

人工智能导论实验指导手册

北京邮电大学人工智能学院模式识别实验室

2020 年 9 月

目录

目录.....	2
一、引言.....	3
二、64 位 Windows 下进行软件环境安装	3
三、各实验说明.....	7
第一次实验：机器学习基础实验.....	7
实验编号：1-1	7
实验编号：1-2	10
第二次实验：神经网络基础实验.....	16
实验编号：2-1	16
实验编号：2-2	17
实验编号：2-3	19
第三次实验：计算机视觉基础实验.....	22
实验编号：3-1	22
实验编号：3-2	25
附录 A：实验报告模板.....	30

一、引言

《人工智能导论》课程安排三次实验。所有实验均使用 Python 语言进行程序设计和测试验证。

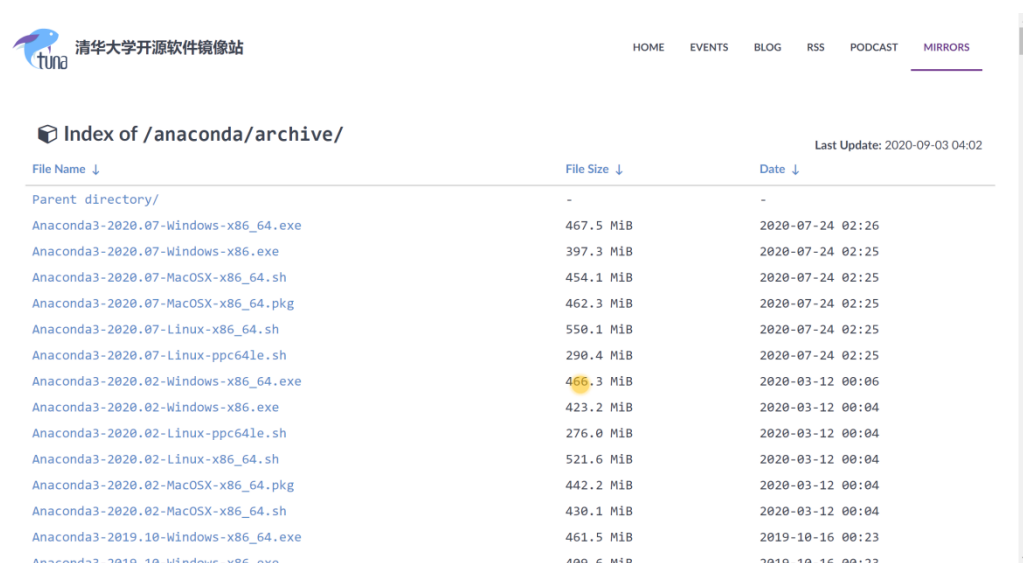
本指导手册将介绍在机房或个人电脑上如何进行 Python 环境及相关软件包的安装和配置，并重点介绍每次实验的实验目的，实验环境，实验内容，基础知识，实验步骤等内容，同学们可参考实验介绍独立完成相关实验。

二、64 位 Windows 下进行软件环境安装

本节介绍机房的 64 位 windows 下如何快速安装 conda, python, numpy, sklearn, pytorch 等。每次上机请确保进入 64 位 windows。

1、安装相应版本的 Anaconda（安装完成后，python，numpy，sklearn 都可用）

（1）访问 <https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/>



Index of /anaconda/archive/		
File Name ↓	File Size ↓	Date ↓
Parent directory/	-	-
Anaconda3-2020.07-Windows-x86_64.exe	467.5 MiB	2020-07-24 02:26
Anaconda3-2020.07-Windows-x86.exe	397.3 MiB	2020-07-24 02:25
Anaconda3-2020.07-MacOSX-x86_64.sh	454.1 MiB	2020-07-24 02:25
Anaconda3-2020.07-MacOSX-x86_64.pkg	462.3 MiB	2020-07-24 02:25
Anaconda3-2020.07-Linux-x86_64.sh	550.1 MiB	2020-07-24 02:25
Anaconda3-2020.07-Linux-ppc64le.sh	290.4 MiB	2020-07-24 02:25
Anaconda3-2020.02-Windows-x86_64.exe	466.3 MiB	2020-03-12 00:06
Anaconda3-2020.02-Windows-x86.exe	423.2 MiB	2020-03-12 00:04
Anaconda3-2020.02-Linux-ppc64le.sh	276.0 MiB	2020-03-12 00:04
Anaconda3-2020.02-Linux-x86_64.sh	521.6 MiB	2020-03-12 00:04
Anaconda3-2020.02-MacOSX-x86_64.pkg	442.2 MiB	2020-03-12 00:04
Anaconda3-2020.02-MacOSX-x86_64.sh	430.1 MiB	2020-03-12 00:04
Anaconda3-2019.10-Windows-x86_64.exe	461.5 MiB	2019-10-16 00:23
Anaconda3-2019.10-Windows-x86.exe	409.6 MiB	2019-10-16 00:23

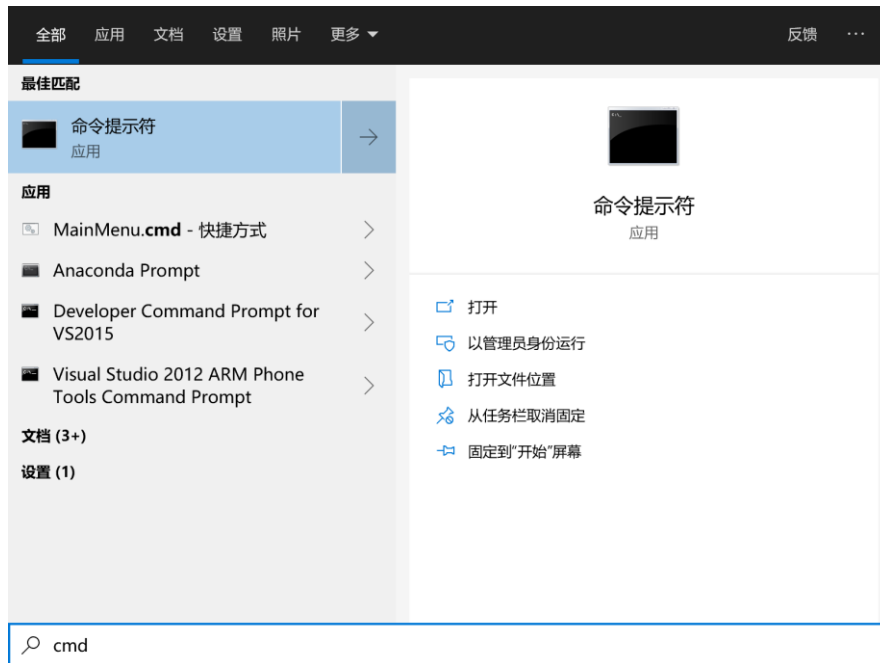
（2）点击下载相应安装程序。

选择下载 64 位安装程序，如：Anaconda3-2019.07-Windows-x86_64.exe。

（3）下载完成后，直接双击 exe 文件安装即可。

安装时，注意把 conda 路径放到 Windows 的 PATH 环境变量中。（此电脑-属性-高级系统设置-环境变量）

（4）安装完成后，运行 cmd，进入 Windows 下的命令行窗口。

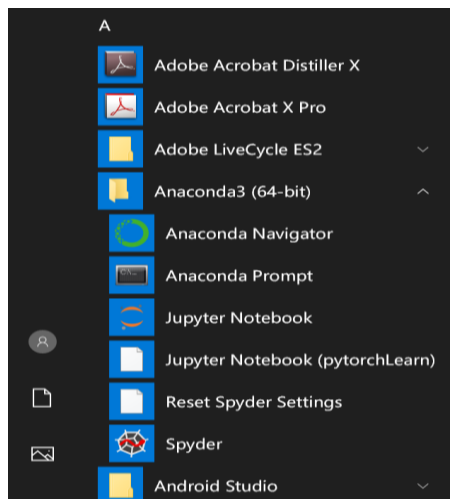


(5) 测试 conda 是否安装成功。

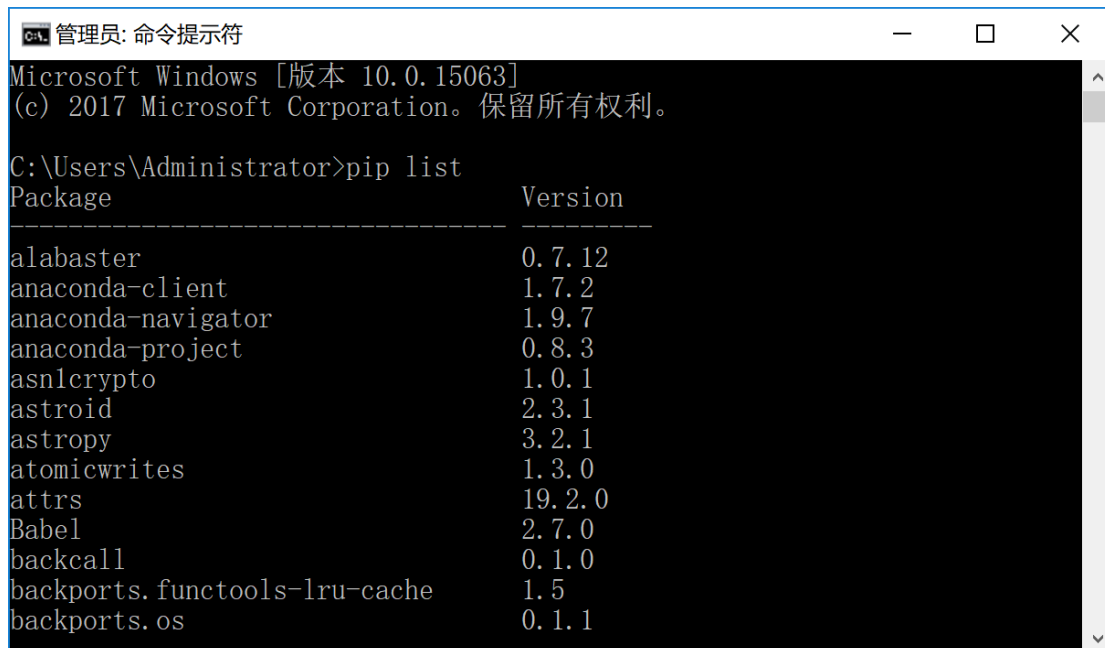
在命令行下运行 `conda activate base` 即进入 base 环境, 然后运行 `python`, 进入 python 环境:

```
管理员: 命令提示符 - python
(c) 2017 Microsoft Corporation。保留所有权利。
C:\Users\Administrator>conda activate base
(base) C:\Users\Administrator>python
Python 3.7.4 (default, Aug 9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)]
:: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 1+2
3
>>> _
```

上述界面表示 conda 安装成功。Windows 开始菜单中也有相关程序:

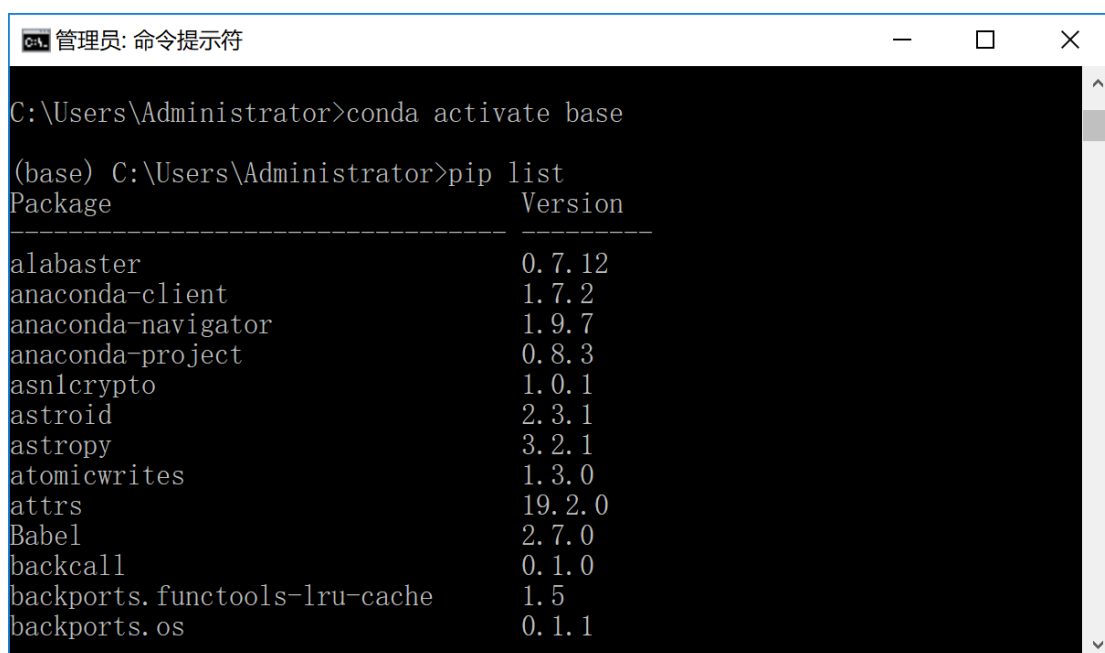


(6) 在 windows 命令行下 conda 的 base 环境中或运行 `pip list` 查看 conda 默认已经安装的包。可以发现 `numpy`, `sklearn` 等都已经安装好。



```
管理员: 命令提示符
Microsoft Windows [版本 10.0.15063]
(c) 2017 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>pip list
Package                                Version
-----
alabaster                              0.7.12
anaconda-client                        1.7.2
anaconda-navigator                    1.9.7
anaconda-project                      0.8.3
asn1crypto                            1.0.1
astroid                                2.3.1
astropy                               3.2.1
atomicwrites                          1.3.0
attrs                                  19.2.0
Babel                                  2.7.0
backcall                              0.1.0
backports.functools-lru-cache         1.5
backports.os                           0.1.1
```



```
管理员: 命令提示符

C:\Users\Administrator>conda activate base

(base) C:\Users\Administrator>pip list
Package                                Version
-----
alabaster                              0.7.12
anaconda-client                        1.7.2
anaconda-navigator                    1.9.7
anaconda-project                      0.8.3
asn1crypto                            1.0.1
astroid                                2.3.1
astropy                               3.2.1
atomicwrites                          1.3.0
attrs                                  19.2.0
Babel                                  2.7.0
backcall                              0.1.0
backports.functools-lru-cache         1.5
backports.os                           0.1.1
```

2. 在 conda 环境中安装 pytorch

(1) 修改配置文件".condarc"

首先运行: `conda config --set show_channel_urls yes`

该命令在 windows 用户目录下 (如 `C:\Users\administrator\`) 生成该文件, 打开该文件, 将如下内容进行替换文件内容:

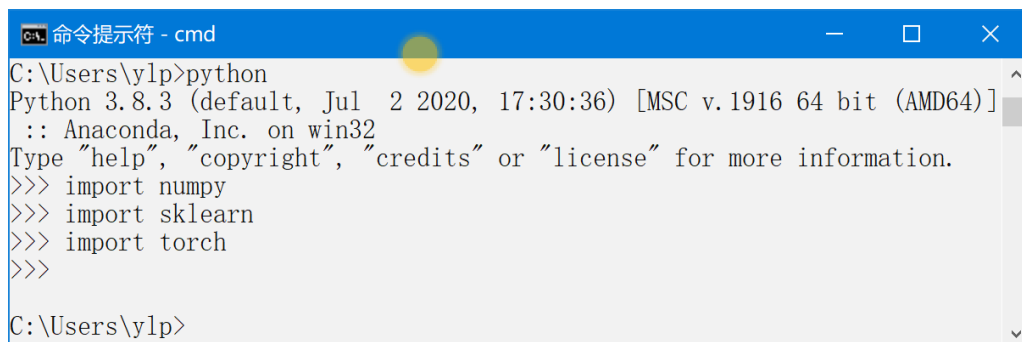
```
channels:
  - defaults
show_channel_urls: true
channel_alias: https://mirrors.tuna.tsinghua.edu.cn/anaconda
default_channels:
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/r
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/pro
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/msys2
custom_channels:
  pytorch: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
```

(2) 运行 cmd, 进入命令行, 运行 `conda install pytorch`

安装 CPU 版本的 pytorch。安装过程中根据提示, 输入相关选择, 如 yes。

若在自己带 GPU 的电脑上安装 pytorch, 请访问 pytorch.org, 根据说明进行安装。

(3) 检查是否可使用 python 下的 pytorch 框架。



```
命令提示符 - cmd
C:\Users\yyp>python
Python 3.8.3 (default, Jul 2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)]
:: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> import sklearn
>>> import torch
>>>
C:\Users\yyp>
```

(4) 若安装 torchvision, 则执行:

```
conda install torchvision -c pytorch
```

三、各实验说明

第一次实验：机器学习基础实验

实验编号：1-1

实验名称：线性回归

实验目的：

- 1、了解线性回归的基本原理；
- 2、掌握通过梯度下降方法实现最优解的求解。

实验内容：

实验环境：python, sklearn, numpy。

基础知识：

实验步骤：

- 1、生成数据。
- 2、定义画图函数、假设函数、损失函数、梯度计算函数、参数更新函数。
- 3、定义线性回归函数，并测试。

参考代码如下：

```
"""
本实验需要提前了解的知识：
1、线性回归
2、偏导数
"""

from sklearn.datasets import make_regression#导入 make_regression()函数，用来生成回归模型数据
import matplotlib.pyplot as plt#导入 matplotlib.pyplot，并且重命名为 plt
import numpy as np#导入 numpy 库，并且重命名为 np
"""

需要提前了解的知识：
1、线性回归
2、偏导数
"""

1、函数 make_regression(): 用来生成回归模型数据。
```

2、参数说明：

`n_samples`: 样本数
`n_features`: 特征数
`noise`: 噪音
`bias`: 偏差

3、`X` : array of shape `[n_samples, n_features]`

`y` : array of shape `[n_samples]` or `[n_samples, n_targets]`

4、下面的语句的作用为:生成一组数据集 $\{(x_1, y_1), (x_2, y_2), \dots, (x_{100}, y_{100})\}$ ，后面我们将学习一个线性模型来尽可能的拟合此数据集。

```
"""
```

```
X, y = make_regression(n_samples=100, n_features=1, noise=0.4, bias=50)
```

```
"""
```

1、定义一个名为 `plotLine()` 的函数，用来画出生成数据集的散点图和拟合线性模型 ($y=k*x+b$)

2、参数说明：

`theta0`: 即 $y=k*x+b$ 中的参数 `b`
`theta1`: 即 $y=k*x+b$ 中的参数 `k`
`X`: 数据集的横坐标（列表类型）
`y`: 数据集的纵坐标（列表类型）

3、`np.linspace(start, stop, num)` 函数: 用来返回 `num` 个等间距的样本，在区间 `[start, stop]` 中。

4、`plt.plot(x, y, color, label)`: 可视化函数

参数说明: `x`: `x` 轴上的数值; `y`: `y` 轴上的数值; `color`: 用来设置线条的颜色, `color='r'` 表示红色(`b` 表示蓝色); `label` 用于指定标签

5、`plt.scatter(x, y)`: 用来画散点图。

参数说明: `x`: `x` 轴上的数值; `y`: `y` 轴上的数值。

6、`plt.axis()` 函数用来指定坐标轴的范围。

参数需要以列表的形式给出。

7、`plt.show()`: 将图像显示出。

```
"""
```

```
def plotLine(theta0, theta1, X, y):
```

```
    max_x = np.max(X) + 100    #np.max(X)用来取出 X 中的最大值
```

```
    min_x = np.min(X) - 100    #np.min(X)用来取出 X 中的最小值
```

```
    xplot = np.linspace(min_x, max_x, 1000)    #在区间[min_x, max_x]中返回 1000
```

个等间隔的样本

```
    yplot = theta0 + theta1 * xplot    #将 x 带入线性方程 y=k*x+b 中求得 y
```

```
    print("目前的参数 b=", theta0)    #打印参数 theta0
```

```
    print("目前的参数 k=", theta1)    #打印参数 theta1
```

```
    plt.plot(xplot, yplot, color='g', label='Regression Line')    #画出线性
```

模型，参数依次表示: 横坐标，纵坐标，颜色，标签

```
    plt.scatter(X, y)    #画散点图，参数依次表示横坐标、纵坐标
```

```
    plt.axis([-10, 10, 0, 200])    #设置横坐标范围为【-10, 10】，纵轴范围为【0,
```

200】

```
    plt.show()    #显示可视化图像
```


"""

1、定义一个名为 `hypothesis()` 的函数,根据给定的 `x` 值预测 `y` 的值,计算公式为: $y = \theta_0 + (\theta_1 * x)$

"""

```
def hypothesis(theta0, theta1, x):  
    return theta0 + (theta1*x)
```

"""

1、定义一个计算损失值的函数,采用最小二乘法来计算损失。

2、`zip(x,y)` 函数用于将可迭代的对象作为参数,将对象中对应的元素打包成一个个元组,然后返回由这些元组组成的列表。

譬如: `x={x1,x2,x3};y={y1,y2,y3}`; 则 `zip(x,y)=[(x1,y1),(x2,y2),(x3,y3)]`

3、`y**2`:表示 `y` 的平方。

"""

```
def cost(theta0, theta1, X, y):    #计算损失  
    costValue = 0  
    for (xi, yi) in zip(X, y):      #使用 zip()函数, 包为元组的列表  
        costValue += 0.5 * ((hypothesis(theta0, theta1, xi) - yi)**2) #使  
    用最小二乘法来计算损失  
    return costValue               #返回损失值
```

"""

1、定义名为 `derivatives()` 的函数,用来计算参数的梯度。

2、`len()` 函数: 用来返回对象(字符、列表、元组等)长度或项目个数。其参数可以是字符、列表、元组等。

"""

```
def derivatives(theta0, theta1, X, y):    #derivative:导数  
    dtheta0 = 0        #dtheta0: 参数 theta0 的梯度, 初始化为 0  
    dtheta1 = 0        #dtheta1: 参数 theta1 的梯度, 初始化为 0  
    for (xi, yi) in zip(X, y): #使用 zip()函数依次取出(xi,yi)  
        dtheta0 += hypothesis(theta0, theta1, xi) - yi    #计算公式为: 损失  
        函数对参数 dtheta0 求偏导。  
        dtheta1 += (hypothesis(theta0, theta1, xi) - yi)*xi    #计算公式为:  
        损失函数对参数 dtheta1 求偏导。  
    dtheta0 /= len(X)    #求平均梯度, len(X)函数用来计算 X 中的样本数  
    dtheta1 /= len(X)    #求平均梯度  
    return dtheta0, dtheta1
```

"""

1、定义一个名为 `updateParameters()` 的函数,用来对参数进行更新。

参数说明:

`theta0` 和 `theta1` 为待更新参数。

x 、 y 分别表示横轴和纵轴的数值。

α : 学习率。

2、参数的更新:

对于参数 w , 其更新方式为: $w=w-\text{学习率} \times \text{梯度值}$ 。其中学习率是一个超参数。

```
"""
def updateParameters(theta0, theta1, X, y, alpha):    #参数的更新, alpha 表示学习率
    dtheta0, dtheta1 = derivatives(theta0, theta1, X, y) #dtheta0, dtheta1 分别表示参数 theta0, theta1 的梯度值。
    theta0 = theta0 - (alpha * dtheta0)    #依据参数更新方式更新参数 theta0
    theta1 = theta1 - (alpha * dtheta1)    #依据参数更新方式更新参数 theta1
    return theta0, theta1    #返回更新好的参数
"""
```

1、定义一个名为 LinearRegression()的线性回归函数。

参数说明:

x : 表示给定数据集的横坐标。

y : 表示给定数据集的纵坐标。

2、np.random.rand()函数: 用来返回一个或一组服从“0~1”均匀分布的随机样本值。随机样本取值范围是[0,1),

不包括 1。当不给定参数时, 返回的是一个[0, 1)区间内的随机数。)

```
"""
def LinearRegression(X, y):
    theta0 = np.random.rand()    #给 theta0 赋一个随机初始值。
    theta1 = np.random.rand()    #给 theta1 赋一个随机初始值。
    for i in range(0, 1000):    #进行 1000 次参数的更新, 每隔 100 次更新打印一次
        图片
        if i % 100 == 0:    #只有当 i 整除 100 时才进行一次图片打印
            plotLine(theta0, theta1, X, y)
            #print(cost(theta0, theta1, X, y))
            theta0, theta1 = updateParameters(theta0, theta1, X, y, 0.005)    #
            调用参数更新函数来对参数进行更新, 其中学习率指定为: 0.005.

LinearRegression(X, y)    #调用线性回归函数。
"""
```

实验编号: 1-2

实验名称: K-means 聚类。

实验目的:

1、了解 K-means 聚类的基本原理;

2、编写程序实现 K-means 聚类方法。

实验内容：

实验环境：python, sklearn, numpy。

基础知识：

实验步骤：

- 1、定义欧式距离函数、类中心选取函数、聚类函数、画图函数。
- 2、读取待聚类数据完成聚类，并画图观察结果。

参考代码如下：

```
from numpy import *
import time
import matplotlib.pyplot as plt

"""
#注：因为采用了“from numpy import *”语句引入 numpy 库中的函数，因此在用到 numpy
库中的函数时直接写函数名，前面不用加 numpy。
1、定义一个名为 euclDistance()的函数，用来计算两个矩阵之间的欧式距离。其中参数
vector1, vector2 分别表示两个矩阵。
2、np.sqrt(x)函数用来计算 x 的开方。
3、np.power(array,m):表示对 array 中的每个元素求它的 m 次方,譬
如:np.power([0,1,2,3],2)=[0,1,4,9]。
4、np.sum(array)表示将 array 中的每个元素相加求和，譬如：
np.sum([0,1,2])=3,np.sum([[0,1,2],[0,1,2]])=6。
"""

# calculate Euclidean distance
def euclDistance(vector1, vector2):
    return sqrt(sum(power(vector2 - vector1, 2))) #求这两个矩阵的距离，
vector1、2 均为矩阵

"""
1、定义一个名为 initCentroids()的函数，用来在样本集中随机选取 k 个样本点作为初始
质心，
    其中参数 dataSet 为已给数据集，k 表示创建中心点的个数。返回值为所创建的 k 个中
心点
2、np.zeros([k, n]): 用来创建一个 k 行 n 列的全 0 数组。
3、np.random.uniform(a,b):返回区间[a,b)中的任意值。
"""

# init centroids with random samples
#在样本集中随机选取 k 个样本点作为初始质心。
def initCentroids(dataSet, k):
    numSamples, dim = dataSet.shape #矩阵的行数、列数 。
    centroids = zeros((k, dim)) # 创建一个 k 行 dim 列的全 0 数组。
```

```

    for i in range(k):
        index = int(random.uniform(0, numSamples)) #随机产生一个浮点数，然
        后将其转化为 int 型。
        centroids[i, :] = dataSet[index, :] #将 dataSet 中第 index+1 行赋
        值给 centroids 的第 i+1 行。
    return centroids

```

"""

- 1、定义一个名为 `kmeans()` 的聚类算法，用于将 `dataSet` 矩阵中的样本分成 `k` 个类。其中参数 `dataSet` 为一个矩阵，参数 `k` 表示将分为 `k` 类。
- 2、`np.mat(a)`: 用于将数组 `a` 转换为矩阵。
- 3、`np.zeros([k, n])`: 用来创建一个 `k` 行 `n` 列的全 0 数组。
- 4、`matrix.A`: 将矩阵类型转换为 `array` 类型。
- 5、`np.nonzero(array)`: 用于得到数组 `array` 中非零元素的位置（数组索引），参数 `array` 为一个数组。
- 6、`np.mean()`: 求均值。经常操作的参数为 `axis`，以 `m * n` 矩阵举例：
 - `axis` 不设置值，对 `m*n` 个数求均值，返回一个实数；
 - `axis = 0`: 压缩行，对各列求均值，返回 `1* n` 矩阵；
 - `axis =1` : 压缩列，对各行求均值，返回 `m *1` 矩阵。
- 7、`plt.plot(x,y,color,marksize)`: 当使用此函数画一个数据点时，参数 `x` 表示横坐标，参数 `y` 表示纵坐标，参数 `color` 用来指定点的颜色，参数 `marksize` 用来指示点的大小。

"""

```

# k-means cluster
#dataSet 为一个矩阵
#k 为将 dataSet 矩阵中的样本分成 k 个类
def kmeans(dataSet, k):
    numSamples = dataSet.shape[0] #读取矩阵 dataSet 的第一维度的长度,即获得
    有多少个样本数据
    clusterAssment = mat(zeros((numSamples, 2))) #得到一个 N*2 的零矩阵,建
    立簇分配结果矩阵，第一列存类别，第二列存误差。
    clusterChanged = True #用来判断样本聚类结果是否变化的变量。

    ## step 1: init centroids
    centroids = initCentroids(dataSet, k) #在样本集中随机选取 k 个样本点作为
    初始质心

    while clusterChanged:
        clusterChanged = False
        ## for each sample
        for i in range(numSamples): #range
            minDist = 100000.0 #创建的一个临时变量，用来储存某个样本到所有聚
            类中心的最小距离。

```

```

        minIndex = 0      #创建的一个临时变量，用来储存和某个样本距离最近的
聚类中心的类别作为该样本的类别。
        ## for each centroid
        ## step 2: find the centroid who is closest
        #计算每个样本点与质点之间的距离，将其归内到距离最小的那一簇
        for j in range(k):
            distance = euclDistance(centroids[j, :], dataSet[i, :]) #
计算每个样本到每个聚类中心之间的距离。
            if distance < minDist:
                minDist = distance
                minIndex = j
        ## step 3: update its cluster
        #k 个簇里面与第 i 个样本距离最小的的标号和距离保存在 clusterAssment
中
        #若所有的样本所属类别不在变化，则退出 while 循环
        if clusterAssment[i, 0] != minIndex:
            clusterChanged = True
            clusterAssment[i, :] = minIndex, minDist**2 #两个**表示的
是 minDist 的平方
        ## step 4: update centroids
        for j in range(k):
            #clusterAssment[:,0].A==j 是找出矩阵 clusterAssment 中第一列元素
中等于 j 的行的下标，返回的是一个以 array 的列表，第一个 array 为等于 j 的下标
            pointsInCluster = dataSet[nonzero(clusterAssment[:, 0].A ==
j)[0]] #将 dataSet 矩阵中相对应的样本提取出来
            centroids[j, :] = mean(pointsInCluster, axis = 0) #计算标注为 j
的所有样本的平均值
            print ('Congratulations, cluster complete!')
        return centroids, clusterAssment

# show your cluster only available with 2-D data
#centroids 为 k 个类别，其中保存着每个类别的质心
#clusterAssment 为样本的标记，第一列为此样本的类别号，第二列为到此类别质心的距
离
def showCluster(dataSet, k, centroids, clusterAssment):
    numSamples, dim = dataSet.shape
    if dim != 2:
        print ("Sorry! I can not draw because the dimension of your data is
not 2!")
        return 1
    mark = ['or', 'ob', 'og', 'ok', '^r', '+r', 'sr', 'dr', '<r', 'pr'] #
样本颜色
    if k > len(mark):
        print ("Sorry! Your k is too large!")

```

```

        return 1
    # draw all samples
    for i in range(numSamples):
        markIndex = int(clusterAssment[i, 0]) #为样本指定颜色
        plt.plot(dataSet[i, 0], dataSet[i, 1], mark[markIndex]) #画出样本
    mark = ['Dr', 'Db', 'Dg', 'Dk', '^b', '+b', 'sb', 'db', '<b', 'pb'] #
    中心的颜色
    # draw the centroids
    for i in range(k):
        plt.plot(centroids[i, 0], centroids[i, 1], mark[i], markersize = 12)
#画出中心点
    plt.show() #显示图片

```

"""

1、".txt"文件的读取:

`f=open(file_path)` #其中 `f` 叫做文件句柄, `file_path` 为文件所在的路径。

`f.readlines()`函数用来读取文件中的全部内容, 返回值为一个列表, 列表中的每个元素为每行对应的内容。

`f.close()`用来关闭所打开的文件。

2、`.strip()`方法用于移除字符串头尾指定的字符(默认为空格或换行符)。

3、`.split(str)`方法通过指定分隔符对字符串进行切片, 其中参数 `str` 为分隔符, 返回值为一个列表。

4、`.append(obj)`方法用于在列表末尾添加 `obj`。

5、`float(a)`表示将 `a` 转化为 `float` 类型。

"""

step 1: 载入待聚类数据

```
print("step 1: load data...")
```

`dataSet = []` #列表, 用来表示, 列表中的每个元素也是一个二维的列表; 这个二维列表就是一个样本, 样本中包含有我们的属性值和类别号。

#与我们所熟悉的矩阵类似, 最终我们将获得 $N \times 2$ 的矩阵, 每行元素构成了我们的训练样本的属性值和类别号

`fileIn = open("D:/testdata.txt")` #`"D:/testdata.txt"`为数据文件所在位置的绝对路径。

```
for line in fileIn.readlines(): #依次遍历每一行
```

```
    temp=[] #定义一个缓存列表
```

`lineArr = line.strip().split('\t')` #`line.strip()`把末尾的'\n'去掉, `.split('\t')`表示以'\t'为分隔符将字符串切片。

```
    temp.append(float(lineArr[0])) #float(a)表示将 a 转化为 float 类型。
```

```
    temp.append(float(lineArr[1]))
```

```
    dataSet.append(temp) #向 dataSet 列表中添加元素。
```

```
fileIn.close() #关闭刚刚打开的 testdata.txt 文件。
```

```
## step 2: 聚类中...
print ("step 2: clustering..." )
dataSet = mat(dataSet) #mat()函数是 Numpy 中的库函数，将数组转化为矩阵
k = 4
centroids, clusterAssment = kmeans(dataSet, k) #调用 KMeans 文件中定义的
kmeans 方法。

## step 3: 画图展示结果
print ("step 3: show the result..." )
showCluster(dataSet, k, centroids, clusterAssment)
```

第二次实验：神经网络基础实验

实验编号：2-1

实验名称：基于 numpy 构建简单的三层回归神经网络。

实验目的：

- 1、了解反向传播网络的基本原理；
- 2、了解梯度下降法进行神经网络中的权值更新；
- 3、学习使用 numpy 编写简单的三层回归网络进行回归实验。

实验内容：

使用 numpy 库构建简单的数据集，编写简单的三层回归神经网络，输入和输出只有一个神经元，中间隐藏层可设置 N 个神经元，采用 sigmoid 函数作为激活函数。数据集按 $y=x+$ 随机噪声进行构建，x 取值为 0, 1, ..., 19。学习梯度计算，及梯度下降法进行神经网络的权值修改。

实验环境：python, numpy

基础知识：

实验步骤：

(1) 确保 python 环境已经搭建完成。

(2) 编写代码，参考代码如下：

```
import numpy as np
import matplotlib.pyplot as plt
def sigmoid_derivative(s):
    ds = s * (1 - s)
    return ds
def sigmoid(x):
    s = 1 / (1 + np.exp(-x))
    return s
# N: batch_size; D_in: 输入维度
# H: 隐藏层的维度; D_out: 输出维度
N, D_in, H, D_out = 20, 1, 64, 1
# 随机生成一批数据
np.random.seed(0)
x = np.arange(0,N,1).reshape(N,D_in)*1.0 #20*1
y = x + np.random.randn(N,D_out)        #20*1
# 随机初始化权重
```



```

w1 = np.random.randn(D_in, H)          #1*64
w2 = np.random.randn(H, D_out)          #64*1
# 定义学习率 learning_rate
learning_rate = 1e-3
for t in range(20000):
    # 进行前向传播
    h = x.dot(w1)
    h_relu = sigmoid(h)
    y_pred = h_relu.dot(w2)

    # 计算损失函数
    loss = np.square(y_pred - y).sum()

    # 进行反向传播
    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = h_relu.T.dot(grad_y_pred)
    grad_h = grad_y_pred.dot(w2.T)      #[N, H]=[N, 1]*[1, H]
    grad_h = grad_h*sigmoid_derivative(h_relu)  #[N, H]=[N, H] . [N, H]]
    grad_w1 = x.T.dot(grad_h)           #[1, H]=[1, N]*{N, H}
    # 更新权重
    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2
    if (t%1000==0):
        plt.cla()
        plt.scatter(x,y)
        plt.scatter(x,y_pred)
        plt.plot(x,y_pred,'r-',lw=1, label="plot figure")
        plt.text(5.0, 2.5, 't=%d:Loss=%.4f' % (t, loss), fontdict={'size':
20, 'color': 'red'})
        plt.show()

```

(3) 运行代码，观察效果。

(4) 修改不同参数，如学习率，循环次数，回归函数修改（如 $y=x^2$ ），观察效果。

(5) 思考：代码中的激活函数采用的是 sigmoid 函数，试修改为 ReLU 函数，观察效果。

实验编号：2-2

实验名称：基于 pytorch 构建简单的三层回归神经网络。

实验目的：

1. 了解 pytorch 下采用 adam 梯度下降法进行神经网络中的权值更新；
- 3、学习使用 pytorch 编写简单的三层神经网络进行回归实验。

实验内容：

使用 numpy 库构建简单的数据集，编写简单的三层回归神经网络，输入和输出只有一个神经元，中间隐藏层可设置 N 个神经元，采用 Sigmoid 函数作为激活函数。数据集按 $y=x+$ 随机噪声进行构建，x 取值为 0, 1, ..., 19。了解 pytorch 中的梯度计算，及梯度下降法进行神经网络的权值更新。

实验环境：python, numpy, pytorch

基础知识：

实验步骤：

(1) 确保 python 环境已经搭建完成。

(2) 编写代码，参考代码如下：

```
import torch
import numpy as np
import matplotlib.pyplot as plt

# N: batch_size; D_in: 输入维度;
# H: 隐藏层维度; D_out: 输出维度;
N, D_in, H, D_out = 20, 1, 64, 1

# 随机生成一批数据
np.random.seed(0)
x = torch.tensor(np.arange(0,N,1).reshape(N,D_in),dtype=torch.float32)
#20*1
y = x +torch.tensor(np.random.randn(N,D_out), dtype=torch.float32)
#20*1

# 定义网络结构与损失函数
model = torch.nn.Sequential(
    torch.nn.Linear(D_in, H),
    torch.nn.ReLU(),
    torch.nn.Linear(H, D_out),
)
loss_fn = torch.nn.MSELoss(reduction='sum')

#定义优化器，这里使用 Adam
learning_rate = 1e-3
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
for t in range(5000):
    # 进行前向传播
    y_pred = model(x)
```

```

# 计算损失函数
loss = loss_fn(y_pred, y)
if t % 200 == 0:
    plt.cla()
    plt.scatter(x.data.numpy(),y.data.numpy())
    plt.scatter(x.data.numpy(),y_pred.data.numpy())
    plt.plot(x.data.numpy(),y_pred.data.numpy(),'r-',lw=1,
label="plot figure")
    plt.text(0.5, 0, 't=%d:Loss=%.4f' % (t, loss), fontdict={'size': 20,
'color': 'red'})
    plt.show()
#将梯度清零（具体可参考 PyTorch 官方文档）
optimizer.zero_grad()
#进行反向传播
loss.backward()
#更新所有参数
optimizer.step()

```

- (3) 运行代码，观察效果。
- (4) 修改不同参数，如学习率，循环次数，回归函数修改（如 $y=x^2$ ），观察效果。
- (5) 思考：代码中的激活函数采用的是 ReLU 函数，试修改为 Sigmoid 函数，观察效果。

实验编号：2-3

实验名称：基于 pytorch 构建简单的三层分类神经网络。

实验目的：

2. 了解 pytorch 下采用 adam 梯度下降法进行神经网络中的权值更新；
- 3、学习使用 pytorch 编写简单的三层神经网络进行分类实验。

实验内容：

使用库构建简单的二分类数据集，编写简单的三层分类神经网络，输入和输出均有两个神经元，中间隐藏层可设置 N 个神经元，采用 Relu 函数作为激活函数。数据集按两个类构建，每个类的样本为两维，分散在类中心附近。学习梯度计算，及梯度下降法进行神经网络的权值修改。

实验环境：python, numpy, pytorch。

基础知识:

实验步骤:

(1) 确保 python 环境已经搭建完成。

(2) 编写代码, 参考代码如下:

```
import torch
import torch.nn.functional as F
import matplotlib.pyplot as plt

# 生成随机数据
n_data = torch.ones(100, 2)
x0 = torch.normal(2*n_data, 1)      # class0_x data
y0 = torch.zeros(100)                # class0_y data
x1 = torch.normal(-2*n_data, 1)      # class1_x data
y1 = torch.ones(100)                 # class1_y data
x = torch.cat((x0, x1), 0).type(torch.FloatTensor)
y = torch.cat((y0, y1), 0).type(torch.LongTensor)

class Net(torch.nn.Module):
    def __init__(self, n_feature, n_hidden, n_output):
        super(Net, self).__init__()
        self.hidden = torch.nn.Linear(n_feature, n_hidden) # 隐藏层
        self.out = torch.nn.Linear(n_hidden, n_output) # 输出层
    def forward(self, x):
        x = torch.sigmoid(self.hidden(x))      # 激活函数
        x = self.out(x)
        return x

#定义网络、优化器与损失函数
net = Net(n_feature=2, n_hidden=10, n_output=2)
print(net) # net architecture
optimizer = torch.optim.SGD(net.parameters(), lr=0.02)
loss_func = torch.nn.CrossEntropyLoss() # the target label is NOT an
one-hotted

#可视化
plt.ion()
for t in range(100):
    out = net(x)      # 进行前向传播
    loss = loss_func(out, y) # 计算损失函数
    optimizer.zero_grad() # 将梯度清零
    loss.backward()     # 进行反向传播
    optimizer.step()    # 更新所有参数
```

```
if t % 2 == 0:
    plt.cla()
    prediction = torch.max(out, 1)[1]
    pred_y = prediction.data.numpy()
    target_y = y.data.numpy()
    plt.scatter(x.data.numpy()[:, 0], x.data.numpy()[:, 1], c=pred_y,
s=100, lw=0, cmap='RdYlGn')
    accuracy = float((pred_y == target_y).astype(int).sum()) /
float(target_y.size)
    plt.text(0.5, -4, 'Accuracy=%.2f' % accuracy, fontdict={'size': 20,
'color': 'red'})
    plt.pause(0.1)
plt.ioff()
plt.show()
```

- (3) 运行代码，观察效果。
- (4) 修改不同参数，如学习率，循环次数，观察效果。
- (5) 思考：代码中的激活函数采用的是 Sigmoid 函数，试修改为 ReLU 函数，观察效果。

第三次实验：计算机视觉基础实验

实验编号：3-1

实验名称：验证码识别

实验目的：

- 1、熟悉图像增强、图像分割等图像处理的应用；
- 2、对验证码进行识别，熟悉图像识别的过程。

实验内容：

本实验针对图像处理和识别技术进行 python 的编程实现，实现对图像进行编辑、处理、增强和识别，让学生对人工智能技术中的图像识别技术建立直观的了解。（1）图像处理基本操作，如：图像编辑、图像增强、图像直方图变换；（2）图像识别相关操作，如：图像特征提取；（3）对验证码和 OCR 技术进行较为深入的理解。

实验环境：python，PIL，pytesseract，tesseact-ocr（安装后需添加环境变量）

基础知识：图像预处理、图像增强、图像分割、图像识别

实验步骤：

1. 确保 python 环境已经搭建完成。
2. 熟悉 PIL 库，进行图像编辑操作，实现图像的打开、旋转、裁剪、保存等。
3. 进行图像增强的技术实现，可以实现图像平滑、滤波、图像边缘提取等。
4. 实现基于直方图的图像分割，查看阈值对分割结果的影响。
5. 使用 Tesseract 识别库进行验证码的识别，统计不同类型验证码的识别效果。

```
from PIL import Image
#图像裁剪
im=Image.open("d:\\test.jpg")
im_L = im.convert("L")
box = (560,1000,1800,1800)
region = im_L.crop(box)
region.save("d:\\crop_img.jpg")
region.show()
#图像合并，此处 1.jpg 与 2.jpg 所有通道必须有相同的尺寸
im1 = Image.open("d:\\1.jpg")
im2 = Image.open("d:\\1.jpg")
r1,g1,b1 = im1.split()
r2,g2,b2 = im2.split()
```

```

print(r1.mode,r1.size,g1.mode,g1.size)
print(r2.mode,r2.size,g2.mode,g2.size)
new_im=[r1,g2,b2]
print(len(new_im))
im_merge = Image.merge("RGB",new_im)
im_merge.show()

from PIL import Image
from pylab import *
# 读取图像到数组中
im = array(Image.open('d:\\test.jpg'))
# 绘制图像
imshow(im)
# 一些点
x = [100,100,400,400]
y = [200,500,200,500]
# 使用红色星状标记绘制点
plot(x,y,'r*')
# 绘制连接前两个点的线
plot(x[:2],y[:2])
# 添加标题，显示绘制的图像
title('Plotting: "empire.jpg"')
show()

from PIL import Image
from PIL import ImageEnhance
#原始图像
image = Image.open("d:\\1.jpg")
image.show()

#亮度增强
enh_bri = ImageEnhance.Brightness(image)
brightness = 1.5
image_brightened = enh_bri.enhance(brightness)
image_brightened.show()

#对比度增强
from PIL import Image
from PIL import ImageEnhance
from PIL import ImageFilter

#原始图像
image = Image.open("d:\\1.jpg")
image.show()

```

#对比度增强

```
enh_con = ImageEnhance.Contrast(image)
contrast = 1.5
image_contrasted = enh_con.enhance(contrast)
image_contrasted.show()
```

#锐度增强

```
enh_sha = ImageEnhance.Sharpness(image)
sharpness = 3.0
image_sharped = enh_sha.enhance(sharpness)
image_sharped.show()
```

#图像模糊

```
im = Image.open("d:\\1.jpg")
im_blur = im.filter(ImageFilter.BLUR)
im_blur.show()
```

#轮廓提取

```
im = Image.open("d:\\1.jpg")
im_contour = im.filter(ImageFilter.CONTOUR)
im_contour.show()
```

#画直方图

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
img=np.array(Image.open('d:/test.jpg').convert('L'))
plt.figure("lena")
arr=img.flatten()
n, bins, patches = plt.hist(arr, bins=256, density=1, facecolor='green',
alpha=0.75)
plt.show()
```

图像分割

图片二值化

```
from PIL import Image
img = Image.open('d:\\test.jpg')
```

模式 L”为灰色图像，它的每个像素用 8 个 bit 表示，0 表示黑，255 表示白，其他数字表示不同的灰度。

```
Img = img.convert('L')
Img.save("d:\\test1.jpg")
```

自定义灰度界限，大于这个值为黑色，小于这个值为白色

```
threshold = 200
```



```

table = []
for i in range(256):
    if i < threshold:
        table.append(0)
    else:
        table.append(1)

# 图片二值化
photo = Img.point(table, '1')
photo.show()

#验证码识别
import pytesseract
from PIL import Image
#1.引入 Tesseract 程序
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files
(x86)\Tesseract-OCR\tesseract.exe'
#2.使用 Image 模块下的 Open()函数打开图片
image = Image.open('d:\\6.jpg',mode='r')
print(image)
#3.识别图片文字
code= pytesseract.image_to_string(image)
print(code)

```

实验编号：3-2

实验名称：图像检索

实验目的：

- 1、了解手工提取特征的目的及过程；
- 2、掌握图像匹配与检索的基本原理。

实验内容：

通过使用不同特征提取算法、图像匹配算法以及图像距离衡量方法的方案组合，进而理解各个算法对图像检索结果的影响。（1）使用 BRISK, AKAZE, KAZE, ORB 四个手工特征，提取自然场景下的图像信息，具体包括关键点检测，以及特征向量构造。根据所得特征向量进行图像匹配与检索。（2）所提供图像匹配有：暴力法（或基于 K 近邻匹配法，以及近似

K 近邻匹配法)。(3) 检索时, 所用衡量图像距离的方法有: chebyshev -切比雪夫距, cityblock-街区距, cosine-余弦夹角, mahalanobis-马氏距离, minkowski-闵可夫斯基距, euclidean-欧式距, hamming-汉明距离。

实验环境: python, numpy, opencv-python, scipy, matplotlib

基础知识: 图像检索流程, 特征提取的基本含义, 图像匹配与检索的基本思想, 图像距离。

实验步骤:

1. 确保 python 环境已经搭建完成。
2. 编写代码, 如下所示; 数据准备, 可使用已提供数据, 也可自己上传数据。

```
import cv2
import numpy as np
import scipy
import scipy.spatial
from imageio import imread
import pickle as pk
import random
import os
import matplotlib.pyplot as plt

# 特征提取
def extract_features(image_path, vector_size=32):
    image = imread(image_path)
    try:
        # 可选的特征检测算法有 BRISK,AKAZE,KAZE 以及 ORB
        # 关键点检测
        alg = cv2.BRISK_create()
        # alg = cv2.AKAZE_create()
        # alg = cv2.KAZE_create()
        # alg = cv2.ORB_create()
        kps = alg.detect(image)
        # 选取前 vector_size=32 个特征点
        # 特征点的个数取决于图像的大小以及颜色分布
        # 按照关键点响应值对特征点进行排序

        kps = sorted(kps, key=lambda x: -x.response)[:vector_size]
        # 计算特征点上对应的特征向量
        kps, dsc = alg.compute(image, kps)
        # 将所有特征向量组成一个大的特征值
        dsc = dsc.flatten()
```

```

    # 预定义一个维度为 64*vector_size 的特征向量
    needed_size = (vector_size * 64)
    if dsc.size < needed_size:
        # 如果计算得到的特征向量小于预定义的大小，则在向量末尾补零
        dsc = np.concatenate([dsc, np.zeros(needed_size - dsc.size)])
except cv2.error as e:
    print ('Error: ', e)
    return None
return dsc

def batch_extractor(images_path,
pickled_db_path="D:/output/features.pck"):
    files = [os.path.join(images_path, p) for p in
sorted(os.listdir(images_path))]
    result = {}
    for f in files:
        print ('Extracting features from image %s' % f)
        name = f.split('/')[1].lower()
        result[name] = extract_features(f)
    # 将所有特征保存在 pickle 文件里
    with open(pickled_db_path, 'wb+') as fp:
        pk.dump(result, fp)

class Matcher(object):
    def __init__(self, pickled_db_path="D:/output/features.pck"):
        with open(pickled_db_path, 'rb+') as fp:
            self.data = pk.load(fp)
            self.names = []
            self.matrix = []
            for k, v in self.data.items():
                self.names.append(k)
                self.matrix.append(v)
            self.matrix = np.array(self.matrix)
            self.names = np.array(self.names)

    def cdist(self, vector):
        # 计算图像之间的 cosine 距离
        v = vector.reshape(1, -1)
        return scipy.spatial.distance.cdist(self.matrix, v,
'cosine').reshape(-1)
        # 可选距离
        # chebyshev: 切比雪夫距离
        # cityblock 街区距离

```

```

# cosine: 余弦夹角
# mahalanobis: 马氏距离
# minkowski: 闵可夫斯基距离
# euclidean: 欧式距离
# hamming: 汉明距离

def match(self, image_path, topn=5):
    features = extract_features(image_path)
    img_distances = self.cdist(features)
    # 找到排名前 5 的匹配结果
    nearest_ids = np.argsort(img_distances)[:topn].tolist()
    nearest_img_paths = self.names[nearest_ids].tolist()

    return nearest_img_paths, img_distances[nearest_ids].tolist()

def show_img(path):
    img = imread(path)
    plt.imshow(img)
    plt.show()

def run():
    images_path = 'D:/images/'
    files = [os.path.join(images_path, p) for p in
sorted(os.listdir(images_path))]
    # 打乱数据库中的图像顺序
    sample = random.sample(files, 3)
    batch_extractor(images_path)
    ma = Matcher('D:/output/features.pck')
    # 查询图像名称
    s = 'D:/images/test.jpg'
    # s = 'images/test-1.jpeg'
    print('Query image =====')
    show_img(s)
    names, match = ma.match(s, topn=3)
    print('Result images =====')
    for i in range(3):
        # 计算 cosine 距离, 将相似性定义为 1-cosine 距离, 当两个图像越近, 此值相
        似指越高
        print('Match %s' % (1-match[i]))
        show_img(os.path.join(images_path, names[i]))

run() #运行从这里开始

```

3. 运行代码，观察效果。
4. 修改不同的特征、匹配算法、距离指标对最终图像检索结果的影响。
5. 思考：图像特征的含义；图像检索的关键技术可能会聚焦于哪些方面；图像检索可能的挑战有哪些。

附录 A：实验报告模板

人工智能导论 实验报告

实验题目： _____

姓 名： _____

学 号： _____

日 期： _____

自我评分： _____ **【X】** _____

自我评分说明：A+， A， B+， B， B-， C， D， 分别对应分数 95、90、85、80、75、70、60

诚信声明

本人郑重承诺：本实验程序和实验报告均是本人独立学习和工作所获得的成果。尽我所知，实验报告中除特别标注的地方外，不包含其他同学已经发表或撰写过的成果；实验程序中对代码工作的任何帮助者所作的贡献均做了明确的说明，并表达了谢意。

如有抄袭，本人愿意承担因此而造成的任何后果。

特此声明。

签名：_____

日期：_____

程序引用说明

序号	引用项	来源	相同代码行数
1	查找函数	《书名》	
2	构造有序链表函数	互联网网址	
3			
小计			

总代码行数_____；引用占比_____

1、实验简介

【实验内容的简要说明，具体说明实验完成的功能和性能要求】

2、程序框架

【实验程序共包含哪些函数，一一列举函数名和函数功能】

3、关键代码实现

【哪些函数是你认为最能体现自己工作成果的函数，说明函数实现基本思想（可用文字或图表示），以及具体的实验步骤（用伪代码或带注释代码）】

3.1 函数名 1

3.2 函数名 2

4、不足

【实验程序哪些函数功能还有缺陷或不足，或者程序架构有不足，或者性能还有待提高、或者代码不和规范等等，所有你自己对程序不满意的地方】

5、心得体会

【所有你在实验中的感受和想和老师说的话都可以放在这里，篇幅不限】