ICA

1. 实验目的

了解 ICA 算法的原理，并且可以简单应用

2. 算法原理

1)简介

在高维数据处理中，为了简化计算量以及储存空间，需要对这些高维数据进行一定程度上的降维，并尽量保证数据的不失真。PCA 和 ICA 是两种常用的降维方法。

PCA：principal component analysis ，主成分分析

ICA ： Independent component analysis，独立成分分析

PCA,ICA 都是统计理论当中的概念，在机器学习当中应用很广，比如图像，语音，通信的分析处理。

从线性代数的角度去理解，PCA 和 ICA 都是要找到一组基，这组基张成一个特征空间，数据的处理就都需要映射到新空间中去。

两者常用于机器学习中提取特征后的降维操作

ICA 是找出构成信号的相互独立部分(不需要正交)，对应高阶统计量分析。ICA 理论认为用来观测的混合数据阵 X 是由独立元 S 经过 A 线性加权获得。ICA 理论的目标就是通过 X 求得一个分离矩阵 W，使得 W 作用在 X 上所获得的信号 Y 是独立源 S 的最优逼近，该关系可以通过下式表示：

Y = WX = WAS ， A = inv(W)

ICA 相比与 PCA 更能刻画变量的随机统计特性，且能抑制高斯噪声。

2）ICA 算法

ICA 算法归功于 Bell 和 Sejnowski，这里使用最大似然估计来解释算法，原始的论文中使用的是一个复杂的方法 Infomax principal。

我们假定每个s_i有概率密度p_s，那么给定时刻原信号的联合分布就是

$$p(s) = \prod_{i=1}^{n} p_s(s_i)$$

这个公式代表一个假设前提：每个人发出的声音信号各自独立。有了 p(s)，我们可以求得 p(x)

$$p(x) = p_s(Wx)|W| = |W| \prod_{i=1} np_s(w_i Tx)$$

左边是每个采样信号 x（n 维向量）的概率，右边是每个原信号概率的乘积的|W|倍。前面提到过，如果没有先验知识，我们无法求得 W 和 s。因此我们需要知道 $p_s(s_i)$，我们打算选取一个概率密度函数赋给 s，但是我们不能选取高斯分布的密度函数。在概率论里我们知道密度函数 p(x)由累计分布函数（cdf）F(x)求导得到。F(x)要满足两个性质是：单调递增和在[0,1]。我们发现 sigmoid 函数很适合，定义域负无穷到正无穷，值域 0 到 1，缓慢递增。我们假定 s 的累积分布函数符合 sigmoid 函数

$$g(s) = \frac{1}{1 + e^{-s}}$$

求导后

$$p_s(s) = g'(s) = \frac{es}{(1 + es)2}$$

这就是 s 的密度函数。这里 s 是实数。

如果我们预先知道 s 的分布函数，那就不用假设了，但是在缺失的情况下，sigmoid 函数能够在大多数问题上取得不错的效果。由于上式中

$p_s(s)$是个对称函数，因此 E[s]=0（s 的均值为 0），那么 E[x]=E[As]=0，x 的均值也是 0。知道了 $p_s(s)$，就剩下 W 了。给定采样后的训练样本 $x(i)(x_1(i),x_2(i),....x_n(i));i=1,...,m$，样本对数似然估计如下：

使用前面得到的 x 的概率密度函数，得

$$l(W) = \sum_{i=1} m(\sum_{j=1} nlog; g'(w_j T x^{(i)}) + log|W|)$$

大括号里面是 $p(x^{(i)})$。

接下来就是对 W 求导了，这里牵涉一个问题是对行列式|W|进行求导的方法，属于矩阵微积分。这里先给出结果，在文章最后再给出推导公式。

$$\nabla_W|W| = |W|(W^{-1})^T$$

最终得到的求导后公式如下，$logg'(s)$的导数为 $1-2g(s)$

（可以自己验证）：

$$W := W + \alpha \left(\begin{bmatrix} 1 - 2g(w_1^T x^{(i)}) \\ 1 - 2g(w_2^T x^{(i)}) \\ \vdots \\ 1 - 2g(w_n^T x^{(i)}) \end{bmatrix} x^{(i)T} + (W^T)^{-1} \right),$$

其中α是梯度上升速率，人为指定。

当迭代求出 W 后，便可得到 $s(i)=Wx(i)$来还原出原始信号。

注意：我们计算最大似然估计时，假设了 $x(i)$与 $y(i)$之间是独立的，然而对于语音信号或者其他具有时间连续依赖特性（比如温度）上，这个假设不能成立。但是在数据足够多时，假设独立对效果影响不大，同时如果事先打乱样例，并运行随机梯度上升算法，那么能够加快收敛速度。

回顾一下鸡尾酒宴会问题，s 是人发出的信号，是连续值，不同时间点的 s 不同，每个人发出的信号之间独立（s_i 和 s_j 之间独立）。s 的累计概率分布函数是 sigmoid 函数，但是所有人发出声音信号都符合这个分布。A（W 的逆阵）代表了 s 相对于 x 的位置变化，x 是 s 和 A 变化后的结果。

3. 实验环境

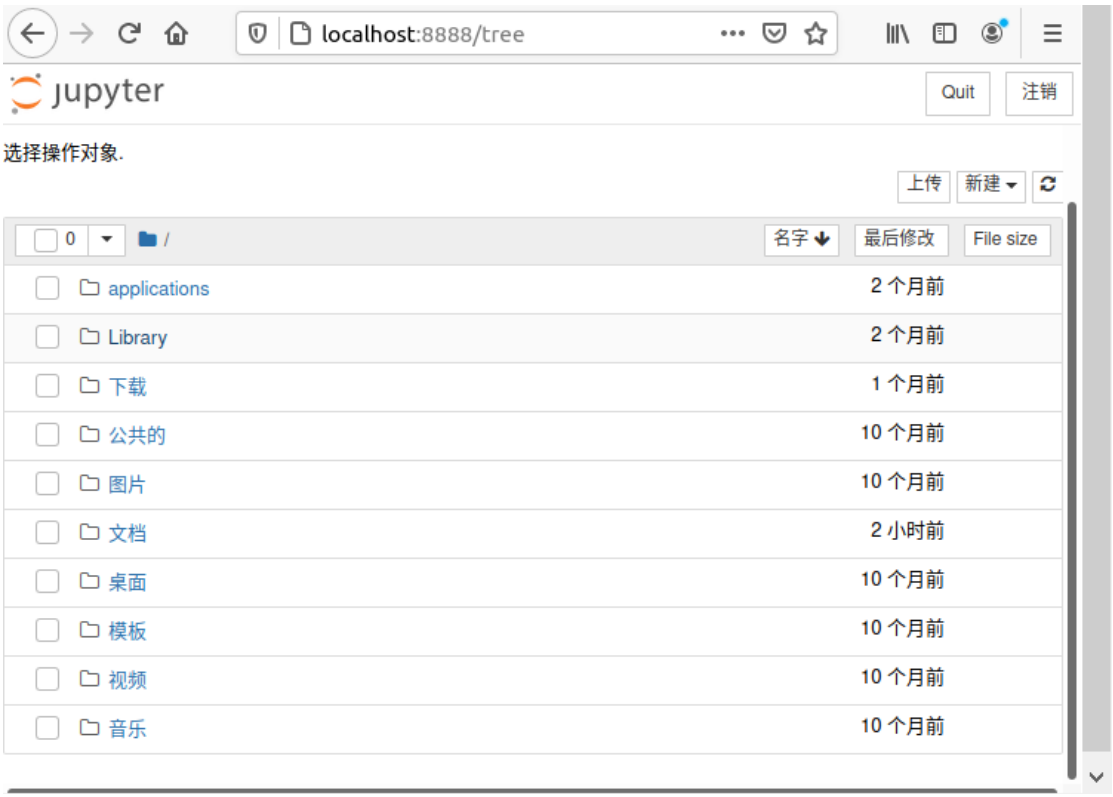
Ubuntu 20.04

Python 3.6

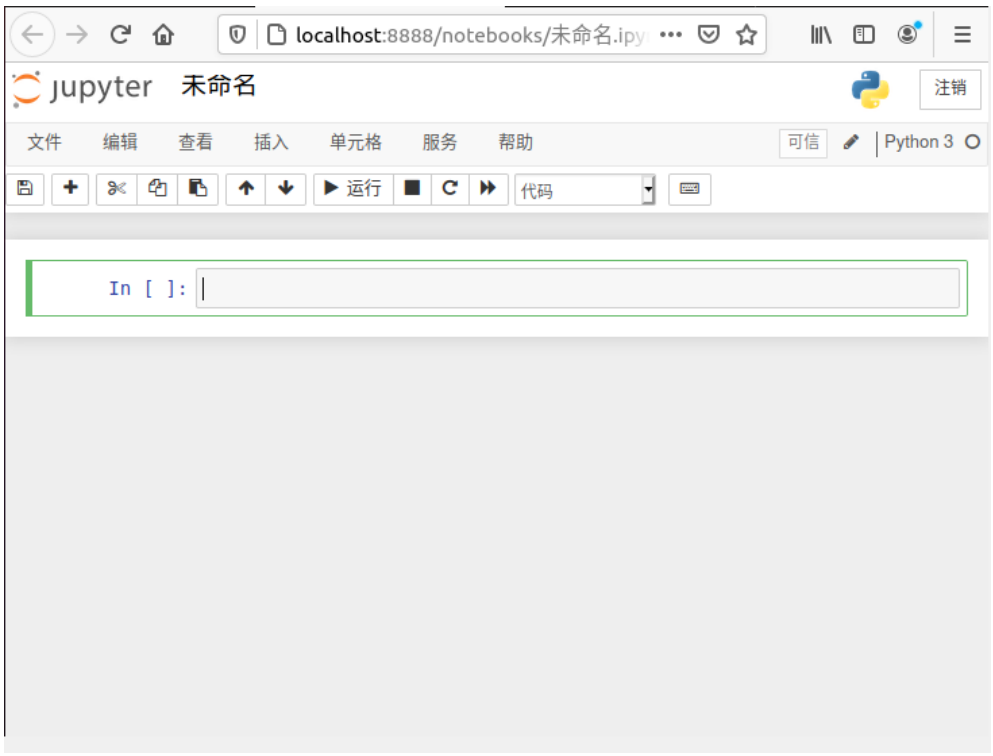
Jupyter notebook

4. 实验步骤

1)打开终端，然后输入 jupyter notebook，出现如下界面



2)选定特定文件夹，新建 ipynb 文件，在未命名出可重命名文件



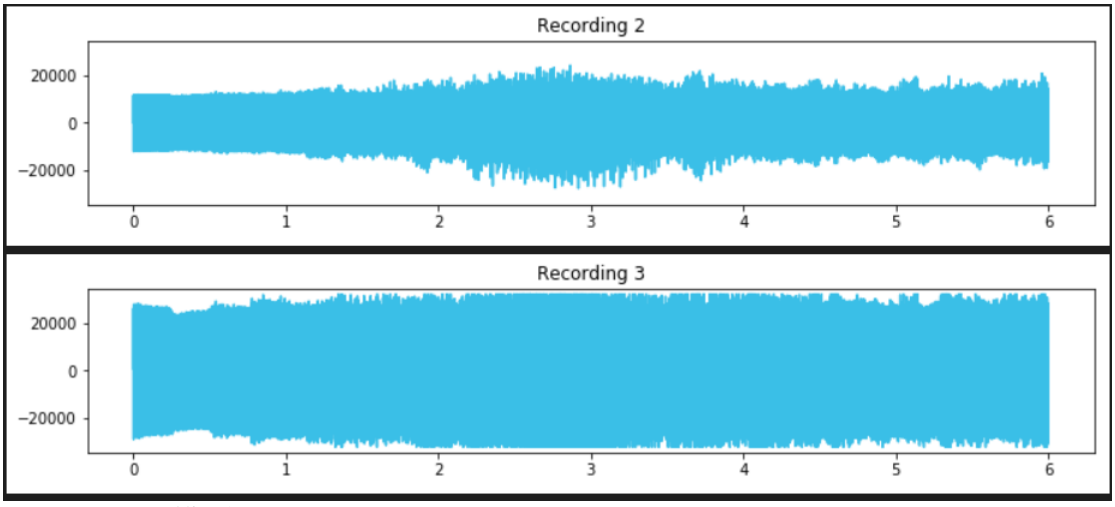
5. 实操

Step 1:数据预处理

1. 导入库
2. 导入数据集
3. 提取音频
4. 分割数据集
5. 数据集标准化

```
#导入库
import numpy as np
import wave

#导入数据集
mix_1_wave = wave.open('ICA mix 1.wav','r')



# Extract Raw Audio from Wav File
signal_1_raw = mix_1_wave.readframes(-1)
signal_1 = np.fromstring(signal_1_raw, 'Int16')

import matplotlib.pyplot as plt

fs = mix_1_wave.getframerate()
timing = np.linspace(0, len(signal_1)/fs, num=len(signal_1))


plt.figure(figsize=(12,2))
```

```
plt.title('Recording 1')
plt.plot(timing,signal_1, c="#3ABFE7")
plt.ylim(-35000, 35000)
plt.show()

mix_2_wave = wave.open('ICA mix 2.wav','r')

#Extract Raw Audio from Wav File
signal_raw_2 = mix_2_wave.readframes(-1)
signal_2 = np.fromstring(signal_raw_2, 'Int16')


mix_3_wave = wave.open('ICA mix 3.wav','r')

#Extract Raw Audio from Wav File
signal_raw_3 = mix_3_wave.readframes(-1)
signal_3 = np.fromstring(signal_raw_3, 'Int16')


plt.figure(figsize=(12,2))
plt.title('Recording 2')
plt.plot(timing,signal_2, c="#3ABFE7")
plt.ylim(-35000, 35000)
plt.show()


plt.figure(figsize=(12,2))
plt.title('Recording 3')
plt.plot(timing,signal_3, c="#3ABFE7")
plt.ylim(-35000, 35000)
plt.show()
```



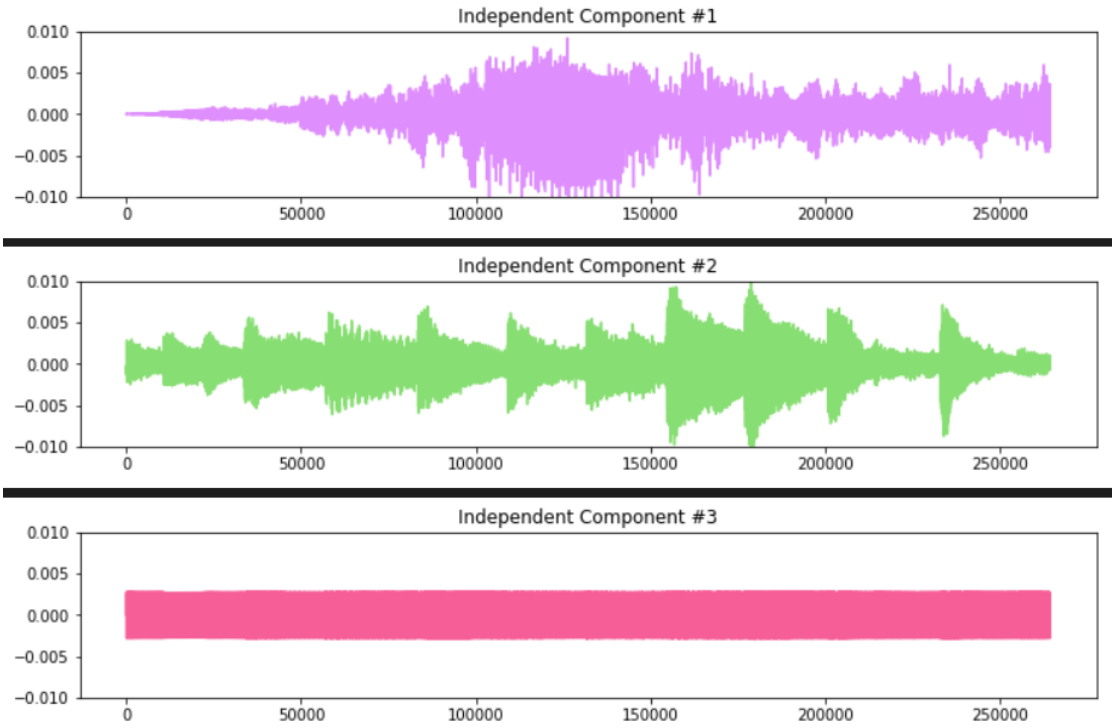
Step 2:ICA 模型

- 实例化 ICA

```
# TODO: Import FastICA
from sklearn.decomposition import FastICA

# TODO: Initialize FastICA with n_components=3
ica = FastICA(n_components=3)

# TODO: Run the FastICA algorithm using fit_transform on dataset X
ica_result = ica.fit_transform(X)
ica_result.shape
```

Step 3:独立成分绘制

```
# Plot Independent Component #1
plt.figure(figsize=(12,2))
plt.title('Independent Component #1')
plt.plot(result_signal_1, c="#df8efd")
plt.ylim(-0.010, 0.010)
plt.show()

# Plot Independent Component #2
plt.figure(figsize=(12,2))
plt.title('Independent Component #2')
plt.plot(result_signal_2, c="#87de72")
plt.ylim(-0.010, 0.010)
plt.show()

# Plot Independent Component #3
plt.figure(figsize=(12,2))
plt.title('Independent Component #3')
plt.plot(result_signal_3, c="#f65e97")
plt.ylim(-0.010, 0.010)
plt.show()
```



Step 4：保存文件

```
from scipy.io import wavfile

# Convert to int, map the appropriate range, and increase the volume a
little bit
result_signal_1_int = np.int16(result_signal_1*32767*100)
result_signal_2_int = np.int16(result_signal_2*32767*100)
result_signal_3_int = np.int16(result_signal_3*32767*100)


# Write wave files
wavfile.write("result_signal_1.wav", fs, result_signal_1_int)
wavfile.write("result_signal_2.wav", fs, result_signal_2_int)
wavfile.write("result_signal_3.wav", fs, result_signal_3_int)
```