AI学习笔记--skLearn--GMM

- ## 概述

GMM（Gaussian Mixture Model），高斯混合模型，也可以简写为MOG（Mixture of Gaussian）。高斯模型就是用高斯概率密度函数（正态分布曲线）精确地量化事物，将一个事物分解为若干的基于高斯概率密度函数（正态分布曲线）形成的模型。主要来源于Chris Stauffer W.E.L Grimson的论文 （如下）。经典的使用场景是对数据进行预测、聚类。

“Adaptive background mixture models for real-time tracking”

- ## 算法原理和过程

对图像背景建立高斯模型的原理及过程：图像灰度直方图反映的是图像中某个灰度值出现的频次，也可以认为是图像灰度概率密度的估计。如果图像所包含的目标区域和背景区域相比比较大，且背景区域和目标区域在灰度上有一定的差异，那么该图像的灰度直方图呈现双峰-谷形状，其中一个峰对应于目标，另一个峰对应于背景的中心灰度。对于复杂的图像，尤其是医学图像，一般是多峰的。通过将直方图的多峰特性看作是多个高斯分布的叠加，可以解决图像的分割问题。 在智能监控系统中，对于运动目标的检测是中心内容，而在运动目标检测提取中，背景目标对于目标的识别和跟踪至关重要。而建模正是背景目标提取的一个重要环节。

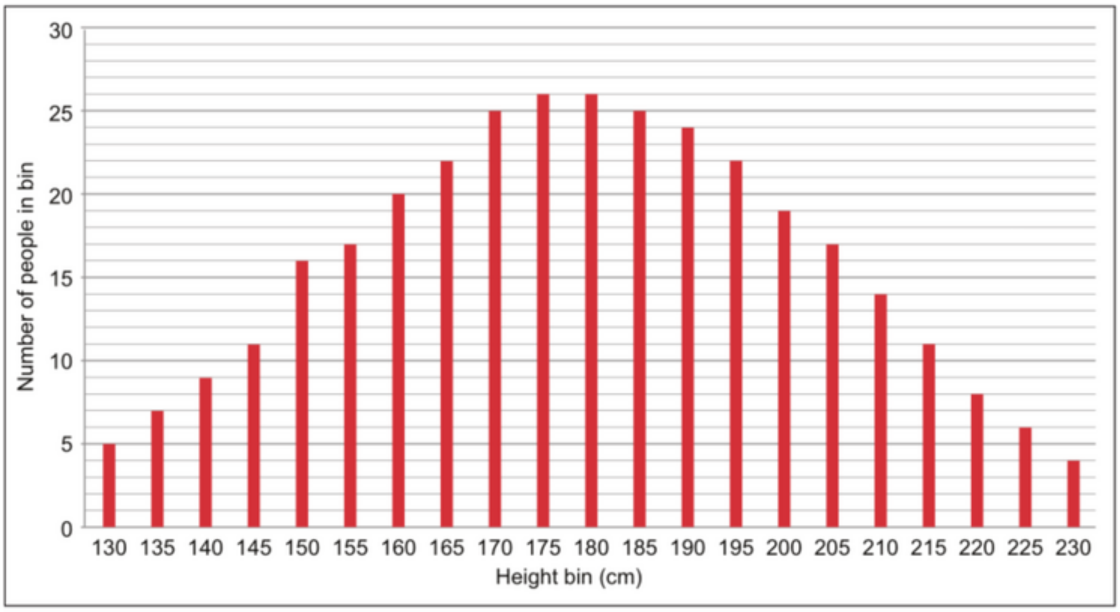


Figure 2.9 Histogram of a normal distribution, in this case the height of 334 fictitious people. The modal (most frequently occurring) bin is centered at 180 cm.

高斯公式具体如下所示：

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

在看一个混合高斯分布的概率密度函数：

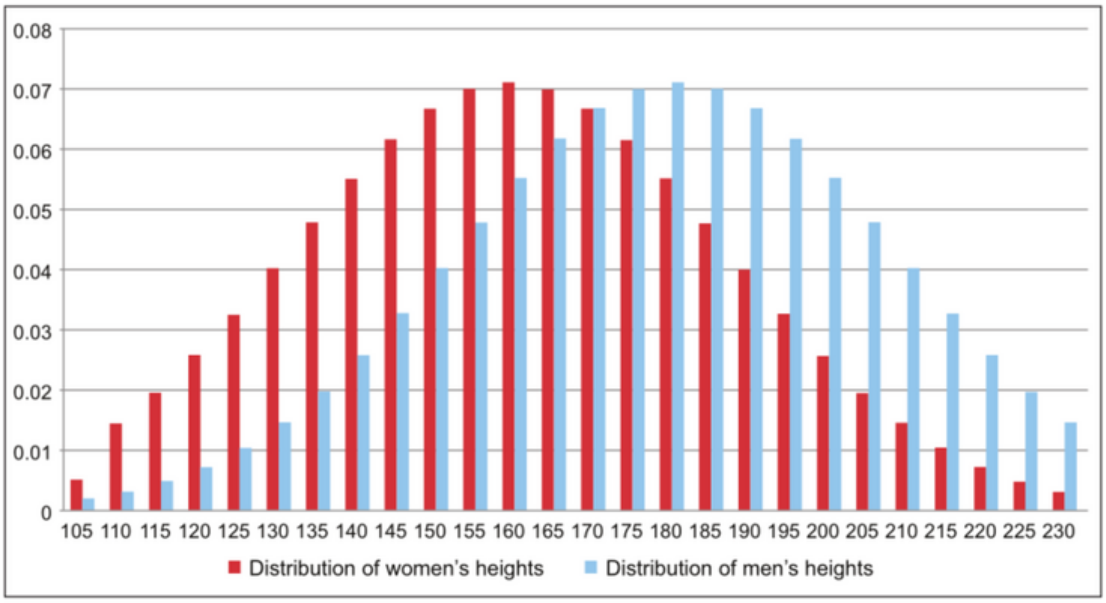


Figure 2.11 Probability distributions of height for men and women. Note that these probabilities are conditioned on the fact that gender is known: so, for example, given that we know a particular person is a woman, her probability of having a height in a particular bucket can be read off the y-axis.

计算公式：

$$Pr(x) = \sum_{k=1}^{K} \pi_k N(x; u_k, \Sigma_k)$$

其中，K需要事先确定好，就像K-means中的K一样，只要K足够大，这个XX Mixture Model就会变得足够复杂，就可以用来逼近任意连续的概率密度分布。πk是权值因子。其中的任意一个高斯分布N(x;uk,Σk) 叫作这个模型的一个component（成分）。GMM是一种聚类算法，每个component就是一个聚类中心。我们知道概率密度函数的前提下，要想得到结果，重要是得到模型的参数，至于模型参数的推导是运用EM算法，EM算法大致流程是：

（1）取参数的初始值开始迭代

（2）E 步：依据当前模型参数，计算分模型 k 对观测数据 y_j 的响应度

$$\hat{\gamma}_{jk}=\frac{\alpha_k\phi(y_j\mid\theta_k)}{\sum_{k=1}^{K}\alpha_k\phi(y_j\mid\theta_k)},\qquad j=1,2,\cdots,N;\quad k=1,2,\cdots,K$$

（3）M 步：计算新一轮迭代的模型参数

$$\hat{\mu}_k=\frac{\sum_{j=1}^{N}\hat{\gamma}_{jk}y_j}{\sum_{j=1}^{N}\hat{\gamma}_{jk}},\qquad k=1,2,\cdots,K$$

$$\hat{\sigma}_k^2=\frac{\sum_{j=1}^{N}\hat{\gamma}_{jk}(y_j-\mu_k)^2}{\sum_{j=1}^{N}\hat{\gamma}_{jk}},\qquad k=1,2,\cdots,K$$

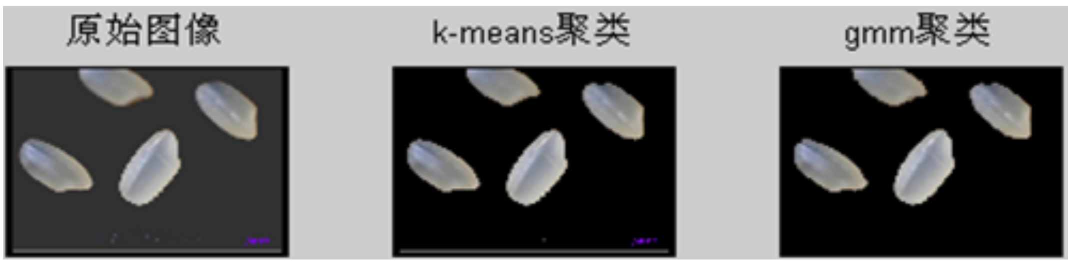
$$\hat{\alpha}_k=\frac{\sum_{j=1}^{N}\hat{\gamma}_{jk}}{N},\qquad k=1,2,\cdots,K$$

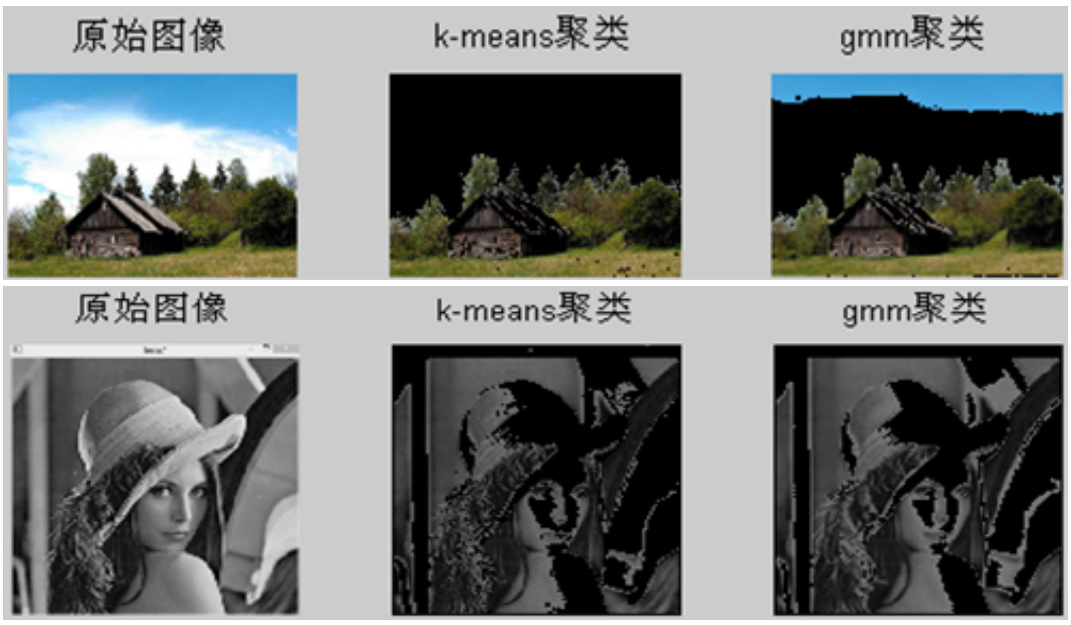
（4）重复第（2）步和第（3）步，直到收敛.

GMM经典的应用有概率密度估计、背景建模、聚类等。混合高斯模型和K-Means算法的主要区别在于：

- K-Means计算出的结果是确定输入为哪一类，混合高斯模型的计算结果为输入为哪一类的概率。

GMM与K-Means在图形分割上的对比：

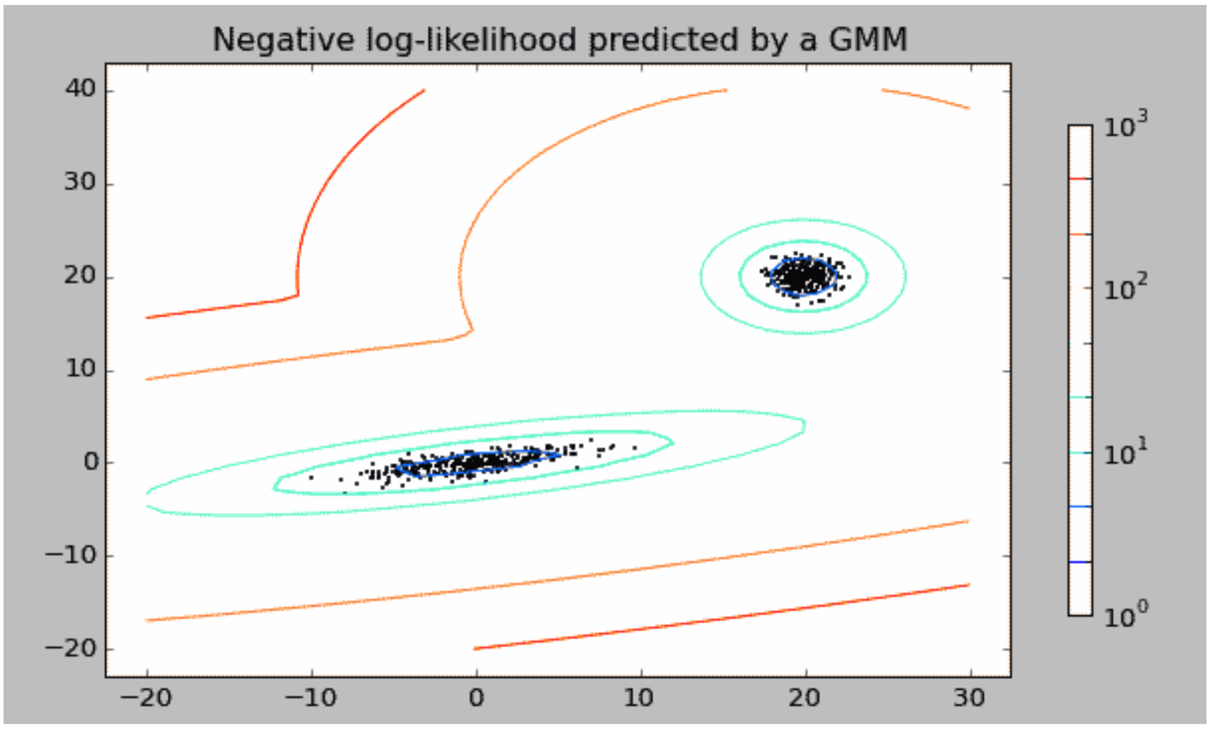




需要注意：

- K-means和GMM用于图像分割由于只考虑了像素的颜色信息，没有考虑空间信息导致其对于复杂背景的效果很差。对于简单背景和前景的颜色分布都比较柔和的情况有较好的效果。
- K-means初始值的选择非常重要。不好的初始值经常会造成较差的聚类效果。
- 应用GMM时，先将3通道彩色图像转换为了灰度图。原因是原始的3个通道数据存在很强的相关性，导致协方差矩阵不可逆。
- 聚类(分割)时需要手动确定类别的数量。类的数量对于聚类效果也有很大的影响。

GMM算法的核心概念是，不论一个数据集合是怎么样的分布情况，都可以用一个或者多个component 高斯拟合而成的一个概率学模型。比如图：



图中我们可以看到存在两个聚类集合核心，我们通过两个单一的高斯模型，把他们拟合到一个分布空间上混合。这个集合了两个单一高斯模型的空间就组成了我们的混合高斯模型，即一个GMM。GMM可以分布到不同的二维空间上，sk-learn也提供多种方式去归纳这个概率模型。

- 算法Demo

最简单的历程：

```
# -*- coding:utf-8 -*-
import numpy as np
from sklearn import mixture
from sklearn.mixture import GaussianMixture
#生成随机观测点，含有2个聚集核心
obs = np.concatenate((np.random.randn(100, 1), 10 + np.random.randn(300,
1)))
clf = mixture.GaussianMixture(n_components=2)
print obs[:10]
clf.fit(obs)
#预测
print clf.predict([[0], [2], [9], [10]])
```

输出结果：

```
[[ 0.67341501]
[ 0.63691391]
[ 2.11450798]
[-1.79399411]
[-1.07718205]
[-0.16423444]
[-1.15966356]
[ 0.9492161 ]
[-1.08082622]
[-0.28620924]]
[1 1 0 0]
```

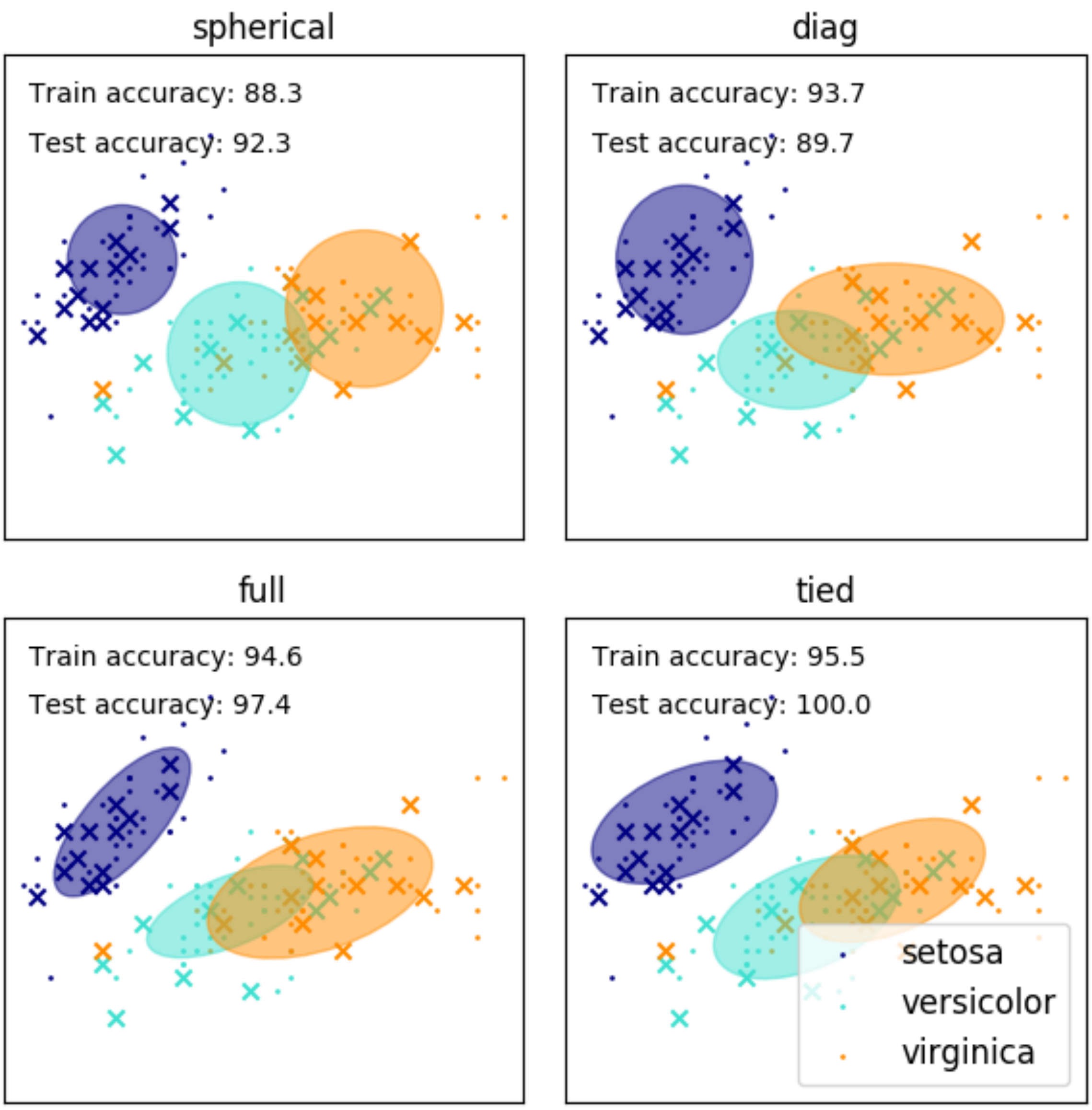
SK-LearnDemo :

```
"""
===============
GMM covariances
===============

Demonstration of several covariances types for Gaussian mixture models.

See :ref:`gmm` for more information on the estimator.

Although GMM are often used for clustering, we can compare the obtained
clusters with the actual classes from the dataset. We initialize the means
of the Gaussians with the means of the classes from the training set to
make
this comparison valid.

We plot predicted labels on both training and held out test data using a
variety of GMM covariance types on the iris dataset.
We compare GMMs with spherical, diagonal, full, and tied covariance
matrices in increasing order of performance. Although one would
expect full covariance to perform best in general, it is prone to
overfitting on small datasets and does not generalize well to held out
test data.

On the plots, train data is shown as dots, while test data is shown as
crosses. The iris dataset is four-dimensional. Only the first two
dimensions are shown here, and thus some points are separated in other
dimensions.
"""

# Author: Ron Weiss <ronweiss@gmail.com>, Gael Varoquaux
# Modified by Thierry Guillemot <thierry.guillemot.work@gmail.com>
# License: BSD 3 clause

import matplotlib as mpl
import matplotlib.pyplot as plt

import numpy as np

from sklearn import datasets
from sklearn.mixture import GaussianMixture
from sklearn.model_selection import StratifiedKFold
```

```
print(__doc__)

colors = ['navy', 'turquoise', 'darkorange']


def make_ellipses(gmm, ax):
    for n, color in enumerate(colors):
        if gmm.covariance_type == 'full':
            covariances = gmm.covariances_[n][:2, :2]
        elif gmm.covariance_type == 'tied':
            covariances = gmm.covariances_[:2, :2]
        elif gmm.covariance_type == 'diag':
            covariances = np.diag(gmm.covariances_[n][:2])
        elif gmm.covariance_type == 'spherical':
            covariances = np.eye(gmm.means_.shape[1]) * gmm.covariances_[n]
        v, w = np.linalg.eigh(covariances)
        u = w[0] / np.linalg.norm(w[0])
        angle = np.arctan2(u[1], u[0])
        angle = 180 * angle / np.pi  # convert to degrees
        v = 2. * np.sqrt(2.) * np.sqrt(v)
        ell = mpl.patches.Ellipse(gmm.means_[n, :2], v[0], v[1],
                                  180 + angle, color=color)
        ell.set_clip_box(ax.bbox)
        ell.set_alpha(0.5)
        ax.add_artist(ell)
        ax.set_aspect('equal', 'datalim')

iris = datasets.load_iris()

# Break up the dataset into non-overlapping training (75%) and testing
# (25%) sets.
skf = StratifiedKFold(n_splits=4)
# Only take the first fold.
train_index, test_index = next(iter(skf.split(iris.data, iris.target)))


X_train = iris.data[train_index]
y_train = iris.target[train_index]
X_test = iris.data[test_index]
y_test = iris.target[test_index]

n_classes = len(np.unique(y_train))

# Try GMMs using different types of covariances.
estimators = dict((cov_type, GaussianMixture(n_components=n_classes,
                   covariance_type=cov_type, max_iter=20, random_state=0))
                  for cov_type in ['spherical', 'diag', 'tied',
'full'])

n_estimators = len(estimators)
```

```
plt.figure(figsize=(3 * n_estimators // 2, 6))
plt.subplots_adjust(bottom=.01, top=0.95, hspace=.15, wspace=.05,
                    left=.01, right=.99)


for index, (name, estimator) in enumerate(estimators.items()):
    # Since we have class labels for the training data, we can
    # initialize the GMM parameters in a supervised manner.
    estimator.means_init = np.array([X_train[y_train == i].mean(axis=0)
                                    for i in range(n_classes)])

    # Train the other parameters using the EM algorithm.
    estimator.fit(X_train)

    h = plt.subplot(2, n_estimators // 2, index + 1)
    make_ellipses(estimator, h)

    for n, color in enumerate(colors):
        data = iris.data[iris.target == n]
        plt.scatter(data[:, 0], data[:, 1], s=0.8, color=color,
                    label=iris.target_names[n])
    # Plot the test data with crosses
    for n, color in enumerate(colors):
        data = X_test[y_test == n]
        plt.scatter(data[:, 0], data[:, 1], marker='x', color=color)

    y_train_pred = estimator.predict(X_train)
    train_accuracy = np.mean(y_train_pred.ravel() == y_train.ravel()) * 100
    plt.text(0.05, 0.9, 'Train accuracy: %.1f' % train_accuracy,
             transform=h.transAxes)

    y_test_pred = estimator.predict(X_test)
    test_accuracy = np.mean(y_test_pred.ravel() == y_test.ravel()) * 100
    plt.text(0.05, 0.8, 'Test accuracy: %.1f' % test_accuracy,
             transform=h.transAxes)

    plt.xticks(())
    plt.yticks(())
    plt.title(name)

plt.legend(scatterpoints=1, loc='lower right', prop=dict(size=12))
plt.show()
```

运行结果：

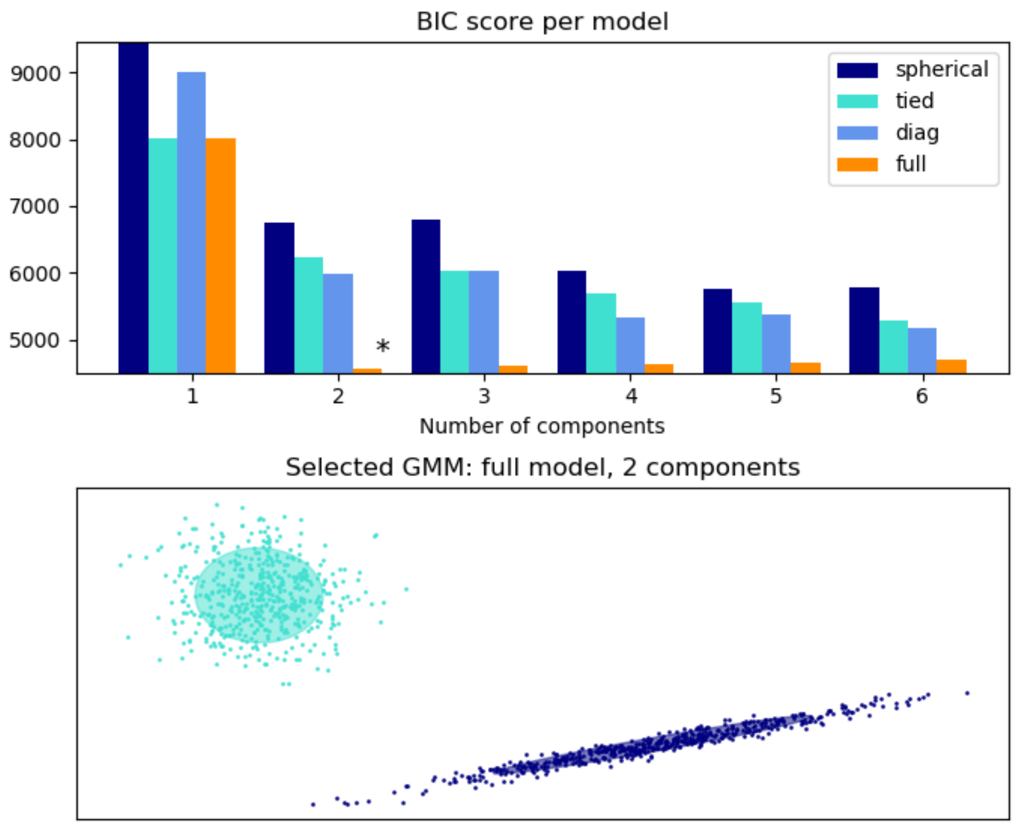


SkLearn提供的另一个demo：

```
"""
================================
Gaussian Mixture Model Selection
================================

This example shows that model selection can be performed with
Gaussian Mixture Models using information-theoretic criteria (BIC).
Model selection concerns both the covariance type
and the number of components in the model.
In that case, AIC also provides the right result (not shown to save time),
but BIC is better suited if the problem is to identify the right model.
Unlike Bayesian procedures, such inferences are prior-free.

In that case, the model with 2 components and full covariance
```

```
(which corresponds to the true generative model) is selected.
"""

import numpy as np
import itertools

from scipy import linalg
import matplotlib.pyplot as plt
import matplotlib as mpl

from sklearn import mixture

print(__doc__)

# Number of samples per component
n_samples = 500

# Generate random sample, two components
np.random.seed(0)
C = np.array([[0., -0.1], [1.7, .4]])
X = np.r_[np.dot(np.random.randn(n_samples, 2), C),
          .7 * np.random.randn(n_samples, 2) + np.array([-6, 3])]

lowest_bic = np.infty
bic = []
n_components_range = range(1, 7)
cv_types = ['spherical', 'tied', 'diag', 'full']
for cv_type in cv_types:
    for n_components in n_components_range:
        # Fit a Gaussian mixture with EM
        gmm = mixture.GaussianMixture(n_components=n_components,
                                      covariance_type=cv_type)
        gmm.fit(X)
        bic.append(gmm.bic(X))
        if bic[-1] < lowest_bic:
            lowest_bic = bic[-1]
            best_gmm = gmm

bic = np.array(bic)
color_iter = itertools.cycle(['navy', 'turquoise', 'cornflowerblue',
                              'darkorange'])
clf = best_gmm
bars = []

# Plot the BIC scores
plt.figure(figsize=(8, 6))
spl = plt.subplot(2, 1, 1)
for i, (cv_type, color) in enumerate(zip(cv_types, color_iter)):
    xpos = np.array(n_components_range) + .2 * (i - 2)
    bars.append(plt.bar(xpos, bic[i * len(n_components_range):
                                  (i + 1) * len(n_components_range)],
                                width=.2, color=color))
plt.xticks(n_components_range)
plt.ylim([bic.min() * 1.01 - .01 * bic.max(), bic.max()])
plt.title('BIC score per model')
xpos = np.mod(bic.argmin(), len(n_components_range)) + .65 +\
    .2 * np.floor(bic.argmin() / len(n_components_range))
plt.text(xpos, bic.min() * 0.97 + .03 * bic.max(), '*', fontsize=14)
spl.set_xlabel('Number of components')
spl.legend([b[0] for b in bars], cv_types)

# Plot the winner
splot = plt.subplot(2, 1, 2)
Y_ = clf.predict(X)
for i, (mean, cov, color) in enumerate(zip(clf.means_, clf.covariances_,
                                           color_iter)):
    v, w = linalg.eigh(cov)
    if not np.any(Y_ == i):
        continue
     plt.scatter(X[Y_ == i, 0], X[Y_ == i, 1], .8, color=color)

    # Plot an ellipse to show the Gaussian component
    angle = np.arctan2(w[0][1], w[0][0])
    angle = 180. * angle / np.pi  # convert to degrees
    v = 2. * np.sqrt(2.) * np.sqrt(v)
    ell = mpl.patches.Ellipse(mean, v[0], v[1], 180. + angle, color=color)
    ell.set_clip_box(splot.bbox)
    ell.set_alpha(.5)
    splot.add_artist(ell)

plt.xticks(())
plt.yticks(())
plt.title('Selected GMM: full model, 2 components')
plt.subplots_adjust(hspace=.35, bottom=.02)
plt.show()
```

运行结果:



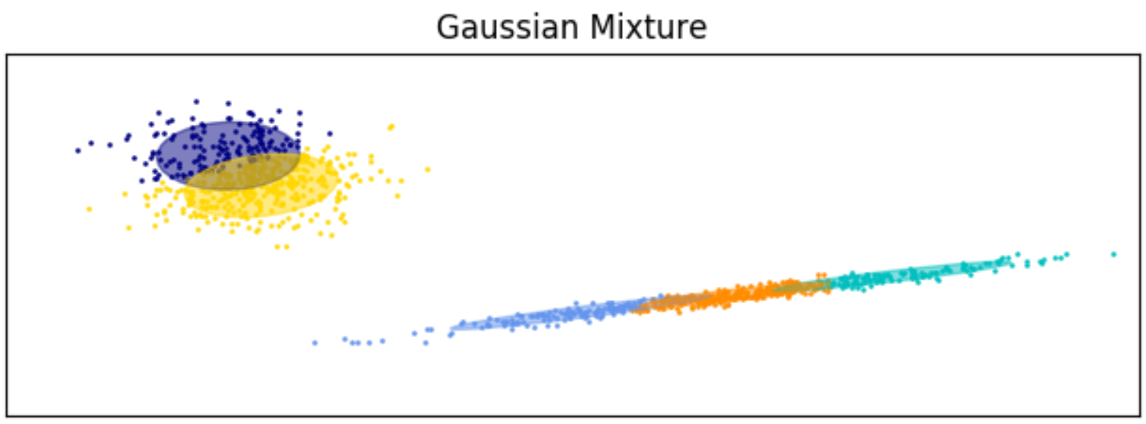
同时，有一种优化算法：

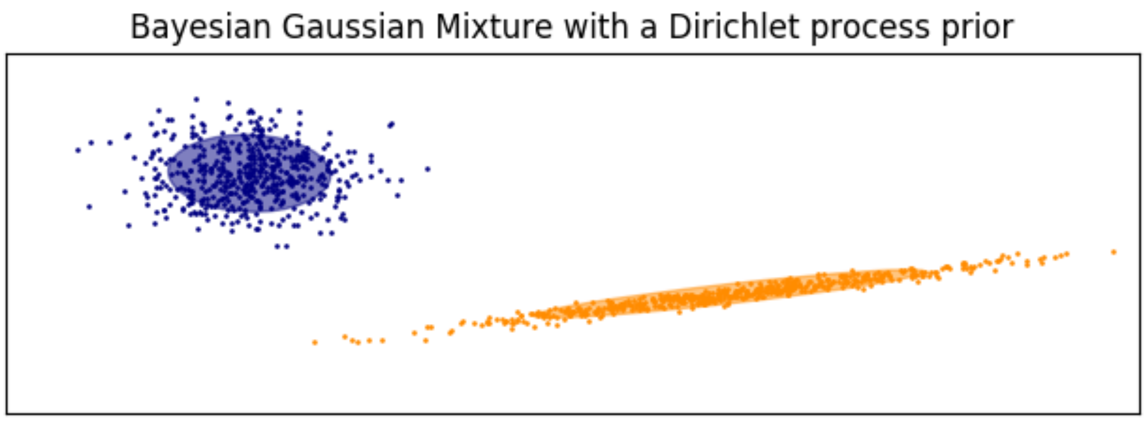
```
"""
=================================
Gaussian Mixture Model Ellipsoids
=================================

Plot the confidence ellipsoids of a mixture of two Gaussians
obtained with Expectation Maximisation (``GaussianMixture`` class) and
Variational Inference (``BayesianGaussianMixture`` class models with
a Dirichlet process prior).

Both models have access to five components with which to fit the data. Note
that the Expectation Maximisation model will necessarily use all five
components while the Variational Inference model will effectively only use
as
many as are needed for a good fit. Here we can see that the Expectation
Maximisation model splits some components arbitrarily, because it is trying
to
fit too many components, while the Dirichlet Process model adapts it number
of
state automatically.

This example doesn't show it, as we're in a low-dimensional space, but
```

```
another advantage of the Dirichlet process model is that it can fit
full covariance matrices effectively even when there are less examples
per cluster than there are dimensions in the data, due to
regularization properties of the inference algorithm.
"""

import itertools

import numpy as np
from scipy import linalg
import matplotlib.pyplot as plt
import matplotlib as mpl

from sklearn import mixture

color_iter = itertools.cycle(['navy', 'c', 'cornflowerblue', 'gold',
                              'darkorange'])


def plot_results(X, Y_, means, covariances, index, title):
    splot = plt.subplot(2, 1, 1 + index)
    for i, (mean, covar, color) in enumerate(zip(
            means, covariances, color_iter)):
        v, w = linalg.eigh(covar)
        v = 2. * np.sqrt(2.) * np.sqrt(v)
        u = w[0] / linalg.norm(w[0])
        # as the DP will not use every component it has access to
        # unless it needs it, we shouldn't plot the redundant
        # components.
        if not np.any(Y_ == i):
            continue
         plt.scatter(X[Y_ == i, 0], X[Y_ == i, 1], .8, color=color)

        # Plot an ellipse to show the Gaussian component
        angle = np.arctan(u[1] / u[0])
        angle = 180. * angle / np.pi  # convert to degrees
        ell = mpl.patches.Ellipse(mean, v[0], v[1], 180. + angle,
color=color)
        ell.set_clip_box(splot.bbox)
        ell.set_alpha(0.5)
        splot.add_artist(ell)

    plt.xlim(-9., 5.)
    plt.ylim(-3., 6.)
    plt.xticks(())
    plt.yticks(())
    plt.title(title)


# Number of samples per component
n_samples = 500
```

```
# Generate random sample, two components
np.random.seed(0)
C = np.array([[0., -0.1], [1.7, .4]])
X = np.r_[np.dot(np.random.randn(n_samples, 2), C),
          .7 * np.random.randn(n_samples, 2) + np.array([-6, 3])]

# Fit a Gaussian mixture with EM using five components
gmm = mixture.GaussianMixture(n_components=5,
covariance_type='full').fit(X)
plot_results(X, gmm.predict(X), gmm.means_, gmm.covariances_, 0,
             'Gaussian Mixture')

# Fit a Dirichlet process Gaussian mixture using five components
dpgmm = mixture.BayesianGaussianMixture(n_components=5,
                                        covariance_type='full').fit(X)
plot_results(X, dpgmm.predict(X), dpgmm.means_, dpgmm.covariances_, 1,
             'Bayesian Gaussian Mixture with a Dirichlet process
prior')

plt.show()
```

运行结果：


Gaussian Mixture


Bayesian Gaussian Mixture with a Dirichlet process prior