**主成分分析和聚类**

1. ****实验目标****

掌握主成成分分析PCA法，在无监督学习中完成降维，

1. ****实验内容****

* PCA 降维
* K-Means

1. ****实验步骤及结果截图****

* 莺尾花数据集

1. 首先导入所有实验所用到的基本模块。

from mpl\_toolkits.mplot3d

import Axes3D

from sklearn

import datasets

from sklearn

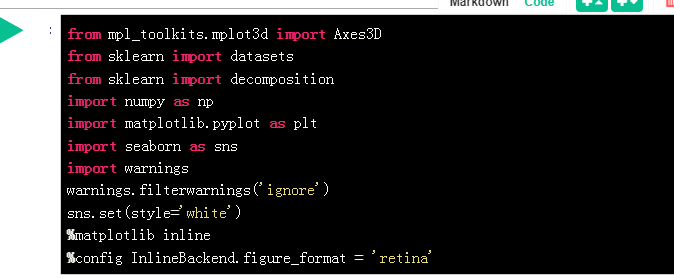
import decomposition

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import warnings warnings.filterwarnings('ignore') sns.set(style='white') %matplotlib inline %config InlineBackend.figure\_format = 'retina'



1. 通过 scikit-learn 提供的数据集接口导入莺尾花数据集。

iris = datasets.load\_iris()

X = iris.data

y = iris.target



（3）为了直观地查看数据的分布，使用三维图画出莺尾花的数据分布图。

fig = plt.figure(1, figsize=(6, 5))

plt.clf()

ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134) plt.cla()

for name, label in [('Setosa', 0), ('Versicolour', 1), ('Virginica', 2)]: ax.text3D(X[y == label, 0].mean(), X[y == label, 1].mean() + 1.5, X[y == label, 2].mean(), name, horizontalalignment='center', bbox=dict(alpha=.5, edgecolor='w', facecolor='w'))

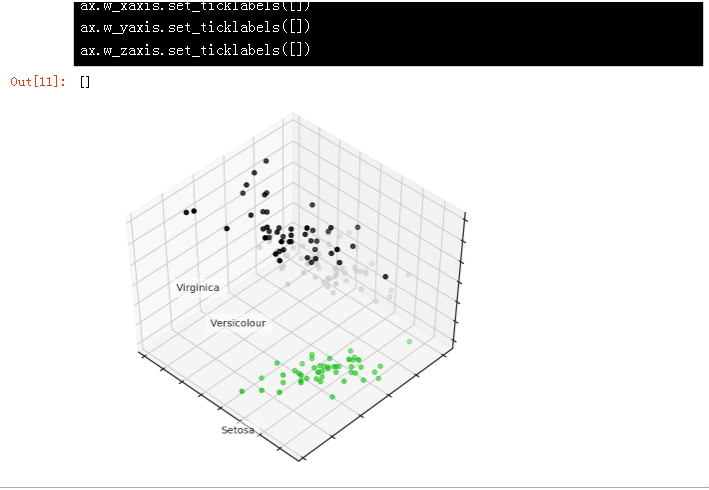
# 改变标签的顺序，让其与数据匹配

y\_clr = np.choose(y, [1, 2, 0]).astype(np.float)

ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y\_clr, cmap=plt.cm.nipy\_spectral) ax.w\_xaxis.set\_ticklabels([])

ax.w\_yaxis.set\_ticklabels([])

ax.w\_zaxis.set\_ticklabels([])



（4）现在让我们看看使用 PCA 是怎么样提高模型的识别性能的。同样先导入实验所用到的模块。

from sklearn.tree

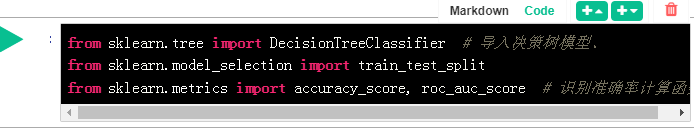
import DecisionTreeClassifier # 导入决策树模型、

from sklearn.model\_selection

import train\_test\_split

from sklearn.metrics

import accuracy\_score, roc\_auc\_score # 识别准确率计算函数



1. 莺尾花数据是一个相对容易区分的数据。因此，为了使实验结果对比明显。选用简单的决策树模型来对莺尾花数据进行分类。

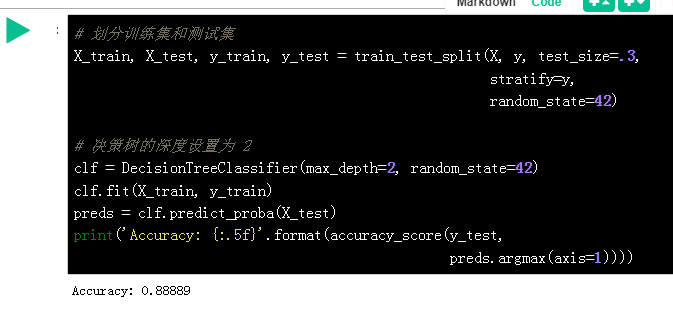
# 划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=.3, stratify=y, random\_state=42) # 决策树的深度设置为 2

clf = DecisionTreeClassifier(max\_depth=2, random\_state=42) clf.fit(X\_train, y\_train)

preds=clf.predict\_proba(X\_test)

print('Accuracy:{:.5f}'.format(accuracy\_score(y\_test, preds.argmax(axis=1))))

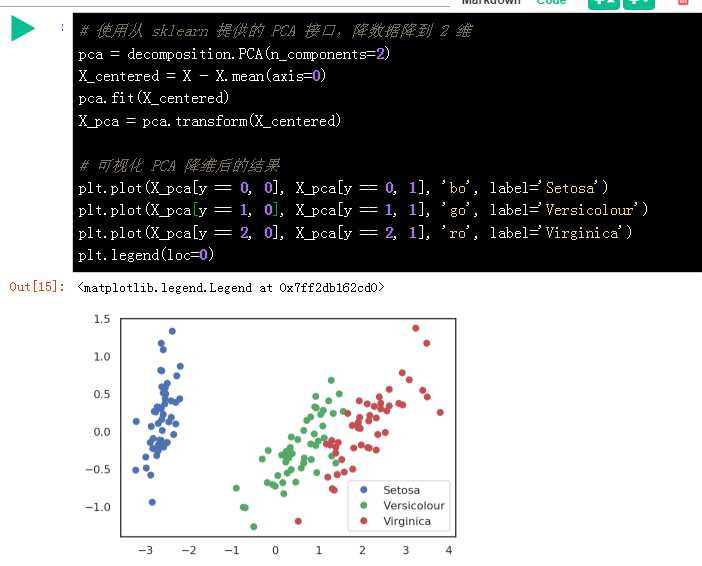


从上面的结果可知。在不对数据进行处理的情况下，使用决策树模型对莺尾花数据进行分类的准确率为 0.88889。

1. 现在使用 PCA 将莺尾花数据的维度降低到 2 维，然后画出降维后的数据分布图。

pca = decomposition.PCA(n\_components=2) X\_centered = X - X.mean(axis=0) pca.fit(X\_centered) X\_pca = pca.transform(X\_centered) # 可视化 PCA 降维后的结果

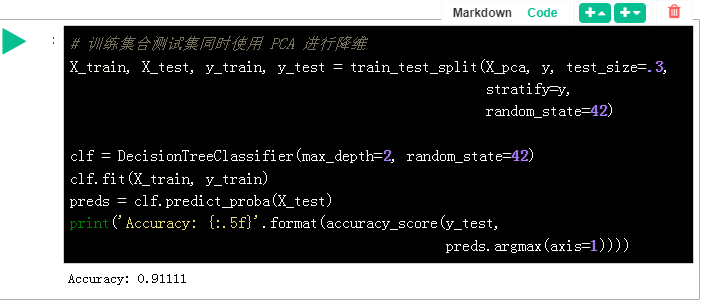
plt.plot(X\_pca[y == 0, 0], X\_pca[y == 0, 1], 'bo', label='Setosa') plt.plot(X\_pca[y == 1, 0], X\_pca[y == 1, 1], 'go', label='Versicolour') plt.plot(X\_pca[y == 2, 0], X\_pca[y == 2, 1], 'ro', label='Virginica') plt.legend(loc=0)



1. 同样的方法，将降维后的莺尾花数据输入到决策树模型中

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_pca, y, test\_size=.3, stratify=y, random\_state=42) clf = DecisionTreeClassifier(max\_depth=2, random\_state=42)

clf.fit(X\_train,y\_train) preds = clf.predict\_proba(X\_test) print('Accuracy: {:.5f}'.format(accuracy\_score(y\_test, preds.argmax(axis=1))))



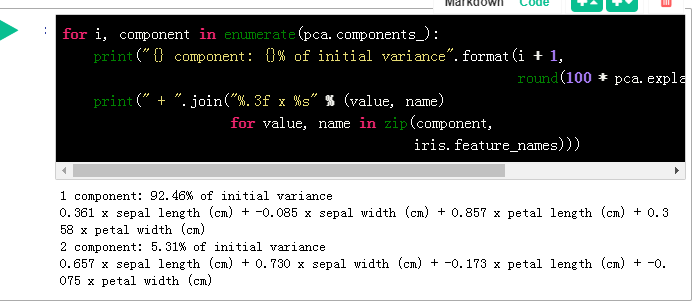
从上面的结果可知，对数据进行 PCA 降维之后，决策树模型的识别准确率提升到了 0.91111。这说明了 PCA 确实可以有效改善大部分机器学习算法的准确性和计算效率。

1. 那么降维后的每个主成分都来自于原始数据的哪些维度呢？让我们通过查看每个维度的方差百分比来解释这个问题。

for i, component in enumerate(pca.components\_):

print("{} component: {}% of initial variance".format(i + 1, round(100 \* pca.explained\_variance\_ratio\_[i], 2)))

print(" + ".join("%.3f x %s" % (value, name) for value, name in zip(component, iris.feature\_names)))



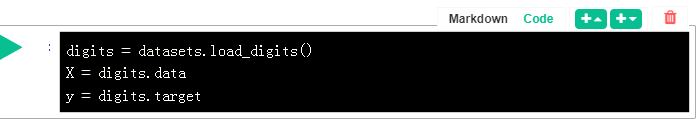
* 手写数字数据集

1. 在上面的例子中，莺尾花的原始数据只有 4 个维度。为了验证 PCA 在其他高维数据同样可行。接下来，使用之前实验所接触到的手写数字体数据集再完成一个示例练习。先导入手写数字数据集。

digits = datasets.load\_digits()

X = digits.data

y = digits.target

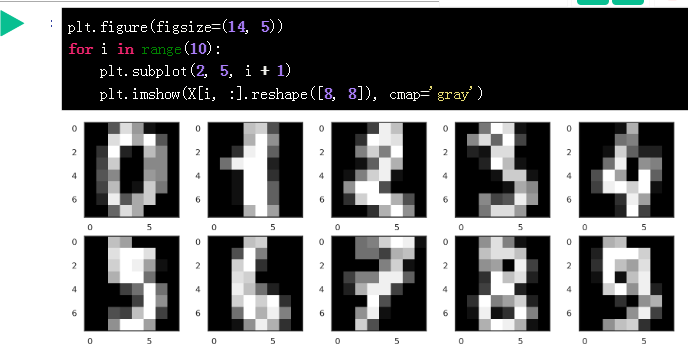


1. 数据集中的每个手写数字都是由 8×8 矩阵表示，每个像素值的表示颜色强度。获取数据集的前 10 个数字。并对这 10 个手写数字体数据进行可视化。

plt.figure(figsize=(14, 5)) for i in range(10):

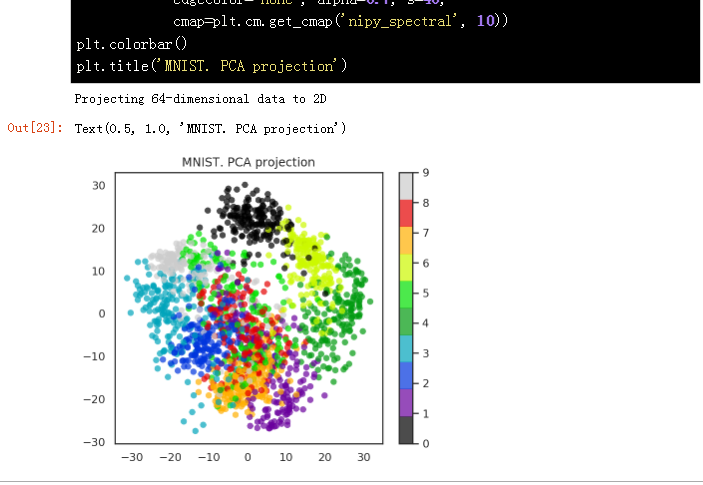
plt.subplot(2, 5, i + 1)

plt.imshow(X[i, :].reshape([8, 8]), cmap='gray')



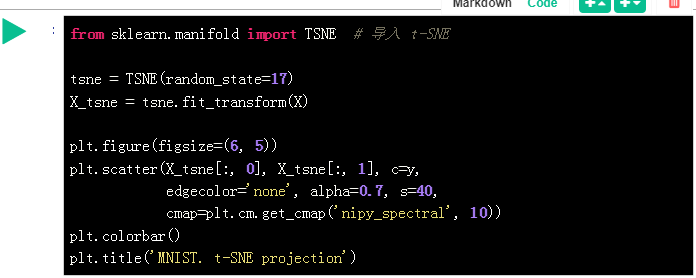
1. 手写数字体数据集的维度是 8×8 维的，即 64 维。只有将其降维减少到 2 维，才能对其进行可视化，以便查看其分布状态。

pca = decomposition.PCA(n\_components=2) X\_reduced = pca.fit\_transform(X) print('Projecting %d-dimensional data to 2D' % X.shape[1]) plt.figure(figsize=(6, 5)) plt.scatter(X\_reduced[:, 0], X\_reduced[:, 1], c=y, edgecolor='none', alpha=0.7, s=40, cmap=plt.cm.get\_cmap('nipy\_spectral', 10)) plt.colorbar() plt.title('MNIST. PCA projection')



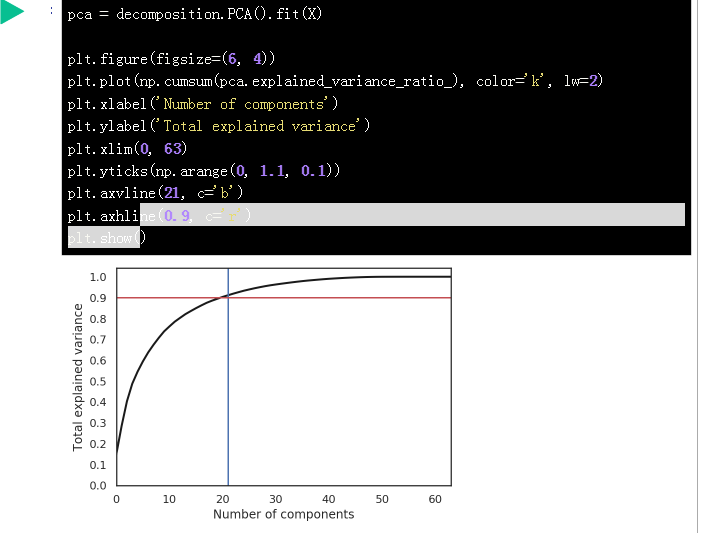
1. 除了 PCA 之外，t-SNE 也是一种常用的降维算法。相比于 PCA， t-SNE 不具有线性约束。下面使用 t-SNE 来对手写数字体数据进行降维，并对降维后的数据进行可视化。

from sklearn.manifold import TSNE # 导入 t-SNE tsne = TSNE(random\_state=17) X\_tsne = tsne.fit\_transform(X) plt.figure(figsize=(6, 5)) plt.scatter(X\_tsne[:, 0], X\_tsne[:, 1], c=y, edgecolor='none', alpha=0.7, s=40, cmap=plt.cm.get\_cmap('nipy\_spectral', 10)) plt.colorbar() plt.title('MNIST. t-SNE projection')



1. 在对手写数据集进行降维时，如果要保留原始数据的 90％ 散度，应该将数据降到多少维呢？先来画出主成分与其所保留的原始数据散度的关系。

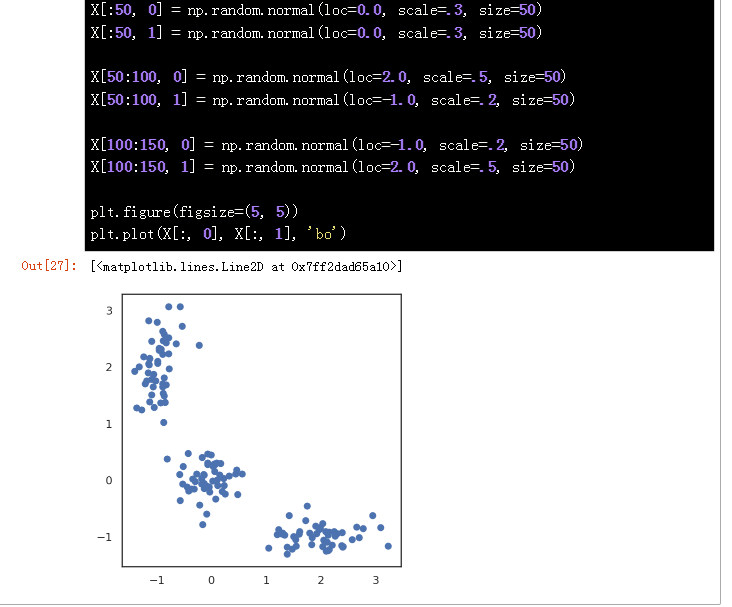
pca = decomposition.PCA().fit(X) plt.figure(figsize=(6, 4)) plt.plot(np.cumsum(pca.explained\_variance\_ratio\_), color='k', lw=2) plt.xlabel('Number of components') plt.ylabel('Total explained variance') plt.xlim(0, 63) plt.yticks(np.arange(0, 1.1, 0.1)) plt.axvline(21, c='b') plt.axhline(0.9, c='r') plt.show()



* K-Means 聚类

1. 为了更好的理解 K-Means 算法的原理，这里通过一个例子来进行说明。先构建出一个数据集并画图它的分布图，该数据集含有三个簇。

X = np.zeros((150, 2)) np.random.seed(seed=42) X[:50, 0] = np.random.normal(loc=0.0, scale=.3, size=50) X[:50, 1] = np.random.normal(loc=0.0, scale=.3, size=50) X[50:100, 0] = np.random.normal(loc=2.0, scale=.5, size=50) X[50:100, 1] = np.random.normal(loc=-1.0, scale=.2, size=50) X[100:150, 0] = np.random.normal(loc=-1.0, scale=.2, size=50) X[100:150, 1] = np.random.normal(loc=2.0, scale=.5, size=50) plt.figure(figsize=(5, 5)) plt.plot(X[:, 0], X[:, 1], 'bo')



1. 开始动手实现 K-Means 算法， K-Means 算法的实现非常简单。按照上述的算法步骤逐步实现即可。在这里，我们使用欧几里德距离来衡量两个数据点之间的距离。当然，你也可以使用其他距离度量方式。

from scipy.spatial.distance import cdist # 随机初始化三个中心点 np.random.seed(seed=42) centroids = np.random.normal(loc=0.0, scale=1., size=6) centroids = centroids.reshape((3, 2)) cent\_history = [] cent\_history.append(centroids) for i in range(3): # 计算每个点到中心的距离 distances = cdist(X, centroids) # 获取数据别分到哪个簇 labels = distances.argmin(axis=1)

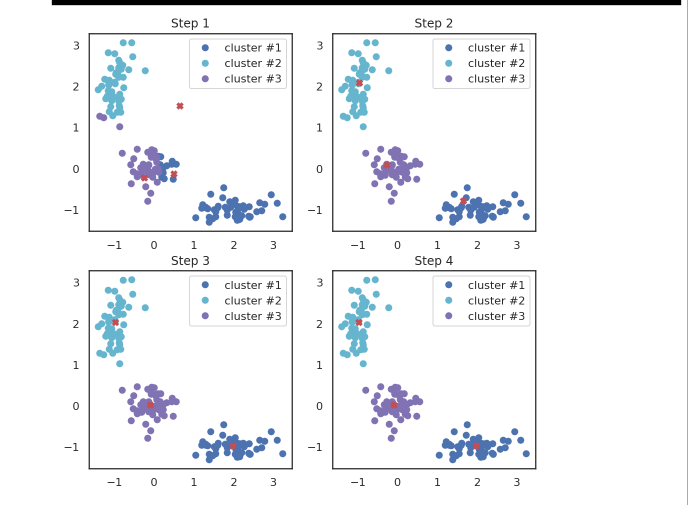
centroids = centroids.copy() centroids[0, :] = np.mean(X[labels == 0, :], axis=0) centroids[1, :] = np.mean(X[labels == 1, :], axis=0) centroids[2, :] = np.mean(X[labels == 2, :], axis=0) cent\_history.append(centroids)

1. 可视化出 K-Means 算法的运行过程，以便更好地理解。

plt.figure(figsize=(8, 8)) for i in range(4): distances = cdist(X, cent\_history[i]) labels = distances.argmin(axis=1)

plt.subplot(2, 2, i + 1)

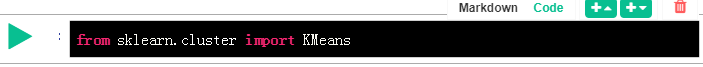
plt.plot(X[labels == 0, 0], X[labels == 0, 1], 'bo', label='cluster #1') plt.plot(X[labels == 1, 0], X[labels == 1, 1], 'co', label='cluster #2') plt.plot(X[labels == 2, 0], X[labels == 2, 1], 'mo', label='cluster #3') plt.plot(cent\_history[i][:, 0], cent\_history[i][:, 1], 'rX') plt.legend(loc=0) plt.title('Step {:}'.format(i + 1))



* K-均值算法中 K 值的选择

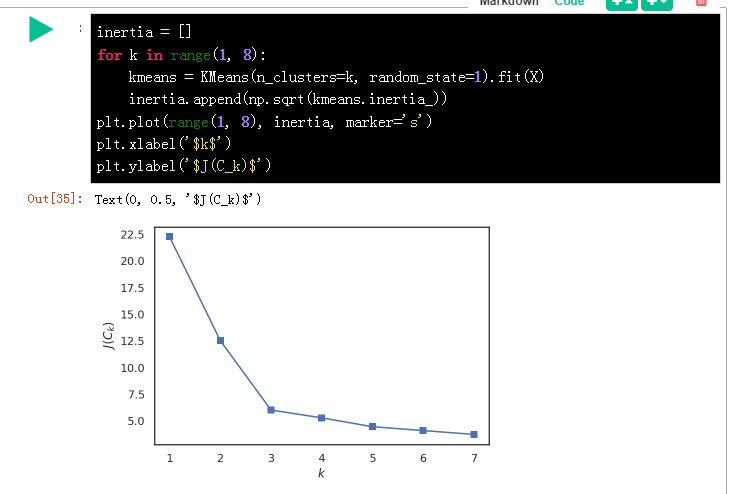
1. 因为 scikit-learn 也提供了各种聚类算法接口，而且使用这些接口有许多优点。例如：这些算法可以并行完成，有效减少了计算时间。所以在这里为了方便，直接使用 scikit-learn 提供的 K-Means 接口进行实验。

from sklearn.cluster import KMeans # 导入 K-均值聚类模型



1. 求出 K 值得选择与 J(C\_k)J(Ck​) 的关系，并画出它们的关系图。

inertia = [] for k in range(1, 8): kmeans = KMeans(n\_clusters=k, random\_state=1).fit(X) inertia.append(np.sqrt(kmeans.inertia\_)) plt.plot(range(1, 8), inertia, marker='s') plt.xlabel('$k$') plt.ylabel('$J(C\_k)$')



从上图中，可以看到当 kk 小于 3 时， J(C\_k)J(Ck​) 下降得非常快。之后相对平稳。这意味着选择 kk 等于 3 最为合适。

1. ****实验分析****

K-Means 算法是一个 \text{NPhard}NPhard 问题。对于 nn 个 dd 维数据，当我们想要将其聚为 kk 个簇时，K-Means 算法的复杂度为 O(n^{d k+1})O(ndk+1) ，这意味着模型的训练需要大量的时间。不过，这里有一些启发式方法可以缓解这个问题，例如像 MiniBatch K-Means 算法。 它每次仅采用部分数据而不是一次使用整个数据集，然后通过前面所述的方法，通过一个簇中所有观测点的平均值来移动质心。

1. ****心得体会****

本次实验主要围绕无监督学习算法进行，主要涉及到 PCA 降维方法、各种聚类方法以及各种对聚类模型的评价指标。至此，对无监督学习有了一个清晰的了解，并会使用 PCA 来对数据进行降维。