实验三：面向图像压缩和人脸识别的自动聚类和降维分析

介绍：

在本练习中，你将实现 K-means 聚类算法，并将其应用于压缩图像。在第二部分中，你将使用主成分分析来找到人脸图像的低维表示。

实验文件说明：

ex7.py–K-means 聚类主文件

ex7_pca.py-PCA 主文件

ex7data1.mat–PCA 示例数据集

ex7data2.mat–K-means 的示例数据集

ex7faces.mat-人脸数据集

bird_small.mat-示例图片

displayData.py -显示存储在矩阵中的二维数据

drawLine.py-在现有图形上画一条线

plotDataPoints.py -初始化 k-means 的中心

plotProgresskMeans.py -绘制 k-means 迭代的每一步图像

runkMeans.py -运行 K-means 算法

[*]pca.py -进行主成分分析

[*]projectData.py -将数据集投射到低维空间中

[*]recoverData.py -从投影恢复原始数据

[*]findClosestCentroids.py - 找到最近的中心(K-means)

[*] computeCentroids.py - 计算中心平均值(K-means)

[*] kMeansInitCentroids.py - k 均值中心的初始化(K-means)

带*的函数有部分代码缺少需要你按照提示将代码补充完整。

1．k - means 聚类

在这个实验中，你将实现 K-means 算法，并将其用于图像压缩。你将首先从一个示例 2D 数据集开始，它将帮助你了解 K-means 算法如何工作。在此之后，

你将使用 K-means 算法进行图像压缩，减少图像中出现的颜色的数量-通过只使用该图像中最常见的颜色。

1.1 实现 k–means

K-means 算法是一种自动聚类相似数据实例的方法。具体来说，给定一个训练集$\{x^{(1)}, \ldots, x^{(m)}\}$ $\left(x^{(i)} \in R^n\right)$,将数据分组成几个“集群”。K-means 实质是一个迭代过程，它首先猜测初始的质心，然后通过反复将数据分配给最近的质心，然后根据分配重新计算质心来改进上一步的分配结果。

1.1.1 寻找最近的质心

在 K-means 算法的“聚类分配”阶段，在给定当前质心位置的情况下，算法将每个训练实例$x^{(i)}$分配到其最近的质心。特别地，对于我们设置的每个例子 i:

$$c^{(i)} := j \quad \text{that minimizes} \quad ||x^{(i)} - \mu_j||^2,$$

其中$c^{(i)}$为最接近的中心的指标，u_j为第 j 个质心的位置。注意，$c^{(i)}$对应于起始代码中的 idx(i)。

你的任务是完成 findClosestCentroids.py 中的代码。这个函数获取数据矩阵 X 和所有中心点的位置，应该输出一个一维数组 idx，该数组包含索引($K \in \{1, \ldots, K\}$)，其中 K 为质心总数)。你可以在每个训练例子和每个中心上使用一个循环来实现这一点。

在 findClosestCentroids.py 中完成代码之后，脚本 ex7.py 将运行你的代码，你应该看到输出[0 2 1]对应于前 3 个示例的质心分配。

1.1.2 质心计算方法

给每个点赋值一个质心后，算法的第二阶段会对每个质心重新计算赋值给它的点的均值。特别地，对于我们设置的每个质心 k。

$$\mu_k := \frac{1}{|C_k|} \sum_{i \in C_k} x^{(i)}$$

其中，C_k是质心为 k 的一组样本。具体地说，如果两个例子$x^{(3)}$和$x^{(5)}$被分配到质心为 k = 2，那么你应该更新$u_2 = \frac{1}{2}(x^{(3)} + x^{(5)})$。

你应该在 computeCentroids.py 中完成代码。你可以在质心上使用一个循环来实现这个函数。

1.2 在示例数据集上使用 K-means

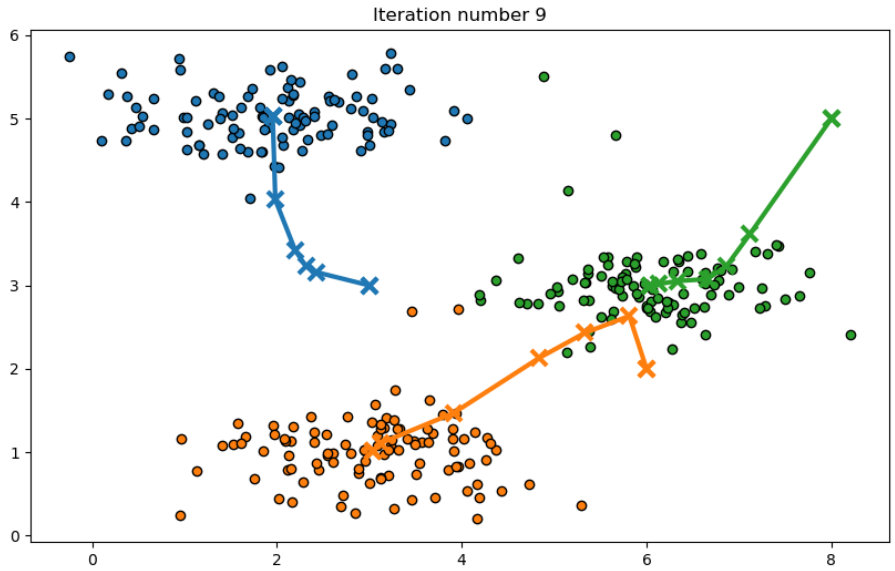

图 1:预期的输出。

在完成两个函数(findClosestCentroids.py 和 computeCentroids.py)之后，在 ex7.py 中的下一步。将在一个 2-D 数据集上运行 K-means 算法，以帮助你理解 K-means 是如何工作的。函数 runKmeans.py 实现数据的可视化。注意，代码调用了你在循环中实现的两个函数。当你运行下一步时，K-means 代码将生成一个可视化视图，在每次迭代中引导你了解算法的进展。多次按 enter 键可以看到 K-means 算法的每一步是如何改变重心和集群分配的。最后，你的图应该如图 1 所示。

1.3 随机初始化

ex7.py 中示例数据集的初始中心点分配的设计使你可以看到与图 1 相同的图。在实践中，初始化质心的一个好策略是从训练集中选择随机的例子。 在本部分的练习中，你应该完成函数 kMeansInitCentroids.py 与以下代码:

```
idx = np.random.randint(0, X.shape[0], size=K)
centroids = X[idx, :]
```

上面的代码首先随机排列示例的索引(使用 np.random.randint)。然后，根据指标的随机排列选择前 K 个样本。这使得示例可以随机选择，以免两次选择同一示例。

1.4 K-means 图像压缩

在这个实验中，你将应用 K-means 压缩图像。在图像的 24 位色图中，用 3 个 8 位无符号整数(从 0 到 255) 表示一个像素，它们指定红色、绿色和蓝色的强度值。这种编码通常被称为 RGB 编码。图像包含数千种颜色，在练习的这一部分，你将把颜色的数量减少到 16 种。通过进行这种还原，可以以一种有效的方式表示(压缩)照片。你只需要存储 16 种选中颜色的 RGB 值，对于图像中的每个像素，你现在只需要存储该位置的颜色索引(这里只需要 4 位来表示 16 种可能性)。

在本实验中，你将使用 K-means 算法来选择 16 种将用于表示压缩图像的颜色。具体来说，你将把原始图像中的每个像素作为一个数据示例，并使用 K-means 算法找到 16 种颜色，在三维 RGB 空间中最好地对像素进行分组(聚类)。一旦计算出图像上的集群重心，就可以使用 16 种颜色替换原始图像中的像素。

1.4.1 像素上的 K-means

在 Python 中，可以读取如下图像:

```
data = loadmat('./data/bird_small.mat')
A = data['A']
A = A / 255    # Divide by 255 so that all values are in the range 0 - 1
print(A[50, 33, 2])
```

这将创建一个三维矩阵 pic，其前两个索引标识像素位置，最后一个索引表示红色、绿色或蓝色。例如，A[50, 33, 2]给出了第 51 行和第 34 列像素的蓝色强度。ex7.py 中的代码首先加载图像，然后对其进行重塑以创建像素颜色的 m × 3 矩阵(其中 m = 16384 = 128 × 128)，并在其上调用 K-means 函数。找到最上面的 K = 16 种颜色来表示图像后，现在就可以使用 findClosestCentroids.py 函数将每

个像素位置分配到最近的质心。这允许你使用每个像素的质心赋值来表示原始图像。注意，你已经大大减少了描述图像所需的位数。原始图像每个 128×128 像素位置需要 24 位，因此总大小为 128×128 × 24 = 393 216 位。新的表示需要一些开销存储，以 16 种颜色的字典的形式，每一种颜色需要 24 位，但图像本身每个像素位置只需要 4 位。因此，最终使用的比特数是 16 × 24 + 128 × 128 × 4 = 65,920 比特，这相当于将原始图像压缩大约 6 倍。

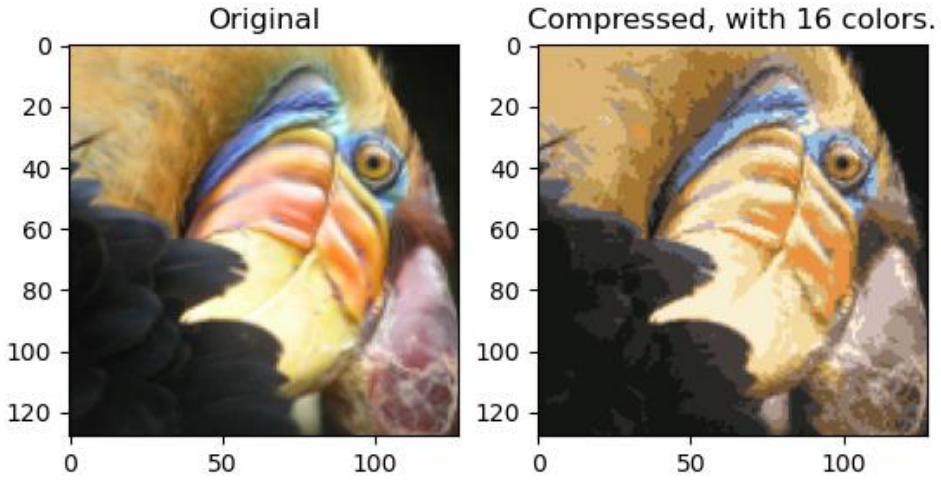

图 2:原始图像和重建图像(K-means)。

最后，您可以通过仅根据质心分配重建图像来查看压缩效果。具体来说，您可以用分配给它的质心的平均值替换每个像素位置。图 3 显示了我们获得的重构。尽管生成的图像保留了原始图像的大部分特征，但我们也看到了一些压缩伪影。

1.5 可选实验:

在实验中使用你自己的图像，修改提供的代码，以便在你自己的图像上运行。请注意，如果你的图像非常大，那么 K-means 可能需要很长时间才能运行。你还可以尝试改变 K，以查看 K 对压缩的影响。

2 主成分分析

在这个练习中，你将使用主成分分析(PCA)来进行降维。你将首先使用一个示例 2-D 数据集进行实验，以直观地了解 PCA 是如何工的，然后在 5000 个人脸

图像数据集的更大数据集上使用它。提供的脚本 ex7_pca.py，会帮助你完成练习的前半部分。

2.1 示例数据集

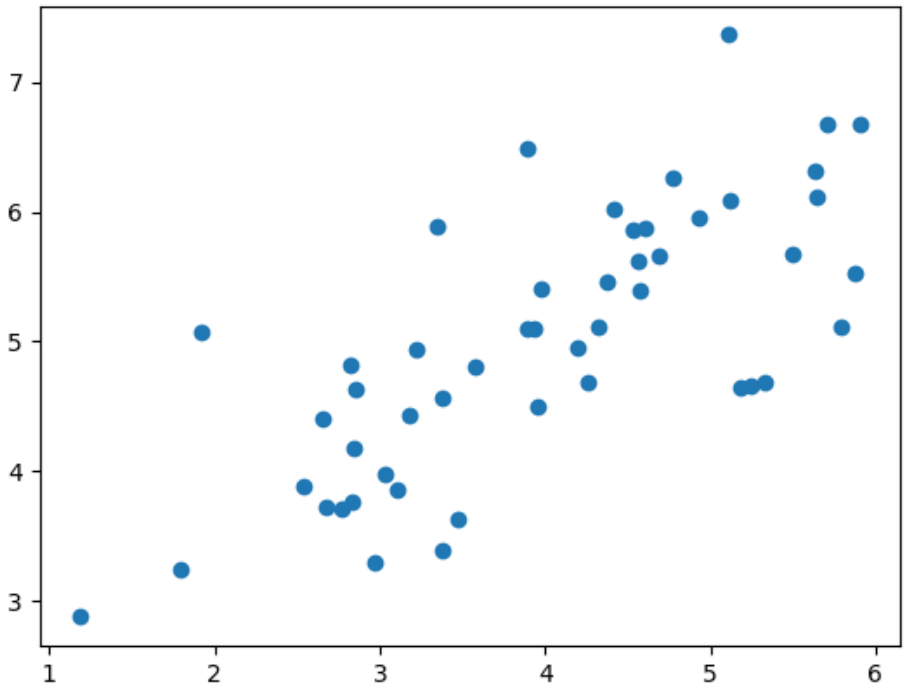

图 3:示例数据集 1

为了帮助你理解 PCA 是如何工作的，你将首先从一个 2-D 数据集开始，它有一个大的变化方向和一个小的变化方向。脚本 ex7_pca.py 将绘制训练数据(图 4)。在这一部分的练习中，你将看到当您使用 PCA 将数据从 2D 减少到 1D 时会发生什么。在实践中，你可能希望将数据从 256 维减少到 50 维;但是在这个例子中使用低维数据可以让我们更好地将算法可视化。

2.2 实现 PCA

在本部分的实验中，你将实现 PCA。主成分分析由两个计算步骤组成:首先，首先，计算数据的协方差矩阵。然后，你使用 Python 中的 np.linalg.svd 函数来计算特征向量$\{U_1, U_2, \ldots., U_n\}$。这些将对应于数据中变化的主要成分。在使用 PCA 之前，重要的是首先通过从数据集中减去每个特征的平均值来归一化数据，并缩放每个维度，使它们在相同的范围内。在提供的脚本 ex7_pca.py，这个归一化已经为你使用 featureNormalize.py 函数执行。归一化数据后，可以运行 PCA 来计

算主成分。你的任务是在 pca.py 中完成代码来计算数据集的主成分。首先，你需要计算数据的协方差矩阵，它由:

$$\Sigma = \frac{1}{m} X^T X$$

其中 X 是包含样本的数据矩阵，m 是样本个数。请注意，Σ 是一个 n × n 的矩阵，而不是求和算子。

在计算完协方差矩阵后，可以对其运行奇异值分解(SVD)来计算主成分。在 Python 中，你可以使用以下命令运行 SVD:U, S, V= np.linalg.svd(Sigma)，其中 U 将包含主成分，S 将包含一个对角矩阵。其中 U 包含主成分，S 包含一个对角矩阵。

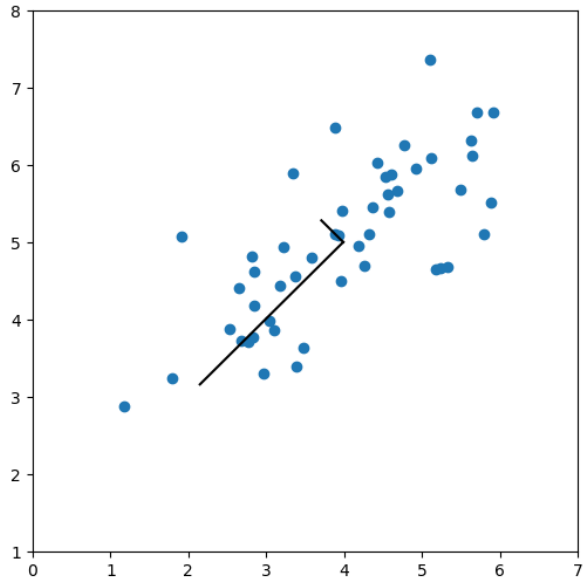

图 5:数据集的特征向量

2.3 主成分分析降维

在计算主成分之后，你可以使用它们来降低你的数据集的特征维数，方法是将每个例子投影到一个低维空间，$x^{(i)} \rightarrow z^{(i)}$ (例如，将数据从 2D 投影到 1D)。在实验的这一部分中，你将使用由 PCA 返回的特征向量，并将示例数据集投影到一维空间中。在实践中，如果你正在使用线性回归或神经网络等学习算法，那么你现在可以使用投影数据而不是原始数据。通过使用投影数据，你可以更快地训练你的模型，因为输入中的维度更少。

2.3.1 将数据投影到主成分上

你现在应该在 projectdata.py.中完成代码。具体地说,给定一个数据集 X,主成分 U,和所需维度降低的数量 K。你应该把 X 中的每个示例投影到 U 中的前 K 个分量上。注意，U 中的前 K 个分量由 U 的前 K 列给出，即 U_reduce＝U(:，1:K)。

2.3.2 重建数据近似值

将数据投影到低维空间后，可以通过将数据投影到原始的高维空间来近似地恢复数据。你的任务是完成 recoverData.py,将 Z 中的每个例子投射回原始空间，并在 X_rec 中返回恢复的近似值。

2.3.3 投影可视化

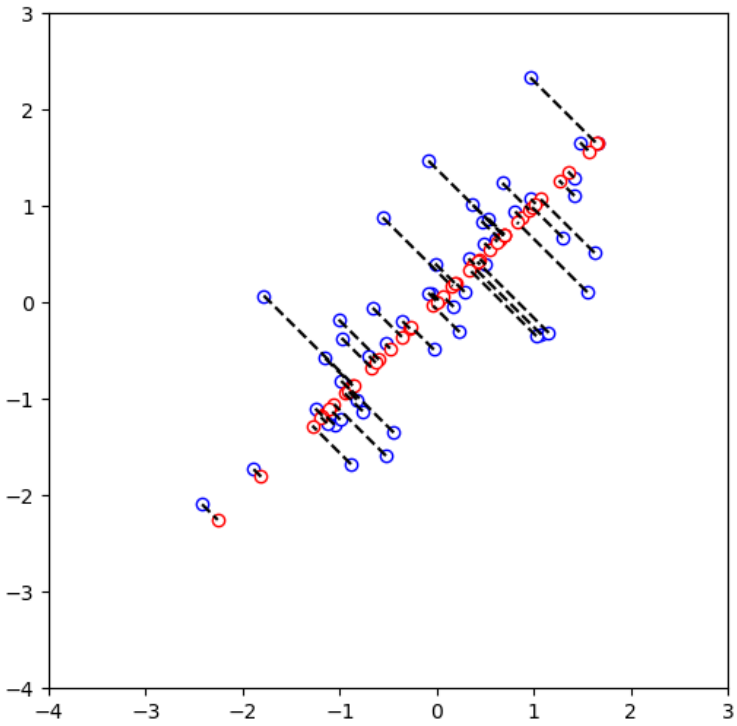

图 5:PCA 后的归一化和投影数据

在完成 projectData.py 和 recoverData.py 后。ex7_pca.py 将执行投影和近似重建，以显示投影如何影响数据。在图 5 中，原始数据点用蓝色圆圈表示，而投影数据点用红色圆圈表示。投影有效地只保留了 U1 给出的方向上的信息。

2.4 人脸图像数据集

在这个部分的实验中，你将在人脸图像上运行 PCA，看看它如何在实践中用于降维。数据集 ex7faces.mat 包含一个数据 X 的人脸图像，每个人脸的大小为 32×32 灰度。每一行 X 对应一个人脸图像(长度为 1024 的行向量)。ex7_pca.py 中将加载并可视化前 100 张人脸图像。

图 6:人脸数据集

2.4.1 人脸数据集上使用 PCA

为了在人脸数据集上运行 PCA，我们首先通过从数据矩阵 x 中减去每个特征的平均值来归一化数据集。脚本 ex7_PCA.py 将为你完成，然后运行你的 PCA 代码。运行 PCA 之后，你将获得数据集的主成分。注意，U 中的每个主成分(每一行)都是一个长度为 n 的向量(其中对于人脸数据集，n = 1024)。我们可以将这些主要成分可视化，方法是将它们重塑成一个 32 × 32 的矩阵，这个矩阵对应于原始数据集中的像素。脚本 ex7_pca.py 显示描述最大变化的前 36 个主成分 (图 7)。

2.4.2 降维

现在已经计算了人脸数据集的主成分，可以使用它来降低人脸数据集的维数。这允许你使用较小的输入尺寸(例如，100 维)而不是原始的 1024 维来使用你的学习算法。这可以帮助加快你的学习算法。

图 7:人脸数据集上的主成分

图 8:原始人脸图像和仅从前 100 个主成分重建的人脸图像。

ex7_pca.py 的下一部分,将把人脸数据集投射到前 100 个主成分上。具体地说，每个人脸图像现在都由向量$z^{(i)} \in R^{100}$来描述。要了解数据集在 PCA 中丢失了什么，可以从投影数据集中恢复数据。在 ex7_pca.py 中对数据进行近似恢复，将原始和投影的人脸图像并排显示(图 8)。从重建结果中可以看出，保留了人脸的总体结构和外观，而丢失了细节。这是对一个数据集的大小来说是一个显著的减少，(超过 10 倍)可以帮助加快你的学习算法训练。例如，如果你正在训练一个神经网络来执行人识别(给定一个人脸图像，预测这个人的身份)你可以使用降维输入，只使用 100 维而不是原始像素。

2.5 用 PCA 进行可视化

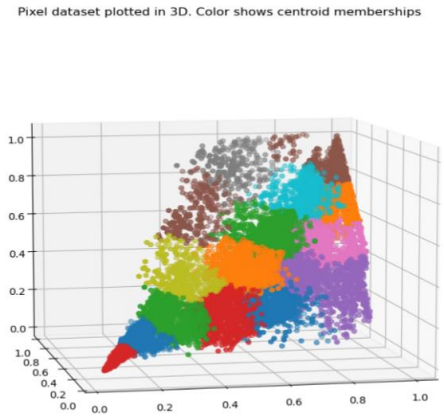

图 9:3D 原始数据

在前面的 K-means 图像压缩实验中，在 3 维 RGB 空间中使用了 K-means 算法。在 ex7.py 主元分析的最后一部分脚本中，已经提供了使用 scatter 函数在这个 3D 空间中可视化最终像素赋值的代码。每个数据点根据它被分配到的集群来着色。你可以在图形上拖动鼠标以旋转和查看三维数据。

事实证明，在三维或更大的空间中可视化数据集是很麻烦的。因此，即使以丢失一些信息为代价，也通常希望只显示 2D 数据。在实践中，主成分分析通常用于降低数据的维数以达到可视化的目的。在 ex7_pca.py 的下一部分时，脚本将应用你的 PCA 实现将三维数据减少到二维，并在二维散点图中显示结果。PCA 投影可以被认为是一个旋转，它选择的视图最大限度地扩大了数据的分布，这通常对应于“最佳”视图。

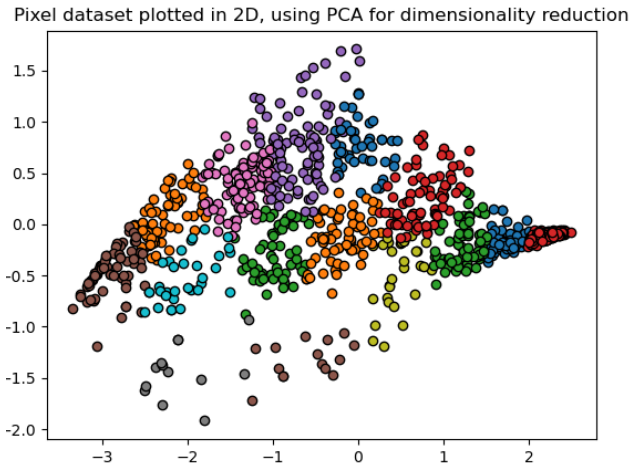

图 10:使用 PCA 生成的 2-D 可视化数据