vec[-1])
plt.xlim(k_s,k_e)
plt.ylim(0.6,1)
plt.show()
```

实验三 隐马尔科夫模型分词任务

1. 问题描述

1.1 问题背景

利用隐马尔科夫模型进行中文语句的分词。

数据集是人民日报 1998 年 1 月份的语料库，对 600 多万字节的中文文章加入了词性标注以及分词处理，由北京大学开发，是中文分词统计的常用资料。可以在语料库基础上构建词典、进行统计、机器学习等。

1.2 问题介绍

中文信息处理是自然语言处理的分支，是指用计算机对中文进行处理。和大部分西方语言不同，书面汉语的词语之间没有明显的空格标记，句子是以字串的形式出现。因此对中文进行处理的第一步就是进行分词，即将字串转变成词串。通过确立状态集合(B, M, E, S)，四个字母分别代表一个字在词语中的开始/中间/结尾/或者单字成词，这样可以将输入的中文句子编为一段状态序列，然后计算初始状态概率、转移概率及发射概率实现整个算法过程。

在人民日报分词语料库上统计语料信息，对隐马尔科夫模型进行训练。利用训练好的模型，对以下语句进行分词测试：

1) 今天的天气很好。
2) 学习模式识别课程是有难度的事情。
3) 我是东南大学的学生。

2. 实现步骤与流程

2.1 HMM 基本描述

本题主要是针对 HMM 原理进行代码实现，使用维特比算法进行预测，使用前向和后向进行 Baum-Welch 算法的实现以对参数进行学习。这里不再赘述原理。

我对训练过程也进行了代码实现包括 HMM forward algorithm，HMM

backward algorithm 和 HMM Baum-Welch learning algorithm。但是由于 B 矩阵过于稀疏，PI 参数总会出现 0 值，陷入局部最优，无法继续训练。对 B 矩阵全位置进行加 1 平滑化后，在训练中还是会因为精度不够而出现 Nan 值而使得训练失败。有兴趣实验的可以对代码中的训练部分取消注释并再次运行尝试。

2.2 模型建立规则

简单的 BMES 设置方式不再赘述，详见代码 88 行至 106 行。这里主要讲述共现概率的统计方法。由于 HMM 中的共现统计是讲究前后顺序的，所以共现词频的统计会少于单个词此频的统计，例如，一个有 6 个字的句子，在其中只能统计 5 组共现关系。我的解决方法是，将每句话的最后一个字和每句话的第一个字作为先后顺序计入统计，这样做的好处有：

a. 不违背 BMES 共现的基本规则，如 B 后不会有 S；E 后不会有 M 等。

b. 可以满足统计后进行先验概率计算时行和为一的限制。

代码实现详见代码第 59 行至 65 行。

3. 实验结果与分析

分词结果：

```
今天 的 天气 很 好 。
学习 模式 识别 课程 是 有 难度 的 事情 。
我 是 东南 大学 的 学生 。
```

可以看出，基本完成了分词的目标，东南大学和模式识别这两个单词由于训练集较小没能精确分割，但整体能满足要求。

4. 代码附录

```
import numpy as np
import random
```

```python
class Preprocsss:  #"./隐马尔科夫分词任务/RenMinData.txt_utf8"
    def __init__(self, dir):
        self.dir = dir

    def mk_vocab(self):
        vocab = {}
        count = 0
        for sentence in self.data:
            for letter in sentence:
                if letter not in vocab.keys():
                    vocab[letter] = count
                    count += 1
        self.vocab = vocab
        self.word_num=len(vocab.keys())

    def mk_label(self):
        f = open(self.dir, 'r', encoding='utf8')
        f.readline()
        cursor = f.readline()
        data = []
        label = []
        while cursor:
            temp=[]
            sentence=cursor.strip()
            check=sentence.replace(' ', '')
            if len(check)>1:
                words = sentence.split(' ')
                for word in words:
                    if len(word) == 1:
                        temp+=[3]
                    else:
                        temp+=[0] + (len(word) - 2) * [1] + [2]
                data.append(check)
                label.append(temp)
                cursor=f.readline()
            else:
                cursor=f.readline()


        f.close()
        self.data_num=len(data)
        self.label_num = len(label)
        self.data=data
        self.label=label
```

```
    def piror(self):
        total_num=0
        num_BMES = 4 * [0]
        piror_a = [4*[0] for _ in range(4)]
        piror_b = [self.word_num * [0] for _ in range(4)]
        for i in range(self.data_num):
            if len(self.data[i])>1:
                piror_b[self.label[i][0]][self.vocab[self.data[i][0]]] +=
1
                num_BMES[self.label[i][0]] += 1
                start = self.label[i][0]
                total_num += len(self.data[i])
                for j in range(1, len(self.data[i])):
                    piror_b[self.label[i][j]][self.vocab[self.data[i][j]]]
+= 1
                    num_BMES[self.label[i][j]] += 1
                    piror_a[start][self.label[i][j]] += 1
                    start=self.label[i][j]
                piror_a[start][self.label[i][0]] += 1
            else:
                continue

self.piror_a=np.divide(np.array(piror_a).T,np.array(num_BMES),where=np.a
rray(num_BMES)!=0).T

self.piror_b=np.divide(np.array(piror_b).T,np.array(num_BMES),where=np.a
rray(num_BMES)!=0).T
        self.part_BMES=np.array(num_BMES)/total_num


    def process(self):
        self.mk_label()
        self.mk_vocab()
        self.piror()
        return self.piror_a,self.piror_b,self.part_BMES

class HMM:
    def __init__(self,a,b,pi,vocab):
        self.a=a
        self.b=b
        self.pi=pi
        self.vocab=vocab
```

```
    ##viertbi predict
    def __call__(self, sentence):
        delta=[]
        psi=[]
        psi.append(4*[0])

delta.append(self.pi*np.array([self.b[i][self.vocab[sentence[0]]] for i
in range(4)]))
        for i in range(1,len(sentence)):
            delta.append(np.max((delta[i-
1]*self.a.T).T,axis=0)*np.array([self.b[ii][self.vocab[sentence[i]]] for
ii in range(4)]))
            psi.append(np.argmax((delta[i-1]*self.a.T).T,axis=0))
        omega=[]
        omega.append(np.argmax(delta[-1]))
        for i in range(len(psi)-1):
            omega.insert(0,psi[-1-i][omega[-1-i]])
        res=''
        for i in range(len(sentence)):
            if omega[i]==2 or omega[i]==3:
                res+=(sentence[i]+' ')
            else:
                res+=sentence[i]
        omega=[['B','M','E','S'][i] for i in omega]
        return res

    def forward(self,sample):
        data,label=sample
        alpha=[]
        alpha.append(self.pi*np.array([self.b[i][self.vocab[data[0]]]
for i in range(4)]))
        for i in range(1,len(label)):
            alpha.append(np.matmul(alpha[i-
1],self.a)*np.array([self.b[ii][self.vocab[data[i]]] for ii in
range(4)]))
        return alpha

    def backward(self,sample):
        data,label=sample
        beta=[]
        beta.append(np.ones(4))
        for i in range(0,len(label)-1):
            beta.insert(0,(np.sum(beta[-1-
i]*np.array([self.b[ii][self.vocab[data[-i-1]]] for ii in
```

```
range(4)])*self.a,axis=1)))
        return beta

    def update(self,sample):
        flag=False
        data, label = sample
        alpha, beta = self.forward(sample), self.backward(sample)
        gamma = []
        dict_b = {}
        store=None
        ##求gamma 矩阵 T*c*c
        for i in range(0, len(label) - 1):
            nume = (alpha[i] * self.a.T).T * (
                    beta[i + 1] * np.array([self.b[ii][self.vocab[data[i +
1]]] for ii in range(4)]))
            gamma.append(np.divide(nume , np.sum(nume),
where=np.sum(nume) != 0))
        self.gamma = gamma
        ##求更新参数
        # 先求self.pi
        pi = np.sum(gamma[0], axis=1)
        # 求self.a
        nume_a = np.sum(np.array(gamma), axis=0)
        denom = np.sum(nume_a, axis=1)
        a = np.divide(nume_a.T, denom, where=denom != 0).T
        # 再求self.b
        for i in range(len(label) - 1):
            temp = np.sum(gamma[i], axis=1)
            try:
                dict_b[self.vocab[data[i]]] += np.divide(temp, denom,
where=denom != 0)
            except:
                dict_b[self.vocab[data[i]]] = np.divide(temp, denom,
where=denom != 0)
        b = self.b.T
        for key in dict_b.keys():
            b[key] = dict_b[key]
        b = b.T

p=Preprocsss("./隐马尔科夫分词任务/RenMinData.txt_utf8")
a,b,pi=p.process()
my_iter = [[p.data[i], p.label[i]] for i in range(len(p.data))]
hmm=HMM(a,b,pi,p.vocab)
'''
```

```
epoch=1
count=0
for i in range(epoch):
    random.shuffle(my_iter)
    for i in range(len(my_iter)//10000):
        temp_a,temp_b,temp_pi=hmm.update(my_iter[i])
        print(temp_a,temp_pi)
        hmm.a,hmm.b,hmm.pi=temp_a,temp_b,temp_pi'''

#print(hmm.a,hmm.pi)
sentences=['今天的天气很好。','学习模式识别课程是有难度的事情。','我是东南大学的学生。']
for sen in sentences:
    print(hmm(sen))
```

心得体会

在做实验一和二前，我以为不会难做，就像以前做物理实验一样，做完实验，然后两下子就将实验报告做完。直到做完测试实验时，我才知道其实并不容易做，实验和做题是两回事。在做实验前，一定要将课本上的知识吃透，因为这是做实验的基础，否则，化到时间反而会变多。就像本次我在做 HMM 作业时，就又花了很多时间整理前向后向的矩阵操作和原理分析。做实验时，一定要亲力亲为，务必要将每个步骤，每个细节弄清楚，弄明白，不能自欺欺人。

本次的代码实验，我也是尽可能的对代码进行了设计，使代码的健壮性，复用性尽可能的高，能建立对象类就尽量设计对象。使代码尽可能的简洁高效。同时，可视化的 plt 显示也可以让自己的成果更生动。

本次实验也让我的矩阵操作有了很大的提高，在本次实验里，我没有用嵌套循环的方式对参数进行更新，而是用了大量的矩阵操作来进行更新，使代码更简洁。

在做线性分类器的 LDA 实现时，我遇到了很多困难，准确率总是在 10%徘徊，后来进行数据的仔细分析，对自己写的线性层的初始化进行观察，终于意识到极低的准确率可能和数据的数量级以及线性层的初始化参数有关。最后终于是用 PCA+LDA 的方法实现。事实上，最优的的方案应该是针对不同的特征进行一个 rescale，再进行训练。或者是针对各个维度特征的数据的数量级，动态的生成权重。本次的 debug 也让我意识到冷静分析的重要性，平时方便易用的库都几乎都帮我们初始化完成了，我之前都没有考虑过如此问题。

本次实验拓宽了我的眼界，使我们认识到这门课程在是线上的困难，使我学到了不少实用的知识。更重要的是，做实验的过程，思考问题的方法，这与是通用的，真正使我受益匪浅。