华为认证 HCIA-AI 系列教程

AI数学基础

实验指导手册

版本:3.0

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： http://e.huawei.com

华为认证体系介绍

基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，华为公司打造了业界唯一覆盖ICT全技术领域的认证体系，包含ICT技术架构认证、平台与服务认证和行业ICT认证三类认证。

根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。

华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代的新型ICT人才，构建良性的ICT人才生态。

华为认证HCIA-AI V3.0定位于培养和认证具备使用机器学习、深度学习等算法设计、开发AI产品和解决方案能力的工程师。

通过HCIA-AI V3.0认证，将证明您了解人工智能发展历史、华为昇腾AI体系和全栈全场景AI战略知识，掌握传统机器学习和深度学习的相关算法；具备利用TensorFlow开发框架和MindSpore开发框架进行搭建、训练、部署神经网络的能力；能够胜任人工智能领域销售、市场、产品经理、项目管理、技术支持等岗位。

目录

1 实验介绍

1.1 实验简介

本课程将介绍基于 Python 的数学基础实验实现，包括基础数学算子、线性代数、概率论和最优化。通过学习本课程，学员将掌握基于 Python 的基础数学方法实现，并能够将其运用到实际项目中，解决业务问题。

通过本实验，您将能够：

- 掌握使用 Python 实现基本数学算子
- 掌握使用 Python 线性代数与概率论中的相关计算
- 掌握使用 Python 最优化的实现

1.2 内容描述

本实验指导书共包含 6 个实验，包含：

- 实验一 基础数学实验。
- 实验二 线性代数相关算子的实现。
- 实验三 概率论相关统计分布的实现。
- 实验四 最小二乘法的实现。
- 实验五 梯度下降法的实现。

1.3 前置技能要求

此课程为基于 Python 的数学基础课程，因此希望您在开始本实验前掌握以下技能，

- 掌握基本线性代数、概率论与最优化知识。

1.4 实验环境说明

- Windows 7 /Windows 10 64 位系统的 PC 机，同时需要能够访问 Internet 网；
- 根据实际的操作系统版本下载并安装 Anaconda 3 4.4.0 及以上版本；
- 每套实验环境可供 1 名学员上机操作。

注意：以下实验均在 Anaconda3 中的 Jupyter Notebook 进行。

2 基础数学实验

2.1 基础数学介绍

2.1.1 内容介绍

基础数学知识在数据挖掘领域有着大量的应用，尤其是在算法设计和数值处理方面。本章节的主要目的就是基于 Python 语言和相应的基础数学模块，实现一些常用的数学基础算法，为进入数据挖掘的学习提供基础支持。

2.1.2 框架介绍

本章节使用到的框架主要包括 math 库，numpy 库和 scipy 库。math 库是 Python 的标准库，提供一些常用的数学函数；numpy 库是 Python 的一个数值计算拓展库，主要用于处理线性代数，随机数生成，傅里叶变换等问题；scipy 库主要用于统计，优化，插值，积分等问题的处理。

2.2 基础数学实现

导入相应库：

```
import math
import numpy as np
```

2.2.1 ceil 实现

ceil(x) 取大于等于 x 的最小的整数值，如果 x 是一个整数，则返回自身。

代码输入：

```
math.ceil(4.01)
```

结果输出：

```
5
```

代码输入：

```
math.ceil(4.99)
```

结果输出：

```
5
```

2.2.2 floor 实现

floor(x)取小于等于 x 的最大的整数值，如果 x 是一个整数，则返回自身。

代码输入：

```
math.floor(4.1)
```

结果输出：

```
4
```

代码输入：

```
math.floor(4.999)
```

结果输出：

```
4
```

2.2.3 degrees 实现

degrees(x)把 x 从弧度转换成角度。

代码输入：

```
math.degrees(math.pi/4)
```

结果输出：

```
45.0
```

代码输入：

```
math.degrees(math.pi)
```

结果输出：

```
180.0
```

2.2.4 exp 实现

exp(x)返回 math.e，也就是 2.71828 的 x 次方。

代码输入：

```
math.exp(1)
```

结果输出：

```
2.718281828459045
```

2.2.5 fabs 实现

fabs(x)返回 x 的绝对值。

代码输入：

```
math.fabs(-0.003)
```

结果输出：

```
0.003
```

2.2.6 factorial 实现

factorial(x)取 x 的阶乘的值。

代码输入：

```
math.factorial(3)
```

结果输出：

```
6
```

2.2.7 fsum 实现

fsum(iterable)对迭代器里的每个元素进行求和操作。

代码输入：

```
math.fsum([1,2,3,4])
```

结果输出：

```
10
```

2.2.8 fmod 实现

fmod(x, y)得到 x/y 的余数，其值是一个浮点数。

代码输入：

```
math.fmod(20,3)
```

结果输出：

```
2.0
```

2.2.9 log 实现

log([x, base])返回 x 的自然对数，默认以 e 为底数，base 参数给定时，按照给定的 base 返回 x 的对数，计算式为：log(x)/log(base)。

代码输入：

```
math.log(10)
```

结果输出：

```
2.302585092994046
```

2.2.10 sqrt 实现

sqrt(x)求 x 的平方根。

代码输入：

```
math.sqrt(100)
```

结果输出：

```
10.0
```

2.2.11 pi 实现

pi 数字常量，圆周率。

代码输入：

```
math.pi
```

结果输出：

```
3.141592653589793
```

2.2.12 pow 实现

pow(x, y)返回 x 的 y 次方，即 x**y。

代码输入：

```
math.pow(3,4)
```

结果输出：

```
81.0
```

3 线性代数实验

3.1 线性代数内容介绍

3.1.1 线性代数介绍

线性代数是一门被广泛运用于各工程技术领域的学科。用线性代数的相关概念和结论，可以极大地简化数据挖掘中相关公式的推导和表述。线性代数将复杂的问题简单化，让我们能够对问题进行高效地数学运算。

线性代数是一个数学工具，它不仅提供了有助于操作数组的技术，还提供了像向量和矩阵这样的数据结构用来保存数字和规则，以便进行加，减，乘，除的运算。

3.1.2 代码实现介绍

numpy 是一款基于 Python 的数值处理模块，在处理矩阵数据方面有很强大的功能与优势。因为线性代数的主要内容就是对矩阵的处理，所以本章节主要的内容都是基于 numpy 进行展开。另外也会涉及到方程组求解，所以也会用到数学科学库 scipy。

3.2 线性代数实现

导入相应库：

```
import numpy as np
import scipy as sp
```

3.2.1 reshape 运算

在数学中并没有 reshape 运算，但是在 numpy 运算库中是一个非常常用的运算，用来改变一个张量的维度数和每个维度的大小，例如一个 10x10 的图片在保存时直接保存为一个包含 100 个元素的序列，在读取后就可以使用 reshape 将其从 1x100 变换为 10x10。示例如下：

代码输入：

生成一个包含整数 0~11 的向量

```
x = np.arange(12)
print(x)
```

结果输出：

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
```

查看数组大小

```
x.shape
```

结果输出：

```
(12,)
```

将 x 转换成二维矩阵，其中矩阵的第一个维度为 1

```
x = x.reshape(1,12)
print(x)
```

结果输出：

```
 [[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11]]
```

查看数组大小

```
x.shape
```

结果输出：

```
(1, 12)
```

将 x 转换 3x4 的矩阵

```
x = x.reshape(3,4)
print(x)
```

结果输出：

```
[[ 0,  1,  2,  3],
 [ 4,  5,  6,  7],
 [ 8,  9, 10, 11]]
```

3.2.2 转置实现

向量和矩阵的转置是交换行列顺序，而三维及以上张量的转置就需要指定转换的维度。

代码输入：

生成 3*4 的矩阵并转置

```
A = np.arange(12).reshape(3,4)
print(A)
```

结果输出：

```
[[ 0,  1,  2,  3],
 [ 4,  5,  6,  7],
 [ 8,  9, 10, 11]]
A.T
```

结果输出：

```
array([[ 0,  4,  8],
       [ 1,  5,  9],
```

```
       [ 2,  6, 10],
       [ 3,  7, 11]])
```

3.2.3 矩阵乘法实现

矩阵乘法：记两个矩阵分别为 A 和 B，两个矩阵能够相乘的条件为第一个矩阵的列数等于第二个矩阵的行数。

代码输入：

```
A = np.arange(6).reshape(3,2)
B = np.arange(6).reshape(2,3)
print(A)
```

结果输出：

```
[[0 1]
 [2 3]
 [4 5]]
print(B)
```

结果输出：

```
[[0, 1, 2],
 [3, 4, 5]]
```

矩阵相乘

```
np.matmul(A,B)
```

结果输出：

```
array([[ 3,  4,  5],
       [ 9, 14, 19],
       [15, 24, 33]])
```

3.2.4 矩阵对应运算

元素对应运算：针对形状相同矩阵的运算统称，包括元素对应相乘、相加等，即对两个矩阵相同位置的元素进行加减乘除等运算。

代码输入：

创建矩阵

```
A = np.arange(6).reshape(3,2)
```

矩阵相乘

```
print(A*A)
```

结果输出：

```
array([[ 0,  1],
       [ 4,  9],
       [16, 25]])
```

矩阵相加：

```
print(A + A)
```

结果输出：

```
array([[ 0,  2],
       [ 4,  6],
       [ 8, 10]])
```

3.2.5 逆矩阵实现

只有方阵才有逆矩阵，逆矩阵实现。

代码输入：

```
A = np.arange(4).reshape(2,2)
print(A)
```

结果输出：

```
array([[0, 1],
       [2, 3]])
```

求逆矩阵：

```
np.linalg.inv(A)
```

结果输出：

```
array([[-1.5,  0.5],
       [ 1. ,  0. ]])
```

3.2.6 特征值与特征向量

求矩阵的特征值与特征向量并实现可视化。

代码输入:

导入相应库：

```
from scipy.linalg import eig
import numpy as np
import matplotlib.pyplot as plt
```

求特征值与特征向量：

```
A = [[1, 2],#生成一个 2*2 的矩阵
     [2, 1]]
evals, evecs = eig(A) #求 A 的特征值（evals）和特征向量(evecs)
evecs = evecs[:, 0], evecs[:, 1]
```

plt.subplots() 返回一个 Figure 实例 fig 和一个 AxesSubplot 实例 ax。fig 代表整个图像，ax 代表坐标轴和画的图。 作图：

```
fig, ax = plt.subplots()
```

让坐标轴经过原点：

```
for spine in ['left', 'bottom']:#让在左下角的坐标轴经过原点
    ax.spines[spine].set_position('zero')
```

画出网格：

```
ax.grid(alpha=0.4)
```

设置坐标轴的范围：

```
xmin, xmax = -3, 3
ymin, ymax = -3, 3
ax.set(xlim=(xmin, xmax), ylim=(ymin, ymax))
```

画出特征向量。用一个箭头指向要注释的地方，再写上一段话的行为，叫做 annotate。s 是输入内容；xy:箭头指向；xytext 文字所处的位置；arrowprops 通过 arrowstyle 表明箭头的风格或种类：

```
for v in evecs:
    ax.annotate(s="", xy=v, xytext=(0, 0),
                arrowprops=dict(facecolor='blue',
                shrink=0,
                alpha=0.6,
                width=0.5))
```

画出特征空间：

```
x = np.linspace(xmin, xmax, 3)#在指定的间隔内返回均匀间隔的数字
for v in evecs:
    a = v[1] / v[0] #沿特征向量方向的单位向量
    ax.plot(x, a * x, 'r-', lw=0.4)# 参数 lw 表示图线的粗细
plt.show()
```

可视化图像：

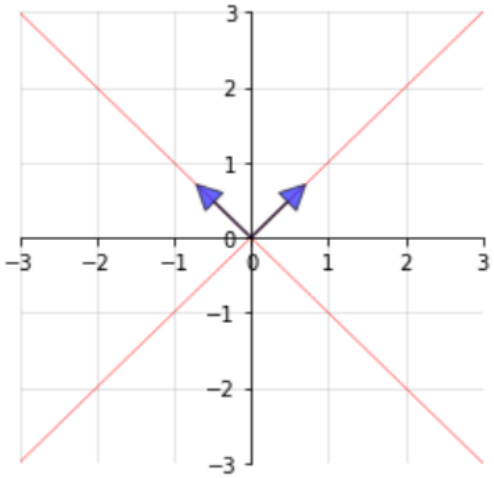

图 2-1

图解：蓝箭头洗向量为特征向量，两条红色直线组成的空间为特征空间

3.2.7 求行列式

求一个矩阵的行列式。

代码输入：

```
E = [[1, 2, 3],
     [4, 5, 6],
     [7, 8, 9]]
print(np.linalg.det(E))
```

结果输出：

```
-9.51619735392994e-16
```

3.2.8 奇异值分解实现

案例引入：

假如我们是一个新闻资讯提供方，每天会有大量不同领域的文章需要推送给用户，但是这些文章质量参差不齐，为了更好的用户体验，我们需要对海量的文章进行归类然后选取质量比较好的推送给用户。那么问题来了，对于海量文章，用什么方式可以自动将文章归类呢？这里介绍一种比较简单的思想如下：通常情况下文章的标题是由少量核心概念词构成，这些核心词代表了文章的主旨，因此我们可以间接通过文章标题的相似度来判断两篇文章是否是一类。文章标题的相似度的判断是以词之间的相似度作为基础的，那么如何判断标题中词和词之间的相似度呢？ 下面我们使用奇异值分解来解决这个问题。假设有 8 个标题，每个标题包含的关键词如下：

title_1 = ["dad","dad","stock"]

title_2 = ["books","books","value","estate"]

title_3 = ["books","decomposition"]

title_4 = ["stock"]

title_5 = ["dad"]

title_6 = ["value","singular","decomposition"]

title_7 = ["dad","singular"]

title_8 = ["singular","estate","decomposition"]

代码输入：

导入相应模块：

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

输入关键字：

```
words = ["books","dad","stock","value","singular","estate","decomposition"]
```

设已知 8 个标题，7 个关键字。记录每个标题中每个关键字出现的次数，得矩阵 X。 X 中每一行表示一个标题，每一列表示一个关键字，矩阵中的每个元素表示一个关键字在一个标题中出现的次数。

```
X=np.array([[0,2,1,0,0,0,0],[2,0,0,1,0,1,0],[1,0,0,0,0,0,1],[0,0,1,0,0,0,0],[0,
1,0,0,0,0,0],[0,0,0,1,1,0,1],[0,1,0,0,1,0,0],[0,0,0,0,1,1,1]])
```

进行奇异值分解：

```
U,s,Vh=np.linalg.svd(X)
```

输出左奇异矩阵 U 及其 shape：

```
print("U=",U)
print("U.shape",U.shape)
```

输出结果：

```
U= [[-1.87135757e-01 -7.93624528e-01  2.45011855e-01 -2.05404352e-01
  -3.88578059e-16  5.75779114e-16 -2.57394431e-01 -4.08248290e-01]
 [-6.92896814e-01  2.88368077e-01  5.67788037e-01  2.22142537e-01
   2.54000254e-01 -6.37019839e-16 -2.21623012e-02  2.05865892e-17]
 [-3.53233681e-01  1.22606651e-01  3.49203461e-02 -4.51735990e-01
  -7.62000762e-01  1.27403968e-15  2.72513448e-01  3.80488702e-17]
 [-2.61369658e-02 -1.33189110e-01  7.51079037e-02 -6.44727454e-01
   5.08000508e-01  1.77635684e-15  3.68146235e-01  4.08248290e-01]
 [-8.04993957e-02 -3.30217709e-01  8.49519758e-02  2.19661551e-01
  -2.54000254e-01 -4.81127681e-16 -3.12770333e-01  8.16496581e-01]
 [-3.95029694e-01  1.56123876e-02 -5.28290830e-01 -6.82340484e-02
   1.27000127e-01 -7.07106781e-01 -2.09360158e-01  1.55512464e-17]
 [-2.02089013e-01 -3.80395849e-01 -2.12899198e-01  4.80790894e-01
   8.04483689e-16 -1.60632798e-15  7.33466480e-01  1.76241226e-16]
 [-3.95029694e-01  1.56123876e-02 -5.28290830e-01 -6.82340484e-02
   1.27000127e-01  7.07106781e-01 -2.09360158e-01 -1.23226632e-16]]
U.shape (8, 8)
```

输出奇异值矩阵及其 shape：

```
print("s=",s)
print("s.shape",s.shape)
```

按每个奇异值一一对应一个左奇异向量和一个右奇异向量奇异值从大到小排列输出结果：

```
s= [2.85653844 2.63792139 2.06449303 1.14829917 1.         1.
 0.54848559]
s.shape (7,)
```

输出右奇异矩阵 Vh 及其 shape：

```
print("Vh",Vh)
print("Vh.shape",Vh.shape)
```

输出结果：

```
Vh [[-6.08788345e-01 -2.29949618e-01 -7.46612474e-02 -3.80854846e-01
  -3.47325416e-01 -3.80854846e-01 -4.00237243e-01]
 [ 2.65111314e-01 -8.71088358e-01 -3.51342402e-01  1.15234846e-01
  -1.32365989e-01  1.15234846e-01  5.83153945e-02]
 [ 5.66965547e-01  1.75382762e-01  1.55059743e-01  1.91316736e-02
  -6.14911671e-01  1.91316736e-02 -4.94872736e-01]
 [-6.48865369e-03  2.52237176e-01 -7.40339999e-01  1.34031699e-01
   2.99854608e-01  1.34031699e-01 -5.12239408e-01]
 [-2.54000254e-01 -2.54000254e-01  5.08000508e-01  3.81000381e-01
   2.54000254e-01  3.81000381e-01 -5.08000508e-01]
 [ 0.00000000e+00 -7.68640544e-16  2.33583082e-15 -7.07106781e-01
  -1.21802199e-15  7.07106781e-01  1.91457709e-15]
 [ 4.16034348e-01 -1.71550021e-01  2.01922906e-01 -4.22112199e-01
   5.73845817e-01 -4.22112199e-01 -2.66564648e-01]]
Vh.shape (7, 7)
```

规定坐标轴的范围

```
plt.axis([-0.8,0.2,-0.8,0.8])
```

原每个关键字由 1*8 的向量表示，现降维成 1*2 的向量以便进行可视化

```
for i in range(len(words)):
    plt.text(U[i,0],U[i,1],words[i])
plt.show()
```

可视化结果：

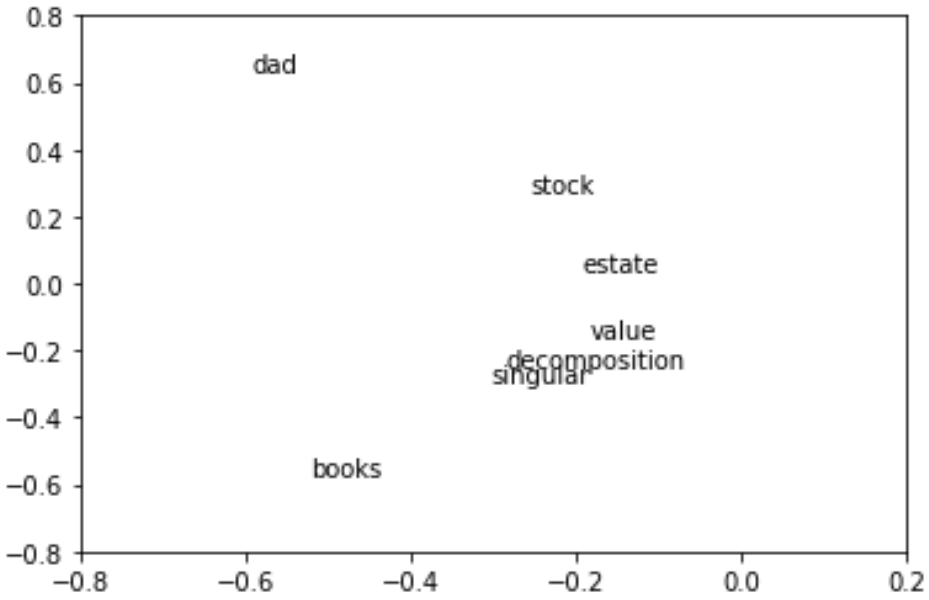

图 2-2

图解：将到 2 维可视化后，我们可以将关键词聚类，如 singular，value 和 decomposition 三个词距离比较近可以被划分为一组，而 stock 和 estate 经常同时出现。

在得到了词的向量表示后，我们就可以根据选取一种向量距离（例如：欧式距离、曼哈顿距离等）计算的方式来计算词间的相似度从而可以再根据一些策略（例如：词对相似度的和、均值等）得到文章标题之间的相似度；之后我们就可以通过文章标题的相似度来近似表示文章内容的相似度从而完成对文章的聚类。

3.2.9 奇异值分解应用 – 图像压缩

一副灰度图像可以看作一个矩阵，对这样矩阵进行奇异值分解，奇异值矩阵的奇异值是按照从大到小的顺序排列的。对应奇异值大的奇异向量所保存的信息量越大，而奇异值大小一般都衰减得比较快。因此前 K 个奇异值及其对应的奇异向量就包含了图像中的大部分信息。所以前 K 项奇异值及其奇异向量来组成的图像可以达到基本和原图一样的清晰度，但是数据量缺大大减少了。这样就可以达到图像数据压缩的效果。

代码输入：

```
import numpy as np
from pylab import *
import matplotlib.pyplot as plt

#读取并保存灰度图像
img = imread('lena.jpg')[:,:,0]
plt.savefig('./lena_gray')
plt.gray()
#画出灰度图
plt.figure(1)
plt.imshow(img)
```

输出结果：

图 2-3

```
#读取并打印图像的长宽
```

```
m,n = img.shape
print(np.shape(img))
```

输入结果：

```
(184, 324)
```

```
#对图像矩阵进行奇异值分解，
U,sigma,V = np.linalg.svd(img)
#打印奇异值大小
print(np.shape(sigma))
```

输出结果：

```
(184,)
```

```
#将奇异值整理成一个对角矩阵
sigma = resize(sigma, [m,1])*eye(m,n)
#取前 K 个奇异值及其奇异向量用于压缩图像
k= 100
#用前 K 个奇异值及其奇异向量构造新图像
img1 = np.dot(U[:,0:k],np.dot(sigma[0:k,0:k],V[0:k,:]))
plt.figure(2)
#打印压缩后的效果图
plt.imshow(img1)
plt.show()
```

图 2-4

3.2.10 线性方程组求解

求解线性方程组比较简单，只需要用到一个函数(scipy.linalg.solve)就可以了。

案例引入：有三种价格未知的水果，苹果、香蕉、葡萄；已知，李雷购买 10 斤苹果、2 斤香蕉、5 斤葡萄花费了 10 元，韩梅梅购买 4 斤苹果、4 斤香蕉、2 斤葡萄花费了 8 元，汤姆购买 2 斤苹果、2 斤香蕉、2 斤葡萄花费了 5 元；问，苹果、香蕉、葡萄分别多少钱一斤?

根据已知条件，可以构建如下多元方程组，其中 x_1、x_2、x_3 分别是苹果、香蕉、葡萄的价格，目的是求解 x_1、x_2、x_3 的值：

$10x_1 + 2x_2 + 5x_3 = 10$

$4x_1 + 4x_2 + 2x_3 = 8$

$2x_1 + 2x_2 + 2x_3 = 5$

代码输入：

```
import numpy as np
from scipy.linalg import solve
a = np.array([[10, 2, 5], [4, 4, 2], [2, 2, 2]])
b = np.array([10,8,5])
x = solve(a, b)
print(x)
```

print(x)结果输出：

```
array([0.25 1.25 1.  ])
```

4 概率论实验

4.1 概率论内容介绍

4.1.1 概率论介绍

概率论是研究随机现象数量规律的数学分支。随机现象是相对于决定性现象而言的，在一定条件下必然发生某一结果的现象称为决定性现象。

概率论是用来描述不确定性的数学工具，很多数据挖掘中的算法都是通过描述样本的概率相关信息或推断来构建模型。

4.1.2 实验介绍

本章节主要实现概率与统计相关的知识点，主要用到的框架是 numpy 和 scipy 框架。

4.2 概率论内容实现

导入相应库：

```
import numpy as np
import scipy as sp
```

4.2.1 均值实现

数据准备

```
ll = [[1,2,3,4,5,6],[3,4,5,6,7,8]]
```

代码输入：

```
np.mean(ll)    #全部元素求均值
```

结果输出：

4.5

```
np.mean(ll,0) #按列求均值，0 代表列向量
```

结果输出：

```
array([2., 3., 4., 5., 6., 7.])
np.mean(ll,1) #按行求均值，1 表示行向量
```

结果输出：

```
array([3.5, 5.5])
```

4.2.2 方差实现

数据准备

```
b=[1,3,5,6]
ll=[[1,2,3,4,5,6],[3,4,5,6,7,8]]
```

求方差：

```
np.var(b)
```

结果输出：

```
3.6875
```

代码输入：

```
np.var(ll,1)  #第二个参数为 1，表示按行求方差
```

结果输出：

```
[2.91666667 2.91666667]
```

4.2.3 标准差实现

数据准备

```
ll=[[1,2,3,4,5,6],[3,4,5,6,7,8]]
```

代码输入：

```
np.std(ll)
```

结果输出：

```
1.9790570145063195
```

4.2.4 协方差实现

数据准备

```
b=[1,3,5,6]
```

代码输入：

```
np.cov(b)
```

结果输出：

```
4.916666666666666
```

4.2.5 相关系数

数据准备

```
vc=[1,2,39,0,8]
vb=[1,2,38,0,8]
```

利用函数实现：

```
np.corrcoef(vc,vb)
```

结果输出：

```
array([[1.        , 0.99998623],
       [0.99998623, 1.        ]])
```

4.2.6 二项分布实现

服从二项分布的随机变量 X 表示在 n 次独立同分布的伯努利试验中成功的次数，其中每次试验的成功概率为 p。

代码输入：

```
from scipy.stats import binom, norm, beta, expon
import numpy as np
import matplotlib.pyplot as plt

#n,p 对应二项式公式中的事件成功次数及其概率，size 表示采样次数
binom_sim = binom.rvs(n=10, p=0.3, size=10000)
print('Data:',binom_sim)
print('Mean: %g' % np.mean(binom_sim))
print('SD: %g' % np.std(binom_sim, ddof=1))
#生成直方图，x 指定每个 bin(箱子)分布的数据,对应 x 轴，binx 是总共有几条条状图，normed 值密度，也就是每个条状图的占比例比,默认为 1
plt.hist(binom_sim, bins=10, normed=True)
plt.xlabel(('x'))
plt.ylabel('density')
plt.show()
```

结果输出：

```
Data: [2 4 3 ... 3 4 1]#符合二项分布的 10000 个数
Mean: 2.9821
SD: 1.43478
```

二项分布图如下：

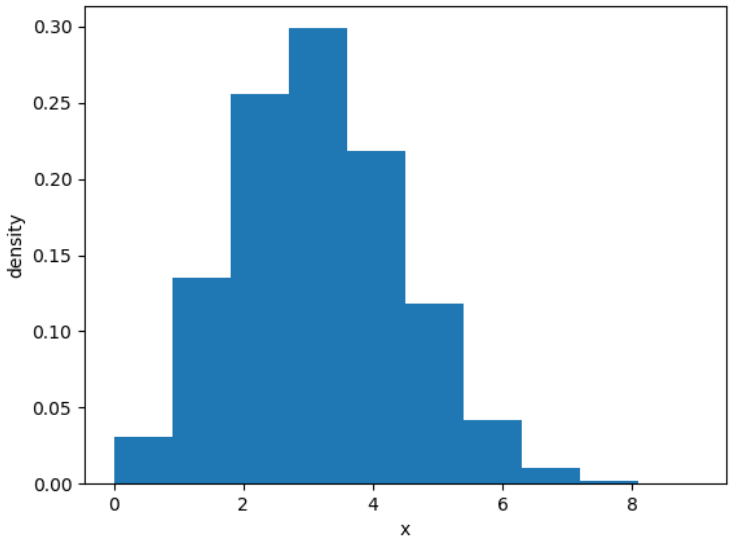

图 4-1

4.2.7 泊松分布实现

一个服从泊松分布的随机变量 X，表示在具有比率参数λ的一段固定时间间隔内，事件发生的次数。参数λ告诉你该事件发生的比率。随机变量 X 的平均值和方差都是λ。

代码输入：

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

#产生 10000 个符合 lambda=2 的泊松分布的数
X= np.random.poisson(lam=2, size=10000)

a = plt.hist(X, bins=15, normed=True, range=[0, 15])
#生成网格
plt.grid()
plt.show()
```

泊松分布图如下：

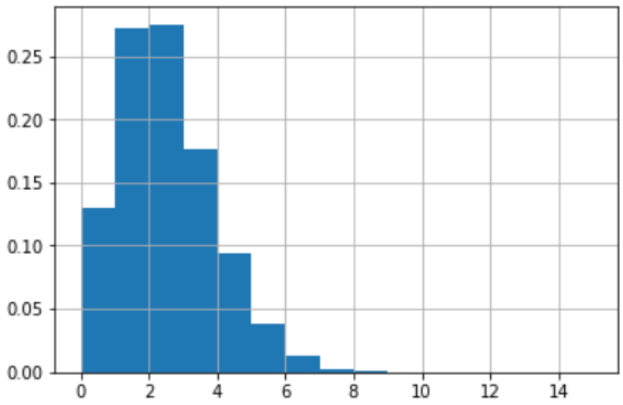

图 4-2

4.2.8 正态分布

正态分布是一种连续分布，其函数可以在实线上的任何地方取值。正态分布由两个参数描述：分布的平均值μ和标准差σ 。

代码输入：

```
from scipy.stats import norm
import numpy as np
import matplotlib.pyplot as plt

mu = 0
sigma = 1
#分布采样点
x = np.arange(-5, 5, 0.1)
#生成符合 mu,sigma 的正态分布
y = norm.pdf(x, mu, sigma)
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('density')
plt.show()
```

分布图如下：

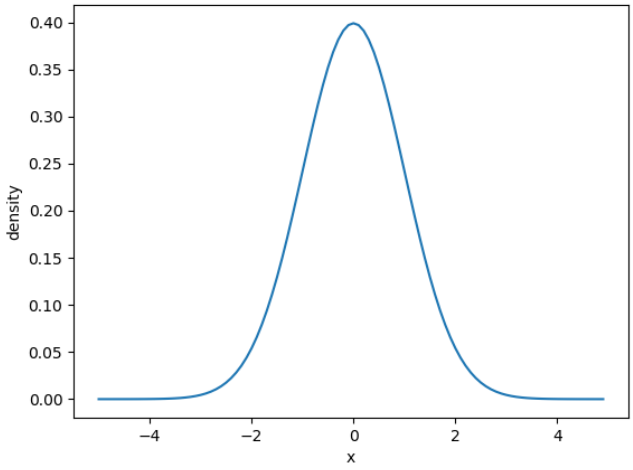

图 4-3

4.3 应用

4.3.1 图像加噪声

案例引入：比如在深度学习中，我们需要训练一个图像分类的模型，但是能拿到的图像不是很多，那有没有一种方法可以根据现有的图像生成一些新的图像出来呢？增加的图像需要满足一些条件：图像本身的内容不能发生根本性的改变，比如图像能看出来是一只狗的话扩增出来的图像应该也是一只狗而不能成为只剩下一个毛茸茸的尾巴。最直接的方式就是在原始的图像上增加一些噪声，也就是使得图像变得粗糙同时保持图像原本要表述的信息。那么怎么给图像增加噪声呢，可以增加哪些噪声呢？我们下面来实现一下这个过程，该过程需要使用到 Python 图像处理的模块 OpenCV，我们使用著名的 Lena 图作为原始图像。

代码输入：

载入原始图片：

```
# 载入原始图片
# -*- coding: utf-8 -*-
from numpy import *
from scipy import *
import numpy as np
import cv2

srcImage = cv2.imread(r"C:\Users\zwx713302\Desktop\jupyter_code\HCIA-AI_base_math\Lena.jpg")
print(srcImage.shape)                          # 打印图像 size
```

```
cv2.namedWindow("Original image")        # 图像显示窗口命名
cv2.imshow("Original image", srcImage) # 显示图像
k = cv2.waitKey(0)
```

灰度处理原始图片：

```
# 灰度处理原始图片
grayImage = cv2.cvtColor(srcImage,cv2.COLOR_BGR2GRAY) #灰度变换
print(grayImage.shape)
cv2.imshow("grayimage", grayImage)
k = cv2.waitKey(0)
```

加入高斯噪声（均值：mean，方差：var，比例：percent)：

```
# 加入高斯噪声
import numpy as np
image = np.array(grayImage/255, dtype=float)
percent=0.01  # 图像加入噪声比例
num=int(percent*image.shape[0]*image.shape[1])

for i in range(num):
    temp1=np.random.randint(image.shape[0])
    temp2=np.random.randint(image.shape[1])

    mean=0
    var=0.04
    noise = np.random.normal(mean, var ** 0.5, 1)
    image[temp1][temp2] +=  noise
out=image

if out.min() < 0:
    low_clip = -1.
else:
    low_clip = 0.
out = np.clip(out, low_clip, 1)
gasuss_image = np.uint8(out*255)
print(gasuss_image.shape)
cv2.imshow("gasuss_image", gasuss_image)
k = cv2.waitKey(0)
```

加入泊松噪声（泊松参数：scale，比例：percent)

```
# 加入泊松噪声
from scipy.stats import expon
import numpy as np
image = np.array(grayImage, dtype=float)
percent=0.001  # 图像加入噪声比例
num=int(percent*image.shape[0]*image.shape[1])
for i in range(num):
    temp1=np.random.randint(image.shape[0])
```

```
    temp2=np.random.randint(image.shape[1])

    scale=150
    noise = np.random.poisson(scale,1)
    image[temp1][temp2] +=  noise

out=image
if out.min() < 0:
    low_clip = -1.
else:
    low_clip = 0.
out = np.clip(out, low_clip, 255)
expon_image = np.uint8(out)
print(expon_image.shape)
cv2.imshow("expon_image", expon_image)
k = cv2.waitKey(0)
```

图 4-7

图解：从左到右分别是原始图像、高斯噪声图像、泊松噪声图像

5 最优化实验

5.1 最小二乘法实现

5.1.1 算法介绍

最小二乘法（Least Square Method），做为分类回归算法的基础，有着悠久的历史。它通过最小化误差的平方和寻找数据的最佳函数匹配。利用最小二乘法可以简便地求得未知的参数，并使得预测的数据与实际数据之间误差的平方和为最小。

5.1.2 案例引入

假设我们需要根据历史数据来预测一只股票的价格，根据以往的经验我们看到股票的价格随着时间的变化呈现出类似正弦的波动，因此我们想用一个多项式$y = \theta_n x^n + \theta_{n-1} x^{n-1} + \ ... + \theta_1 x^0 + \theta_0 x^0$来表示这种关系，其中y表示股票价格，x表示时间；其中参数$\theta_i (i = 0, \dots\dots, n)$未知，一旦$\theta_i (i = 0, \dots\dots, n)$确定了，那我们就可以根据公式来估算任意时间股票的价格。那么我们怎么根据以往的历史数据中的信息来求得比较好的参数$\theta_i (i = 0, \dots\dots, n)$使得表达式尽可能的符合实际情况呢？最直接的方法就是将历史数据中的时间带入到多项式中得到预测的股票价格$\hat{y}$，该预测价格$\hat{y}$价格真实的价格y尽可能的接近，那么我们就可以认为这个多项式已经比较好的反映了真实的规律；如何表示多项式得到的股票价格和真实股票价格之间的差距呢，常用的一种方式是使用两者之间的差值的平方来评估，即使用函数$\frac{1}{2}(\hat{y} - y)^2$；但是我们有很多的历史数据，好的多项式应该在整体数据上达到最优而不仅仅是某个数据点，我们使用$\frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}(\hat{y_i} - y_i)^2$来表示整整体性的好坏，其中 m 表示历史数据点数量；那么该问题就简化成了求解使得$\frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}(\hat{y_i} - y_i)^2$尽可能小的$\theta_i (i = 0, \dots\dots, n)$这样一个优化问题。下面将首先构造一份带有噪声的正弦数据假设是获取到的历史数据，然后根据这个历史数据，使用最小二乘法来求解$\frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}(\hat{y_i} - y_i)^2$最小的参数$\theta_i (i = 0, \dots\dots, n)$。

5.1.3 代码实现

代码输入：

```
import numpy as np
import scipy as sp
import pylab as pl
from scipy.optimize import leastsq  # 引入最小二乘函数
```

```
n = 9  # 多项式次数
```

定义目标函数：

```
def real_func(x):
    #目标函数：sin(2*pi*x)
     return np.sin(2 * np.pi * x)
```

定义多项式函数，用多项式去拟合数据：

```
def fit_func(p, x):
     f = np.poly1d(p)
     return f(x)
```

定义残差函数，残差函数值为多项式拟合结果与真实值的差值：

```
def residuals_func(p, y, x):
     ret = fit_func(p, x) - y
     return ret


x = np.linspace(0, 1, 9)  # 随机选择 9 个点作为 x
x_points = np.linspace(0, 1, 1000)  # 画图时需要的连续点
y0 = real_func(x)  # 目标函数
y1 = [np.random.normal(0, 0.1) + y for y in y0]  # 在目标函数上添加符合正态分布噪声后的函数
p_init = np.random.randn(n)  # 随机初始化多项式参数
# 调用 scipy.optimize 中的 leastsq 函数，通过最小化误差的平方和来寻找最佳的匹配函数
#func 是一个残差函数，x0 是计算的初始参数值，把残差函数中除了初始化以外的参数打包到 args 中
plsq = leastsq(func=residuals_func, x0=p_init, args=(y1, x))

print('Fitting Parameters: ', plsq[0])  # 输出拟合参数

pl.plot(x_points, real_func(x_points), label='real')
pl.plot(x_points, fit_func(plsq[0], x_points), label='fitted curve')
pl.plot(x, y1, 'bo', label='with noise')
pl.legend()
pl.show()
```

结果输出：

```
Fitting Parameters:  [-1.22007936e+03  5.79215138e+03 -1.10709926e+04
1.08840736e+04
 -5.81549888e+03  1.65346694e+03 -2.42724147e+02  1.96199338e+01
 -2.14013567e-02]
```

可视化图像：

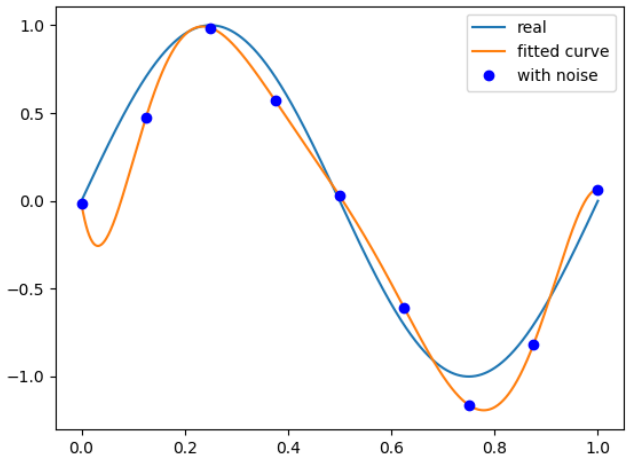

图 5-1

5.2 梯度下降法实现

5.2.1 算法介绍

梯度下降法（gradient descent），又名最速下降法，是求解无约束最优化问题最常用的方法，它是一种迭代方法，每一步主要的操作是求解目标函数的梯度向量，将当前位置的负梯度方向作为搜索方向（因为在该方向上目标函数下降最快，这也是最速下降法名称的由来）。

梯度下降法特点：越接近目标值，步长越小，下降速度越慢。

5.2.2 案例引入

如今房地产行火热，我们想要根据一些信息来预测一套房屋的价格，要怎么做呢？假如根据经验可以知道房价y是由房屋面积x0、小区的容积率x1、绿化面积x2这三个关键因素决定的，并且呈现出的是线性关系，也就是说认为$y = \theta_0$* x0 + θ_1* x1 + θ_2* x2 以大概描房价和关键因素的关系，其中θ_0，θ_1，θ_2未知；一旦θ_0，θ_1，θ_2确定了，那我们就可以根据公式来估算任意一套房屋的价格。为了得到θ_0，θ_1，θ_2，我们拿到了全市一份房价及其房屋面积、小区的容积率、绿化面积的数据，那么如何使用这份数据帮助我们求取θ_0，θ_1，θ_2呢？直觉来讲，好的θ_0，θ_1，θ_2应该是对已拿到的大部分真实数据通过 θ_0*s + θ_1*r + θ_2*d 得到的房价y_{pre}应该和真实的房价y不要相差太多，常用$\frac{1}{2}(y - y_{pre})^2$来描述房价和预测房价的差值；类似 5.2.1 中的案例，我们有很多的已知数据，好的多项式应该在整体数据上达到最优而不仅仅是某个数据点，因此使用$\frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}(y_i - y_{(pre,i)})^2$表示整整体性的好坏，其中 m 表示已知数据点数量。那么该问题就简化成了求解使得$\frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}(y_i - y_{(pre,i)})^2$尽可能小的$\theta_0$，$\theta_1$，$\theta_2$这样一个优化问题。如下案例：我们拿到了 5 份真实数据：

房价	房屋面积	容积率	绿化面积
95.364	1	0	3
97.217205	1	1	3
75.195834	1	2	3
60.105519	1	3	2
49.342380	1	4	4

下面我们将通过梯度下降的算法去求解最优的θ_0，θ_1，θ_2，使得$\frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left(y_i - y_{(pre,i)}\right)^2$最小。

5.2.3 代码实现

代码输入：

训练集(x,y)共 5 个样本,每个样本点有 3 个分量 (x0,x1,x2)

```
x = [(1, 0., 3), (1, 1., 3), (1, 2., 3), (1, 3., 2), (1, 4., 4)]
y = [95.364, 97.217205, 75.195834, 60.105519, 49.342380]  #y[i] 样本点对应的输出
epsilon = 0.0001  #迭代阀值，当两次迭代损失函数之差小于该阀值时停止迭代
alpha = 0.01  #学习率
diff = [0, 0]
max_itor = 1000
error1 = 0
error0 = 0
cnt = 0
m = len(x)
#初始化参数
theta0 = 0
theta1 = 0
theta2 = 0
while True:
    cnt += 1

    # 参数迭代计算
    for i in range(m):
        # 拟合函数为 y = theta0 * x[0] + theta1 * x[1] +theta2 * x[2]
        # 计算残差，即拟合函数值-真实值
        diff[0] = (theta0* x[i][0] + theta1 * x[i][1] + theta2 * x[i][2]) -
y[i]
        # 梯度 = diff[0] * x[i][j]。根据步长*梯度更新参数
        theta0 -= alpha * diff[0] * x[i][0]
        theta1 -= alpha * diff[0] * x[i][1]
        theta2 -= alpha * diff[0] * x[i][2]
```

```
    # 计算损失函数
    error1 = 0
    for lp in range(len(x)):
        error1 += (y[lp]-(theta0* x[lp][0] + theta1 * x[lp][1] + theta2 *
x[lp][2]))**2/2
    #若当两次迭代损失函数之差小于该阀值时停止迭代，跳出循环；
    if abs(error1-error0) < epsilon:
        break
    else:
        error0 = error1

    print(' theta0 : %f, theta1 : %f, theta2 : %f, error1 : %f' % (theta0,
theta1, theta2, error1) )

print('Done: theta0 : %f, theta1 : %f, theta2 : %f' % (theta0, theta1,
theta2)  )
print('迭代次数: %d' % cnt  )
```

根据梯度下降算法地不断更新，误差越来越小直至循环跳出，结果输出：

```
theta0 : 2.782632, theta1 : 3.207850, theta2 : 7.998823, error1 : 5997.941160
theta0 : 4.254302, theta1 : 3.809652, theta2 : 11.972218, error1 : 3688.116951
theta0 : 5.154766, theta1 : 3.351648, theta2 : 14.188535, error1 : 2889.123934
theta0 : 5.800348, theta1 : 2.489862, theta2 : 15.617995, error1 : 2490.307286
theta0 : 6.326710, theta1 : 1.500854, theta2 : 16.676947, error1 : 2228.380594
theta0 : 6.792409, theta1 : 0.499552, theta2 : 17.545335, error1 : 2028.776801
… …
theta0 : 97.717864, theta1 : -13.224347, theta2 : 1.342491, error1 : 58.732358
theta0 : 97.718558, theta1 : -13.224339, theta2 : 1.342271, error1 : 58.732258
theta0 : 97.719251, theta1 : -13.224330, theta2 : 1.342051, error1 : 58.732157
Done: theta0 : 97.719942, theta1 : -13.224322, theta2 : 1.341832
迭代次数: 2608
```