22336216_陶宇卓_模式识别_homework2

实验目的

1. 掌握 PCA 和 LDA 的原理和实现。
2. 掌握 KNN 分类器的原理和实现。

实验内容

1. **实现PCA算法**：计算数据的协方差矩阵。对协方差矩阵进行奇异值分解（SVD），得到特征值和特征向量。利用特征向量构建投影矩阵，实现数据降维。
2. **实现LDA算法**：将数据按类别分组，计算每个类别的均值向量。计算类内散度矩阵（Sw）和类间散度矩阵（Sb）。求解广义特征值问题，得到投影矩阵，实现数据降维。
3. **数据降维与可视化**：分别用PCA和LDA对训练集学习投影矩阵，并将降维后的数据可视化。
4. **KNN分类与评估**：使用降维后的训练数据训练KNN分类器。

实验步骤与原理

1. PCA算法实现

原理：
PCA通过线性变换将数据投影到方差最大的方向上，保留数据的主要特征，实现降维。具体步骤：

1. **数据标准化**：对训练数据进行中心化处理（减去均值）。
2. **计算协方差矩阵**：协方差矩阵反映数据各维度间的相关性。
3. **特征分解**：对协方差矩阵进行SVD分解，得到特征值和特征向量。
4. **选择主成分**：按特征值从大到小排序，选择前k个特征向量构成投影矩阵。
5. **数据降维**：将原始数据投影到选定的特征向量上，得到降维后的数据。

```python
# ======================== PCA 类 ========================
class PCA:
    def fit(self, X):
        self.mean = np.mean(X, axis=0)
        X_centered = X - self.mean
        cov = np.cov(X_centered, rowvar=False)
        U, S, Vt = np.linalg.svd(cov)
        self.components = Vt

    def transform(self, X, dim):
        X_centered = X - self.mean
        return np.dot(X_centered, self.components[:dim].T)
```

---

2. LDA算法实现

原理:

LDA通过最大化类间距离和最小化类内距离，寻找最优投影方向，使得降维后的数据更有利于分类。具体步骤:

1. **计算类别均值**：对每个类别的数据计算均值向量。
2. **计算类内散度矩阵（Sw）**：衡量同一类别内数据的离散程度。
3. **计算类间散度矩阵（Sb）**：衡量不同类别间数据的分离程度。
4. **求解投影矩阵**：通过广义特征值问题求解最优投影方向，最大化类间散度与类内散度的比值。
5. **数据降维**：将数据投影到选定的特征向量上。

```python
# ======================== LDA 类 ========================
class LDA:
    def fit(self, X, y):
        self.classes = np.unique(y)
        n_features = X.shape[1]
        mean_overall = np.mean(X, axis=0)

        Sw = np.zeros((n_features, n_features))
        Sb = np.zeros((n_features, n_features))

        for c in self.classes:
            X_c = X[y == c]
            mean_c = np.mean(X_c, axis=0)
            Sw += np.dot((X_c - mean_c).T, (X_c - mean_c))
            n_c = X_c.shape[0]
            mean_diff = (mean_c - mean_overall).reshape(-1, 1)
            Sb += n_c * mean_diff @ mean_diff.T

        eigvals, eigvecs = np.linalg.eig(np.linalg.pinv(Sw) @ Sb)
        sorted_indices = np.argsort(-eigvals.real)
        self.components = eigvecs[:, sorted_indices].T.real

    def transform(self, X, dim):
        return np.dot(X, self.components[:dim].T)
```

---

3. 数据降维与可视化

1. **数据集划分**：将Yale人脸数据集按比例划分为训练集和测试集。
2. **学习投影矩阵**：分别用PCA和LDA从训练集中学习投影矩阵。
3. **可视化**：
    - 将数据降维到2维，绘制散点图，不同类别用不同颜色标记。
    - 显示前8个特征向量对应的图像，观察主要特征。

```python
# ======================== PCA 降维 ========================
pca = PCA()
pca.fit(train_data)
train_pca = pca.transform(train_data, dim=8)
test_pca = pca.transform(test_data, dim=8)

# 可视化前8个特征向量
fig, axes = plt.subplots(2, 4, figsize=(8, 4))
for i in range(8):
    axes[i//4, i%4].imshow(pca.components[i].reshape(64, 64), cmap='gray')
    axes[i//4, i%4].set_title(f'PCA {i+1}')
plt.tight_layout()
plt.savefig("result/pca_basis.png")
# PCA 降维到 2D 可视化
pca_2d = pca.transform(train_data, dim=2)

plt.figure(figsize=(10, 7))
colors = plt.colormaps.get_cmap('tab20')

for i in range(class_number):
    start = i * train_test_split
    end = (i + 1) * train_test_split
    plt.scatter(pca_2d[start:end, 0], pca_2d[start:end, 1],
                label=f'Class {i+1}', color=colors(i / class_number), s=40, alpha=0.7, edgecolors='k', linewidths=0.3)

plt.title("PCA 降维至二维空间 - 15类人脸分布", fontsize=14)
plt.xlabel("PCA 第一维", fontsize=12)
plt.ylabel("PCA 第二维", fontsize=12)
plt.grid(True, linestyle='--', alpha=0.5)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', fontsize=9, title="类别")
plt.tight_layout()
plt.savefig("result/pca_2d.png", dpi=300)
plt.show()

```

```python
# ======================== LDA 降维 ========================
lda = LDA()
lda.fit(train_data, train_label)
train_lda = lda.transform(train_data, dim=8)
test_lda = lda.transform(test_data, dim=8)

# 可视化前8个特征向量
fig, axes = plt.subplots(2, 4, figsize=(8, 4))
for i in range(8):
    axes[i//4, i%4].imshow(lda.components[i].reshape(64, 64), cmap='gray')
    axes[i//4, i%4].set_title(f'LDA {i+1}')
plt.tight_layout()
plt.savefig("result/lda_basis.png")
# LDA 降维到 2D 可视化
lda_2d = lda.transform(train_data, dim=2)

plt.figure(figsize=(10, 7))
colors = plt.colormaps.get_cmap('tab20')

for i in range(class_number):
    start = i * train_test_split
    end = (i + 1) * train_test_split
    plt.scatter(lda_2d[start:end, 0], lda_2d[start:end, 1],
                label=f'Class {i+1}', color=colors(i / class_number), s=40, alpha=0.7, edgecolors='k', linewidths=0.3)

plt.title("LDA 降维至二维空间 - 15类人脸分布", fontsize=14)
plt.xlabel("LDA 第一维", fontsize=12)
plt.ylabel("LDA 第二维", fontsize=12)
plt.grid(True, linestyle='--', alpha=0.5)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', fontsize=9, title="类别")
plt.tight_layout()
plt.savefig("result/lda_2d.png", dpi=300)
plt.show()
```

结果:

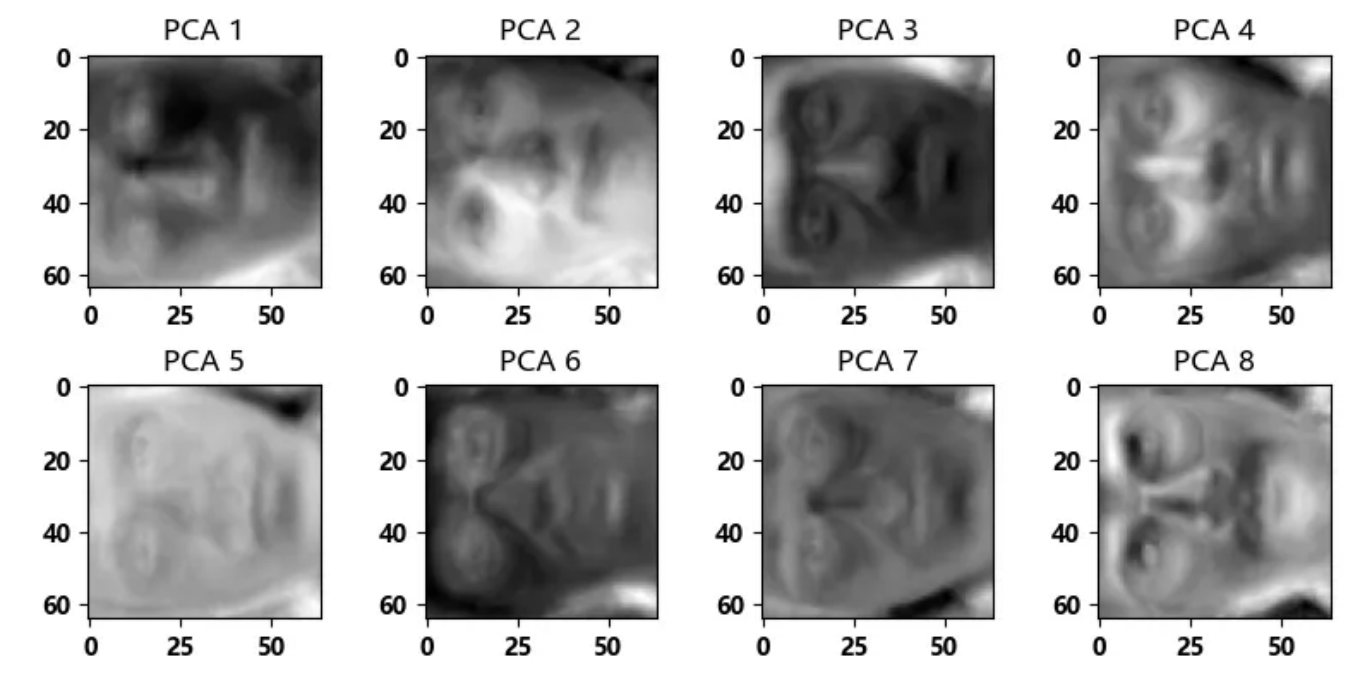

图1 PCA前八个特征向量

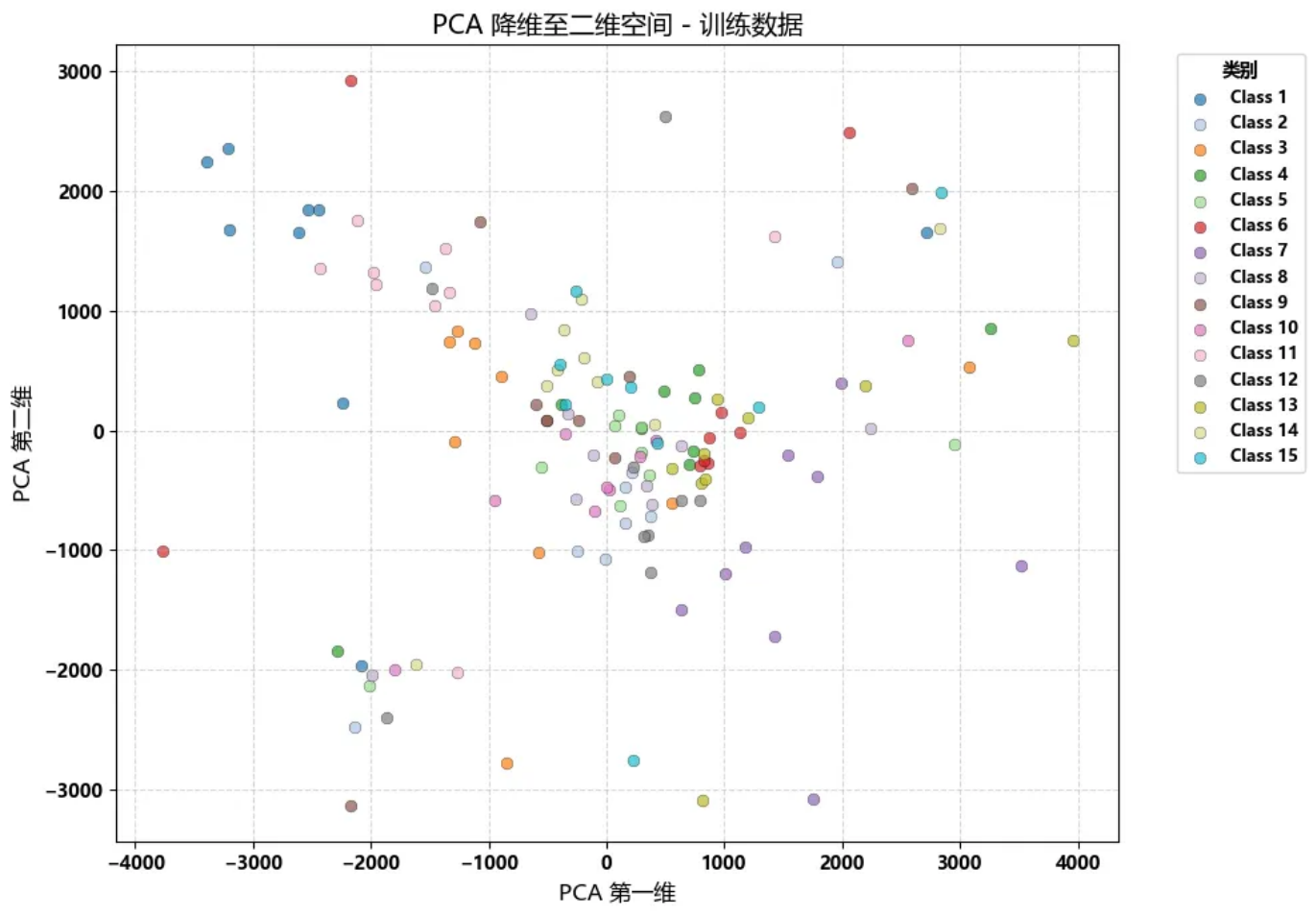

图2

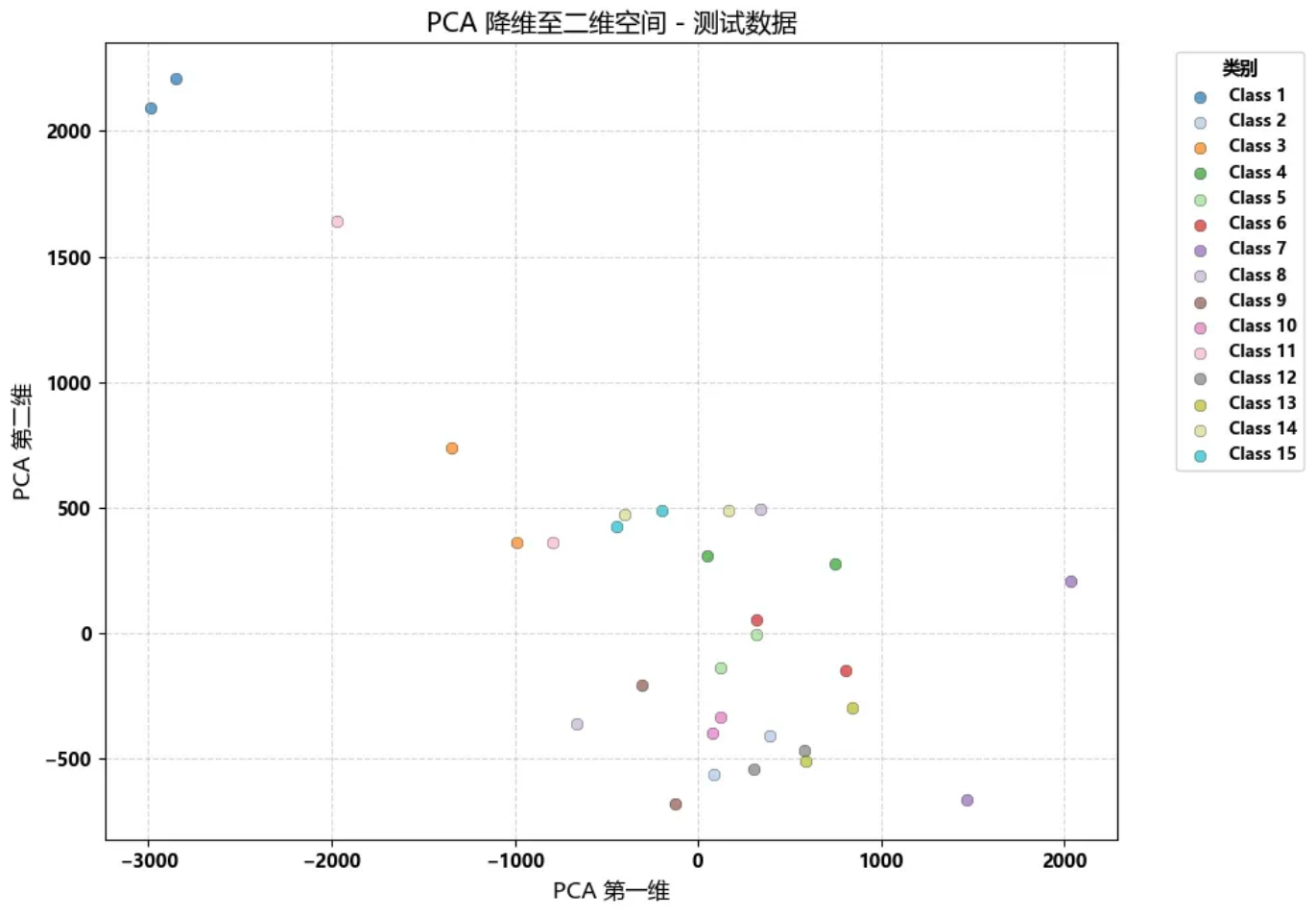

图3

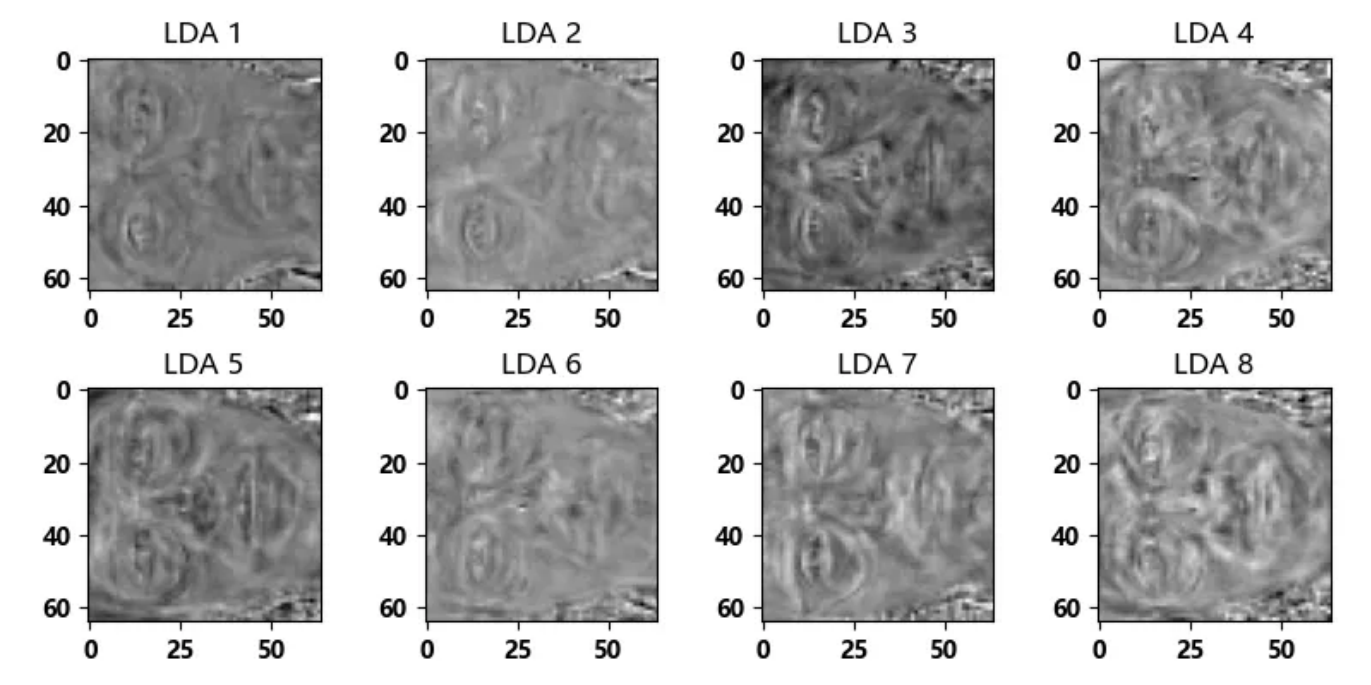

图4 LDA前八个特征向量

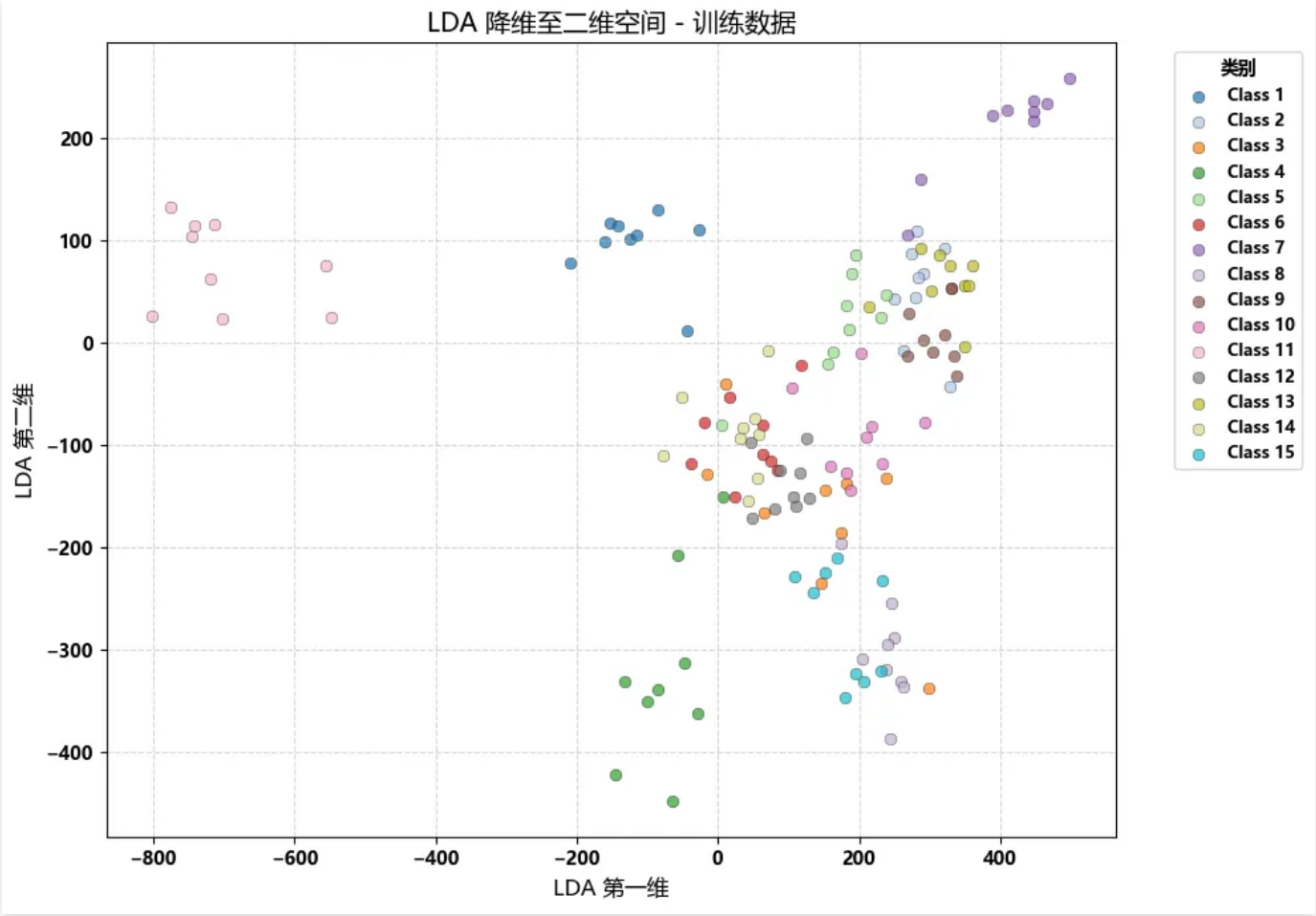
LDA 降维至二维空间 - 训练数据
LDA 第二维
LDA 第一维
200
100
0
−100
−200
−300
−400
−800
−600
−400
−200
0
200
400
类别
Class 1
Class 2
Class 3
Class 4
Class 5
Class 6
Class 7
Class 8
Class 9
Class 10
Class 11
Class 12
Class 13
Class 14
Class 15

图5

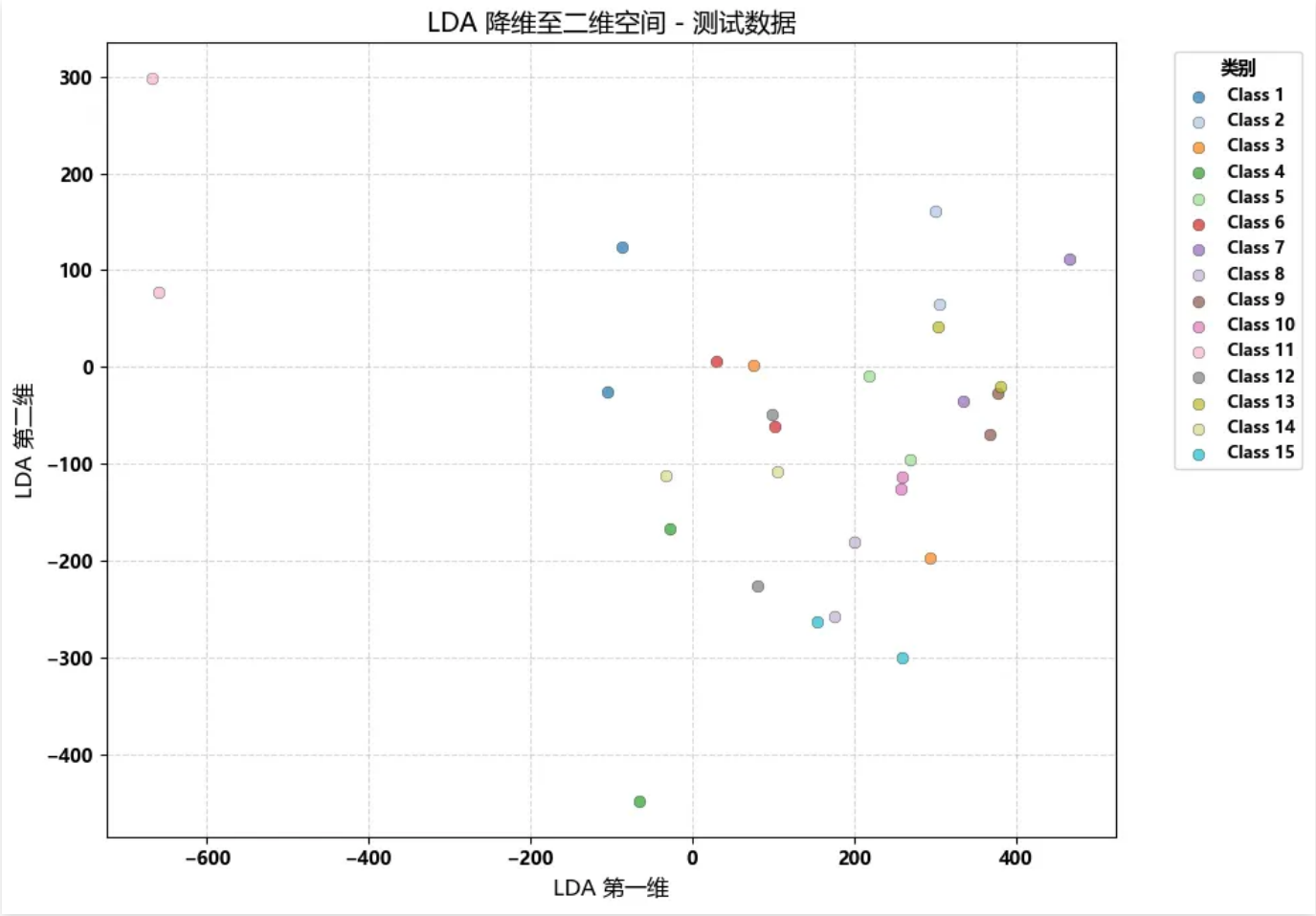

图6

4. KNN 分类和评估

原理：

KNN通过计算测试样本与训练样本的距离，选取最近的k个邻居，根据邻居的类别投票决定测试样本的类别。具体步骤：

1. **训练KNN分类器**：使用降维后的训练数据训练KNN模型。
2. **测试评估**：在降维后的测试数据上计算分类准确率。
3. **分析维度影响**：比较不同降维维度（如2维、8维）对分类准确率的影响。

```python
# ======================== KNN 类 ========================
class KNNClassifier:
    def __init__(self, k=3):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def predict(self, X):
        y_pred = []
        for x in X:
            distances = np.linalg.norm(self.X_train - x, axis=1)
            k_indices = distances.argsort()[:self.k]
            k_labels = self.y_train[k_indices]
            y_pred.append(np.bincount(k_labels).argmax())
        return np.array(y_pred)

    def score(self, X, y):
        y_pred = self.predict(X)
        return np.mean(y_pred == y)
```

```python
# ======================== KNN 分类器 ========================
results = []
knn = KNNClassifier(k=3)

for dim in dims:
    # PCA+KNN 准确率
    train_pca = pca.transform(train_data, dim=dim)
    test_pca = pca.transform(test_data, dim=dim)
    knn.fit(train_pca, train_label)
    acc_pca = knn.score(test_pca, test_label)
    pca_knn_acc.append(acc_pca)
    results.append(f"PCA + KNN Accuracy (dim={dim}): {acc_pca:.4f}")
    print(f"PCA + KNN Accuracy (dim={dim}): {acc_pca:.4f}")

    # LDA+KNN 准确率
    train_lda = lda.transform(train_data, dim=min(dim, class_number - 1))
    test_lda = lda.transform(test_data, dim=min(dim, class_number - 1))
    knn.fit(train_lda, train_label)
    acc_lda = knn.score(test_lda, test_label)
    if dim > class_number - 1:
        lda_knn_acc.append(None)
    else:
        lda_knn_acc.append(acc_lda)
    results.append(f"LDA + KNN Accuracy (dim={min(dim, class_number - 1)}): {acc_lda:.4f}")
    print(f"LDA + KNN Accuracy (dim={min(dim, class_number - 1)}): {acc_lda:.4f}")
    results.append("\n")
# 将结果写入文件
with open("result/accuracy.txt", "w", encoding="utf-8") as f:
    for line in results:
        f.write(line + "\n")

print("所有维度的 KNN 准确率已写入 result/accuracy.txt")
```

5. 拓展内容

使用SVM对降维后的数据进行分类，比较其与KNN的分类效果，分析不同分类器在降维数据上的表现差异。

```python
# ======================== SVM 分类器 ========================
results = []
svm = SVC(kernel='linear', C=1.0, random_state=42)

for dim in dims:
    # PCA+SVM 准确率
    train_pca = pca.transform(train_data, dim=dim)
    test_pca = pca.transform(test_data, dim=dim)
    svm.fit(train_pca, train_label)
    acc_pca_svm = svm.score(test_pca, test_label)
    pca_svm_acc.append(acc_pca_svm)
    results.append(f"PCA + SVM Accuracy (dim={dim}): {acc_pca_svm:.4f}")
    print(f"PCA + SVM Accuracy (dim={dim}): {acc_pca_svm:.4f}")

    # LDA+SVM 准确率
    train_lda = lda.transform(train_data, dim=min(dim, class_number - 1))
    test_lda = lda.transform(test_data, dim=min(dim, class_number - 1))
    svm.fit(train_lda, train_label)
    acc_lda_svm = svm.score(test_lda, test_label)
    if dim > class_number - 1:
        lda_svm_acc.append(None)
    else:
        lda_svm_acc.append(acc_lda_svm)
    results.append(f"LDA + SVM Accuracy (dim={min(dim, class_number - 1)}): {acc_lda_svm:.4f}")
    print(f"LDA + SVM Accuracy (dim={min(dim, class_number - 1)}): {acc_lda_svm:.4f}")
    results.append("\n")

# 将结果写入文件
with open("result/accuracy.txt", "a", encoding="utf-8") as f:
    for line in results:
        f.write(line + "\n")

print("所有维度的 SVM 准确率已写入 result/accuracy.txt")
```

结果对比与分析

维度 \ 方式	PCA+KNN	PCA+SVM	LDA+KNN	LDA+SVM
2	0.4333	0.3333	0.4667	0.4667
4	0.7333	0.5667	0.8333	0.8333

8	0.9333	0.8333	0.9333	0.9000
16 | 14	0.9333	0.9333	0.9333	0.9333
32	0.9333	0.9333		
64	0.9333	0.9333		
128	0.9333	0.9667		
256	0.9333	0.9667		
512	0.9333	0.9667		
1024	0.9333	0.9667		
2048	0.9333	0.9667		

> 注：LDA 最多只能降到 `类别数 − 1 = 14` 维，后续维度不适用于 LDA，因此未作对比。

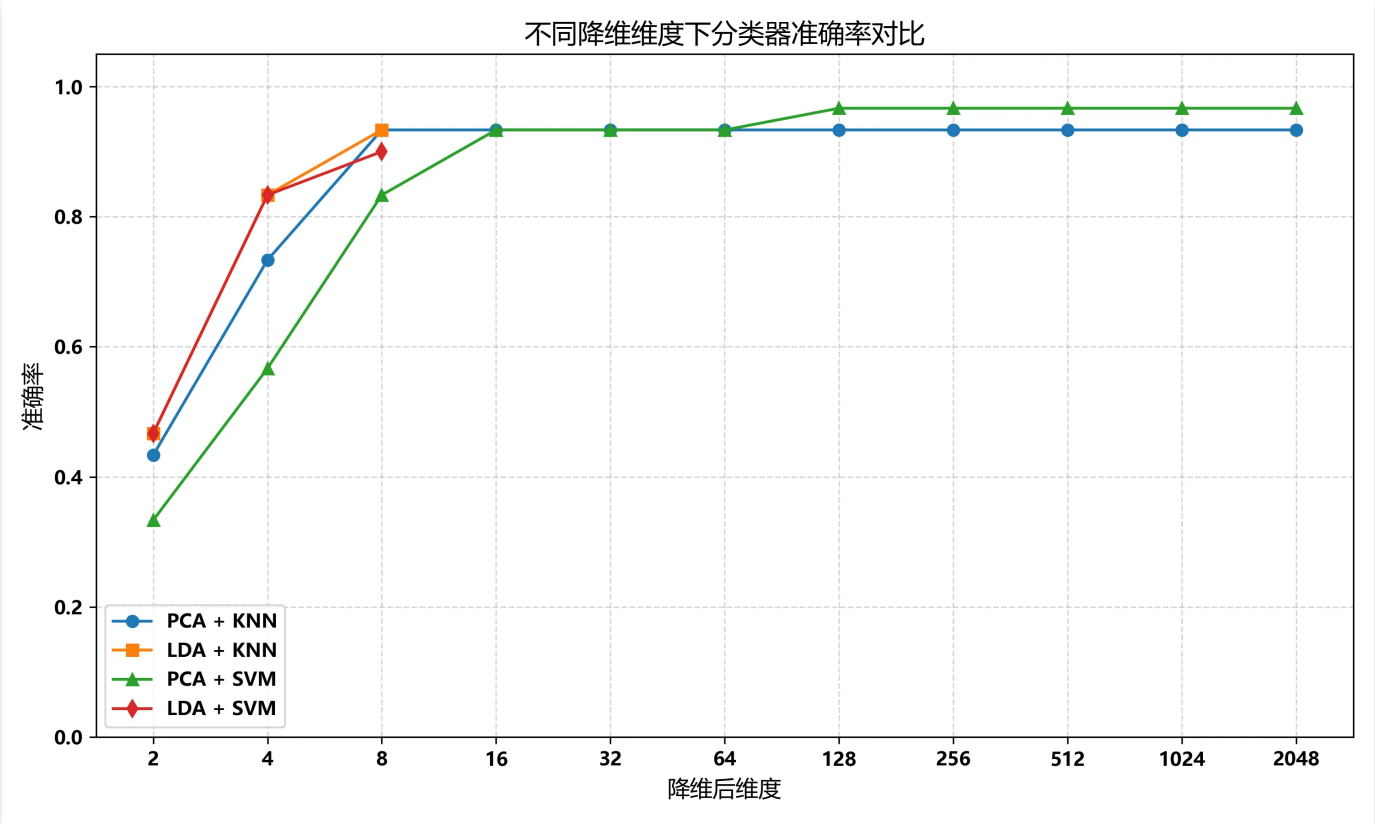

图5 不同降维维度下分类器准确率对比

1. 准确率随维度变化的趋势

- 低维（24维）时准确率相对较低，四种方法准确率均在 0.3–0.5 区间，信息损失较大；
- 中等维度（8维）时，PCA 和 LDA 的表现都接近饱和，准确率迅速上升到 0.9 左右；
- 高维（≥16维）时，PCA 的准确率趋于稳定（KNN 稳定在 93.33%，SVM 稳定上升到 96.67%），说明进一步增加维度对于准确率提升已无明显效果；
- LDA 由于理论限制，最多只能使用 14 维，准确率也在 0.9333 附近达到上限。

2. PCA vs LDA

- 在低维情况下（2~4维），**LDA 的准确率明显优于 PCA**，说明 LDA 更擅长保留类别判别信息；
- 在高维下（≥8维），二者准确率相近，PCA 略优于 LDA 的主要原因在于 PCA 能继续提升维度，而 LDA 被限制在 14 维。

3. KNN vs SVM

- SVM 的整体表现略好于 KNN，尤其是在高维下表现更稳定；
- 在低维时，KNN 对噪声和局部距离敏感，准确率偏低；
- SVM 能更有效利用边界信息，对线性可分的降维特征效果更好。