混合高斯模型

1. 实验目的

了解 GMM 算法的原理，并且可以简单应用

2. 算法原理

1）概述

正态分布也叫高斯分布，正太分布的概率密度曲线也叫高斯分布概率曲线，GaussianMixtureModel(混合高斯模型，GMM)。

聚类算法大多数通过相似度来判断，而相似度又大多采用欧式距离长短作为衡量依据。而 GMM 采用了新的判断依据：概率，即通过属于某一类的概率大小来判断最终的归属类别。

GMM 的基本思想就是：任意形状的概率分布都可以用多个高斯分布函数去近似，也就是说 GMM 就是有多个单高斯密度分布（Gaussian）组成的，每个 Gaussian 叫一个"Component"，这些"Component"线性加成在一起就组成了 GMM 的概率密度函数，也就是下面的函数。

2）数学公式

K：模型的个数，即 Component 的个数（聚类的个数）为第 k 个高斯的权重

p（x |k） 则为第 k 个高斯概率密度,其均值为 μk,方差为 σk

上述参数，除了 K 是直接给定之外，其他参数都是通过 EM 算法估算出来的。(有个参数是指定 EM 算法参数的)

3）GaussianMixtureModel 算法函数

a）from sklearn.mixture.GaussianMixture

b）主要参数

n_components ：高斯模型的个数，即聚类的目标个数

covariance_type ：通过 EM 算法估算参数时使用的协方差类型，默认是"full"

full：每个模型使用自己的一般协方差矩阵

tied：所用模型共享一个一般协方差矩阵

diag：每个模型使用自己的对角线协方差矩阵

spherical：每个模型使用自己的单一方差

3. 实验环境

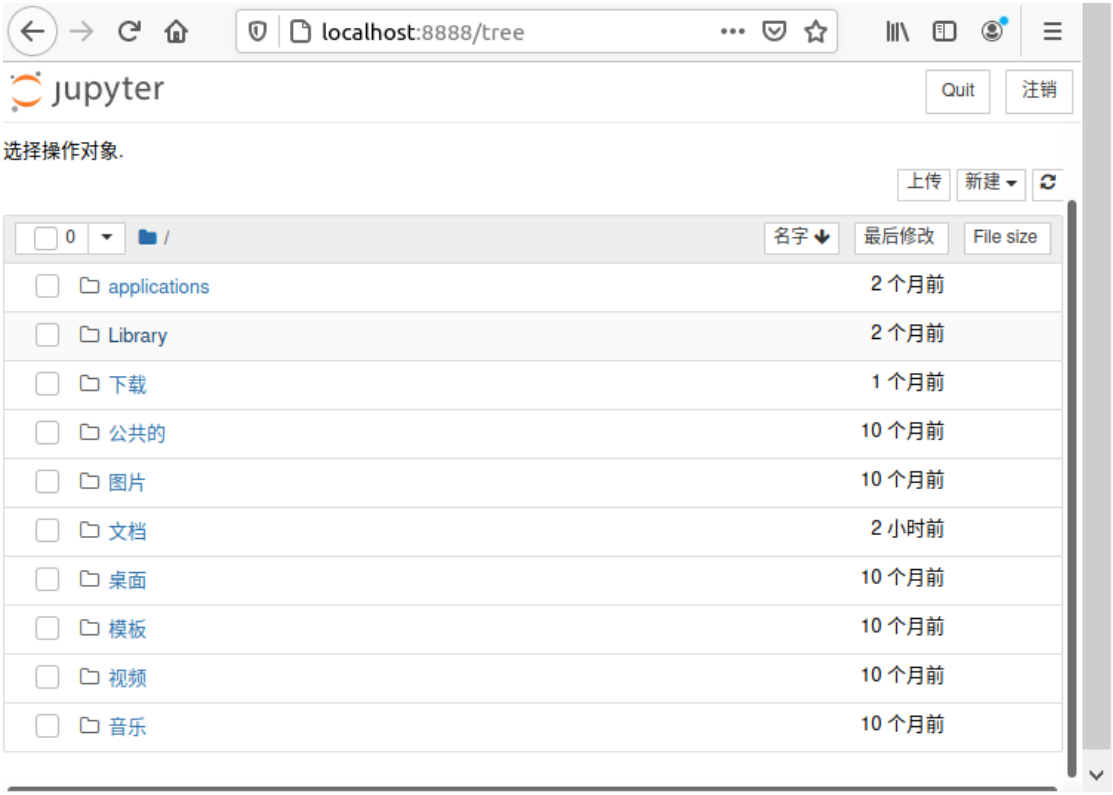
Ubuntu 20.04

Python 3.6

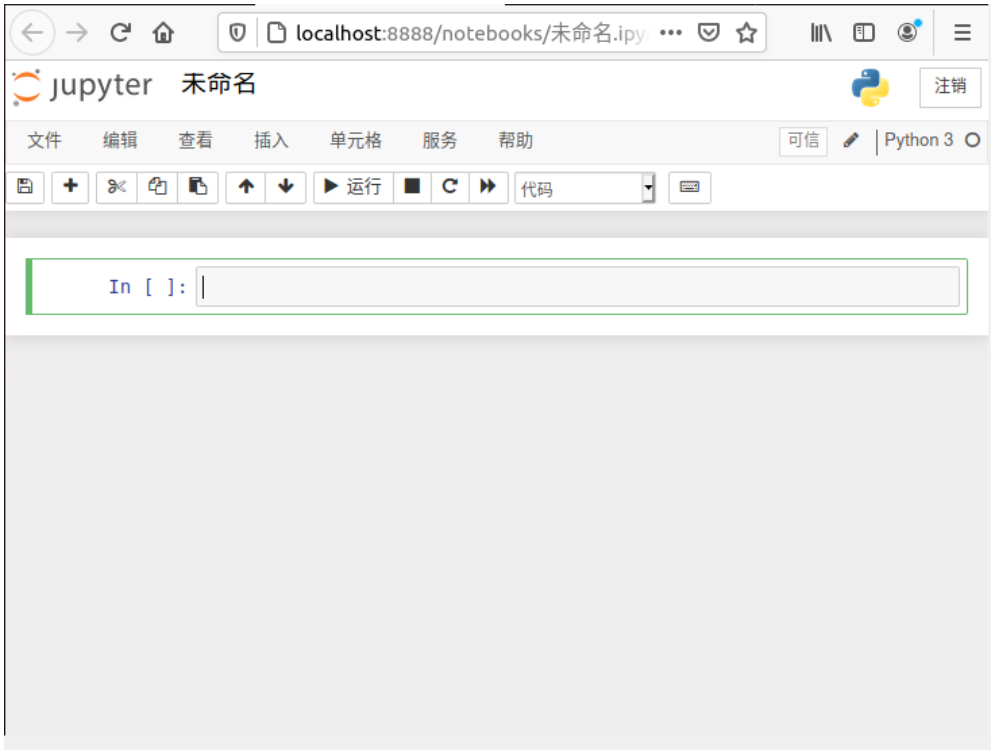
Jupyter notebook

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. 绘制函数
3. 读取数据集
4. 初始化

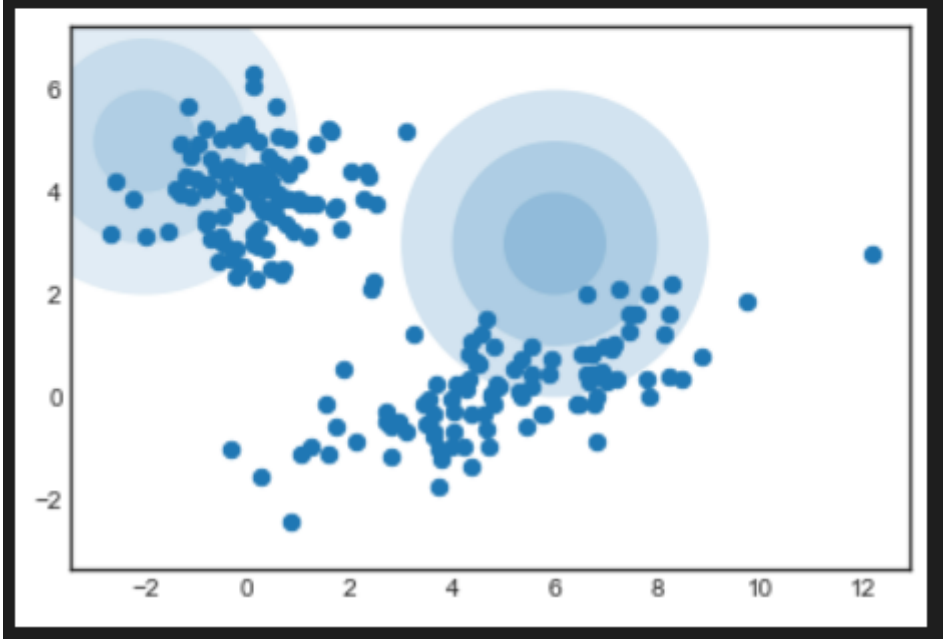
5. 数据集标准化

```
#导入库
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

from sys import maxsize

maxint = maxsize

sns.set_style("white")

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import warnings
warnings.filterwarnings('ignore')

%matplotlib inline

#绘制函数
from matplotlib.patches import Ellipse

def draw_ellipse(position, covariance, ax=None, **kwargs):
    """Draw an ellipse with a given position and covariance"""
    ax = ax or plt.gca()

    # Convert covariance to principal axes
    if covariance.shape == (2, 2):
        U, s, Vt = np.linalg.svd(covariance)
        angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
        width, height = 2 * np.sqrt(s)
    else:
        angle = 0
        width, height = 2 * np.sqrt(covariance)

    # Draw the Ellipse
    for nsig in range(1, 4):
        ax.add_patch(Ellipse(position, nsig * width, nsig * height,
                             angle, **kwargs))

def plot_gmm(gmm, X, label=[], ax=None):
    ax = ax or plt.gca()
```

```
    if len(label) > 0:
        labels = label
        ax.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis', zo
rder=2)
    else:
        ax.scatter(X[:, 0], X[:, 1], s=40, zorder=2)
    ax.axis('equal')

    ws = [value for g in gmm for (param, value) in g.items() if param =
= 'w']
    w_factor = 0.2 /np.max(ws)
    for param in gmm:
        pos = param['mu']
        covar = param['Covar']
        w = param['w']

        draw_ellipse(pos, covar, alpha=w * w_factor)


# 读取数据集
df = pd.read_csv("bi_dimensional_n_bi_modal_data.csv", index_col=False)

df.sample(5).head(n=5)

fig = plt.figure()
plt.scatter(df['x'], df['y'], 24, c=df['label'])


# 初始化
initial_guess = [{ 'mu': np.asarray([-2.0,5.0]),
          'Covar': np.asarray([ [1.0, 0], [0, 1.0] ]),
          'w': 0.4 },
          {'mu': np.asarray([6.0,3.0]),
          'Covar': np.asarray([ [1.0, 0], [0, 1.0] ]),
          'w': 0.6}]

iters = 0
df_copy = df.copy()

df_copy.sample(5).head(5)

labels = df_copy.label
X = df[['x','y']].values
```

```
plot_gmm(initial_guess, X)
```

	label	x	y
68	1	-0.138802	4.271919
176	2	3.969816	-0.933927
90	1	-0.723951	4.658043
142	2	7.434391	1.634299
195	2	5.535699	0.237943



Step 2:GMM 模型

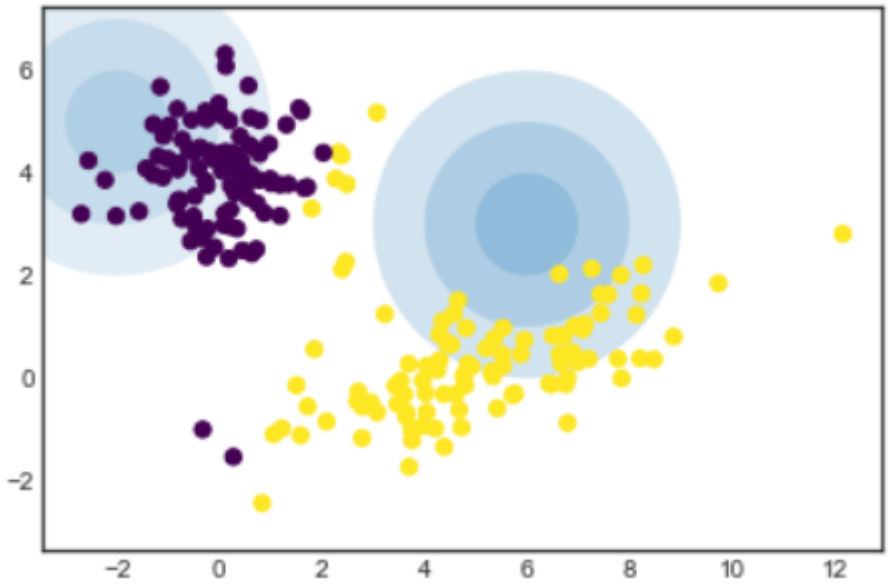
```
# Gaussian probability density function
def gaussian_prob(x, mu, covar):

    covar_inv = np.linalg.inv(covar)

    d = mu.shape[0]
    det_covar = np.linalg.det(covar)
    norm_const = 1.0/np.sqrt(np.power(2*np.pi,d))/det_covar

    a = x - mu
    a_times_covar_inv = np.matmul(a, covar_inv)

    p = norm_const* np.exp(-0.5* np.matmul(a_times_covar_inv,a.transpos
e()))
```

```
    return p

# assign every data point to its most likely cluster
def expectation(dataFrame, gmm):


    param = gmm[0]
    mu1 = param['mu']
    covar1 = param['Covar']
    w1 = param['w']

    param = gmm[1]
    mu2 = param['mu']
    covar2 = param['Covar']
    w2 = param['w']

    for i in range(dataFrame.shape[0]):

        x = df[['x','y']].values[i]

        # probability that a point came from a Guassian with given para
meters
        p_cluster1 = w1*gaussian_prob(x, mu1, covar1)
        p_cluster2 = w2*gaussian_prob(x, mu2, covar2)


        if p_cluster1 > p_cluster2:
            dataFrame['label'][i] = 1
        else:
            dataFrame['label'][i] = 2

    return dataFrame
```

Step 3:迭代

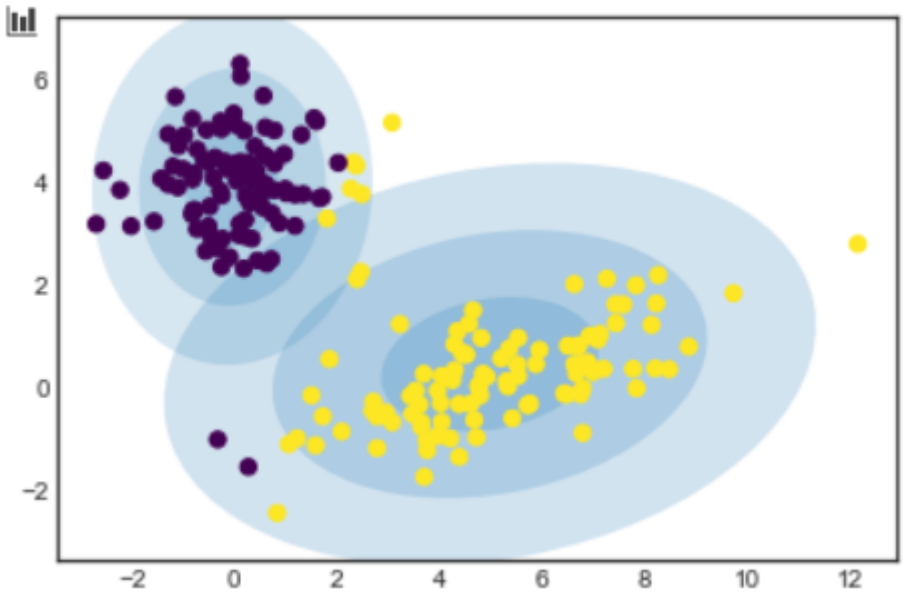
```
%%time
iters += 1

# E-step
updated_labels = expectation(df_copy, initial_guess)

updated_labels.sample(5).head(5)

labels = updated_labels.label
```

```
X = df[['x','y']].values
plot_gmm(initial_guess, X, label=labels)
```




```
# update estimates of lambda, mu and sigma
def maximization(dataFrame, gmm):

    points_assigned_to_cluster1 = dataFrame[dataFrame['label'] == 1]
    points_assigned_to_cluster2 = dataFrame[dataFrame['label'] == 2]

    percent_assigned_to_cluster1 = len(points_assigned_to_cluster1) / f
loat(len(dataFrame))
    percent_assigned_to_cluster2 = 1 - percent_assigned_to_cluster1

    X1 = points_assigned_to_cluster1[['x','y']].values
    mu1 = np.mean(X1,axis=0)
    covar1 = np.cov(X1.transpose())

    X2 = points_assigned_to_cluster2[['x','y']].values
    mu2 = np.mean(X2,axis=0)
    covar2 = np.cov(X2.transpose())


    gmm = [{ 'mu': mu1,
          'Covar': covar1,
          'w': percent_assigned_to_cluster1},
          {'mu': mu2,
          'Covar': covar2,
          'w': percent_assigned_to_cluster2}]
```

```
    return gmm

# get the distance between points
def distance(old_params, new_params):

    dist = 0

    dmu1 = np.linalg.norm(new_params[0]['mu'] - old_params[0]['mu'])
    dmu2 = np.linalg.norm(new_params[1]['mu'] - old_params[1]['mu'])

    return [dmu1,dmu2]

%%time

# M-step
new_gmm = maximization(updated_labels, initial_guess)

# see if our estimates of mu have changed
mu_shift = distance(initial_guess, new_gmm)


print('New gmm: {}'.format(new_gmm))
print('mu_shift:{}'.format(mu_shift))

plot_gmm(new_gmm, X, label=labels)
```




```
%%time
iters += 1
```

```
# E-step
updated_labels = expectation(df_copy, new_gmm)

# M-step
prv_gmm = new_gmm.copy()
new_gmm = maximization(updated_labels, prv_gmm)

# see if our estimates of mu have changed
mu_shift = distance(prv_gmm, new_gmm)
print('New gmm: {}'.format(new_gmm))
print('mu_shift:{}'.format(mu_shift))

plot_gmm(new_gmm, X, label=labels)
```