

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（ S_w ）和类间散度矩阵（ S_b ）。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。

```
1 # ===== PCA 类 =====
2 class PCA:
3     def fit(self, X):
4         self.mean = np.mean(X, axis=0)
5         X_centered = X - self.mean
6         cov = np.cov(X_centered, rowvar=False)
7         U, S, Vt = np.linalg.svd(cov)
8         self.components = Vt
9
10    def transform(self, X, dim):
11        X_centered = X - self.mean
12        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（ S_w ）**：衡量同一类别内数据的离散程度。
3. **计算类间散度矩阵（ S_b ）**：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。

```

1  # ===== LDA 类 =====
2  class LDA:
3      def fit(self, X, y):
4          self.classes = np.unique(y)
5          n_features = X.shape[1]
6          mean_overall = np.mean(X, axis=0)
7
8          Sw = np.zeros((n_features, n_features))
9          Sb = np.zeros((n_features, n_features))
10
11         for c in self.classes:
12             X_c = X[y == c]
13             mean_c = np.mean(X_c, axis=0)
14             Sw += np.dot((X_c - mean_c).T, (X_c - mean_c))
15             n_c = X_c.shape[0]
16             mean_diff = (mean_c - mean_overall).reshape(-1, 1)
17             Sb += n_c * mean_diff @ mean_diff.T
18
19         eigvals, eigvecs = np.linalg.eig(np.linalg.pinv(Sw) @ Sb)
20         sorted_indices = np.argsort(-eigvals.real)
21         self.components = eigvecs[:, sorted_indices].T.real
22
23     def transform(self, X, dim):
24         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。

```

1  # ===== PCA 降维 =====
2  pca = PCA()
3  pca.fit(train_data)
4  train_pca = pca.transform(train_data, dim=8)
5  test_pca = pca.transform(test_data, dim=8)
6
7  # 可视化前8个特征向量
8  fig, axes = plt.subplots(2, 4, figsize=(8, 4))
9  for i in range(8):
10     axes[i//4, i%4].imshow(pca.components[i].reshape(64, 64), cmap='gray')
11     axes[i//4, i%4].set_title(f'PCA {i+1}')
12 plt.tight_layout()
13 plt.savefig("result/pca_basis.png")
14 # PCA 降维到 2D 可视化
15 pca_2d = pca.transform(train_data, dim=2)
16
17 plt.figure(figsize=(10, 7))
18 colors = plt.colormaps.get_cmap('tab20')
19
20 for i in range(class_number):
21     start = i * train_test_split
22     end = (i + 1) * train_test_split
23     plt.scatter(pca_2d[start:end, 0], pca_2d[start:end, 1],
24                label=f'Class {i+1}', color=colors(i / class_number), s=40
25                , alpha=0.7, edgecolors='k', linewidths=0.3)
26
27 plt.title("PCA 降维至二维空间 - 15类人脸分布", fontsize=14)
28 plt.xlabel("PCA 第一维", fontsize=12)
29 plt.ylabel("PCA 第二维", fontsize=12)
30 plt.grid(True, linestyle='--', alpha=0.5)
31 plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', fontsize=9, title=
32 "类别")
33 plt.tight_layout()
34 plt.savefig("result/pca_2d.png", dpi=300)
35 plt.show()

```

```

1  # ===== LDA 降维 =====
2  lda = LDA()
3  lda.fit(train_data, train_label)
4  train_lda = lda.transform(train_data, dim=8)
5  test_lda = lda.transform(test_data, dim=8)
6
7  # 可视化前8个特征向量
8  fig, axes = plt.subplots(2, 4, figsize=(8, 4))
9  for i in range(8):
10     axes[i//4, i%4].imshow(lda.components[i].reshape(64, 64), cmap='gray')
11     axes[i//4, i%4].set_title(f'LDA {i+1}')
12 plt.tight_layout()
13 plt.savefig("result/lda_basis.png")
14 # LDA 降维到 2D 可视化
15 lda_2d = lda.transform(train_data, dim=2)
16
17 plt.figure(figsize=(10, 7))
18 colors = plt.colormaps.get_cmap('tab20')
19
20 for i in range(class_number):
21     start = i * train_test_split
22     end = (i + 1) * train_test_split
23     plt.scatter(lda_2d[start:end, 0], lda_2d[start:end, 1],
24                label=f'Class {i+1}', color=colors(i / class_number), s=40
25                , alpha=0.7, edgecolors='k', linewidths=0.3)
26
27 plt.title("LDA 降维至二维空间 - 15类人脸分布", fontsize=14)
28 plt.xlabel("LDA 第一维", fontsize=12)
29 plt.ylabel("LDA 第二维", fontsize=12)
30 plt.grid(True, linestyle='--', alpha=0.5)
31 plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', fontsize=9, title=
32 "类别")
33 plt.tight_layout()
34 plt.savefig("result/lda_2d.png", dpi=300)
35 plt.show()

```

结果：

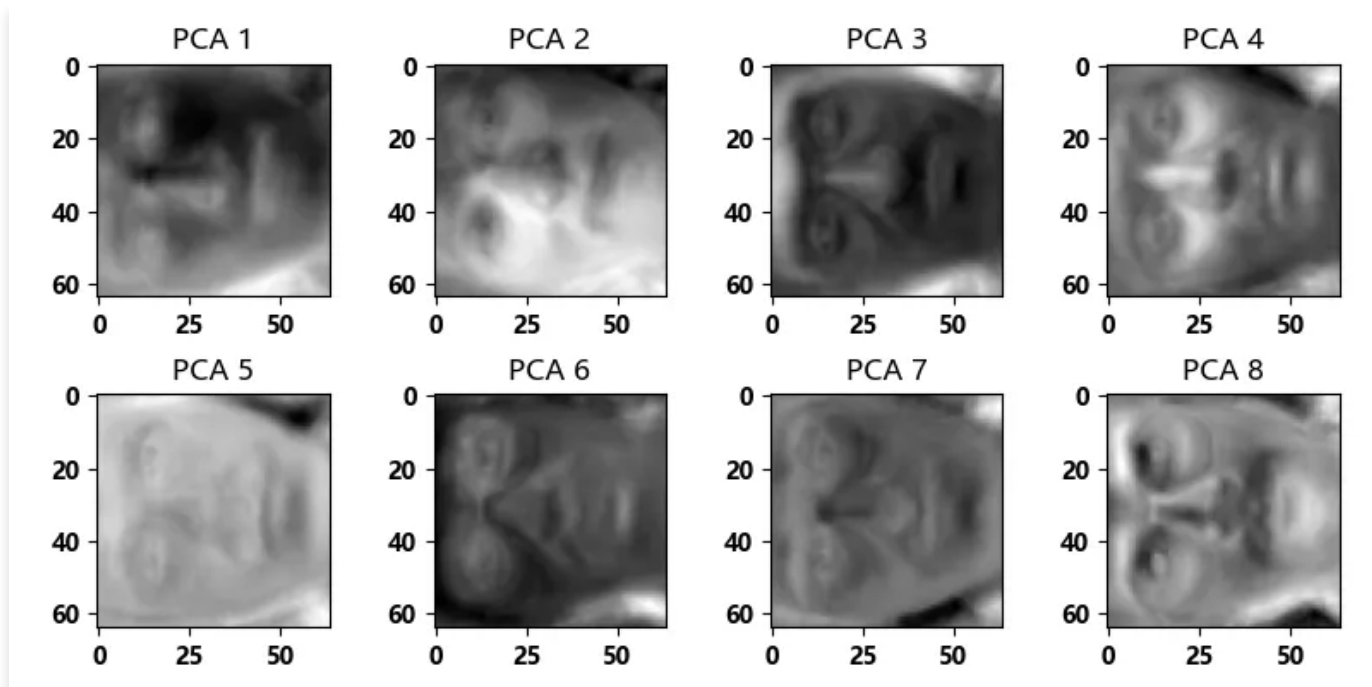


图1 PCA前八个特征向量

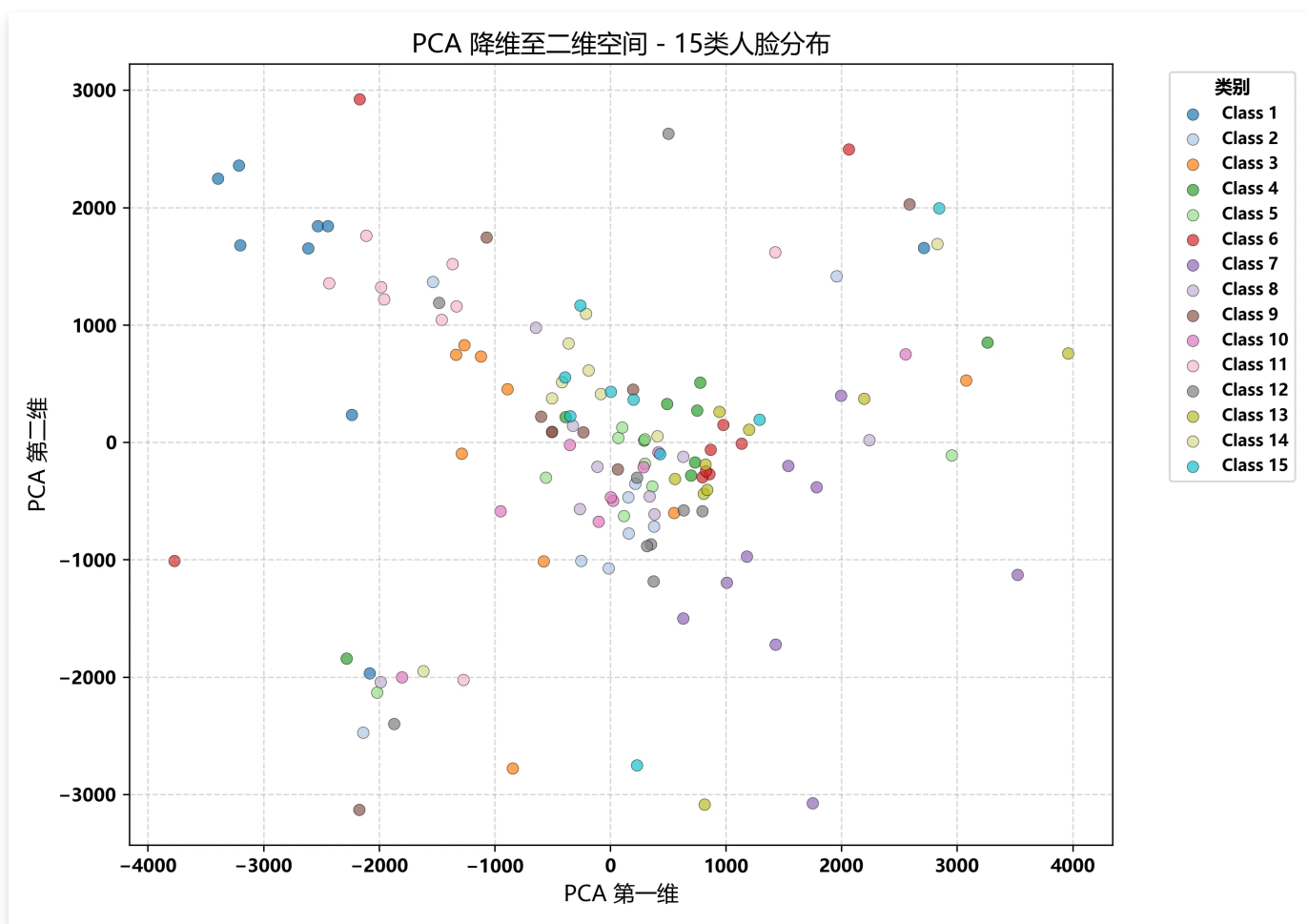


图2 PCA降维至二维空间-15类人脸分布

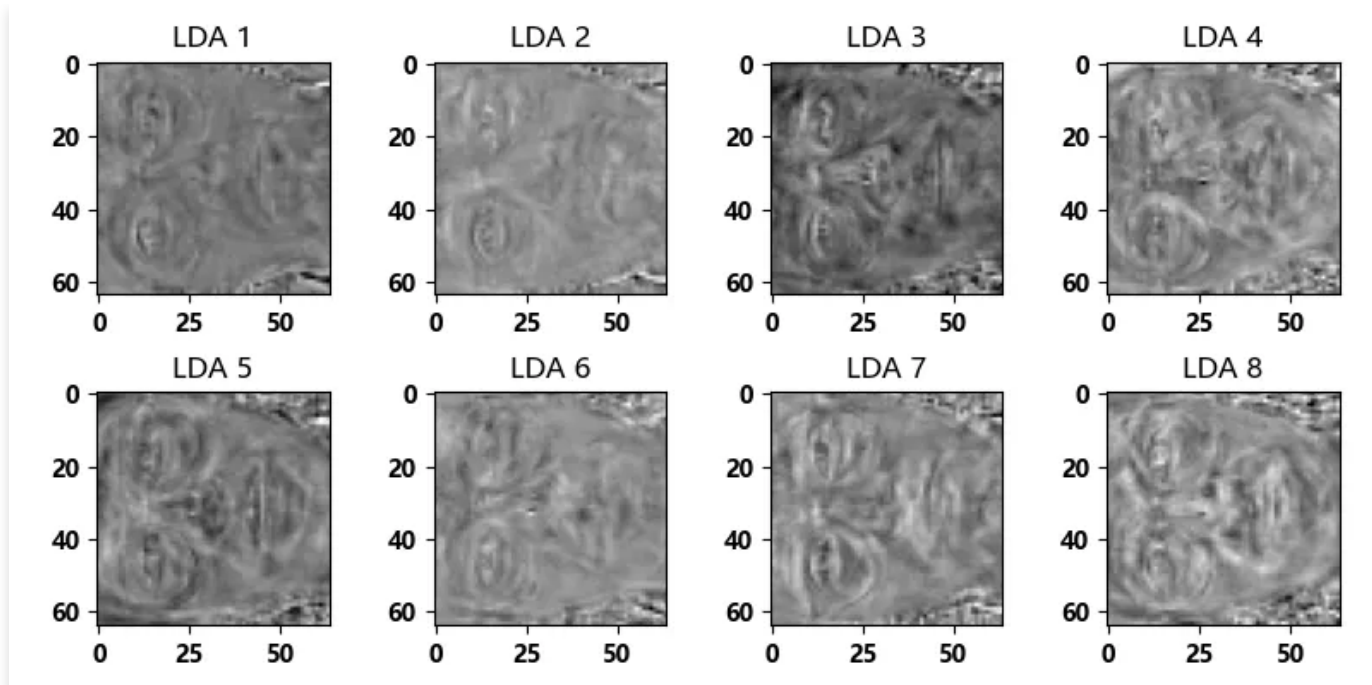


图3 LDA前八个特征向量

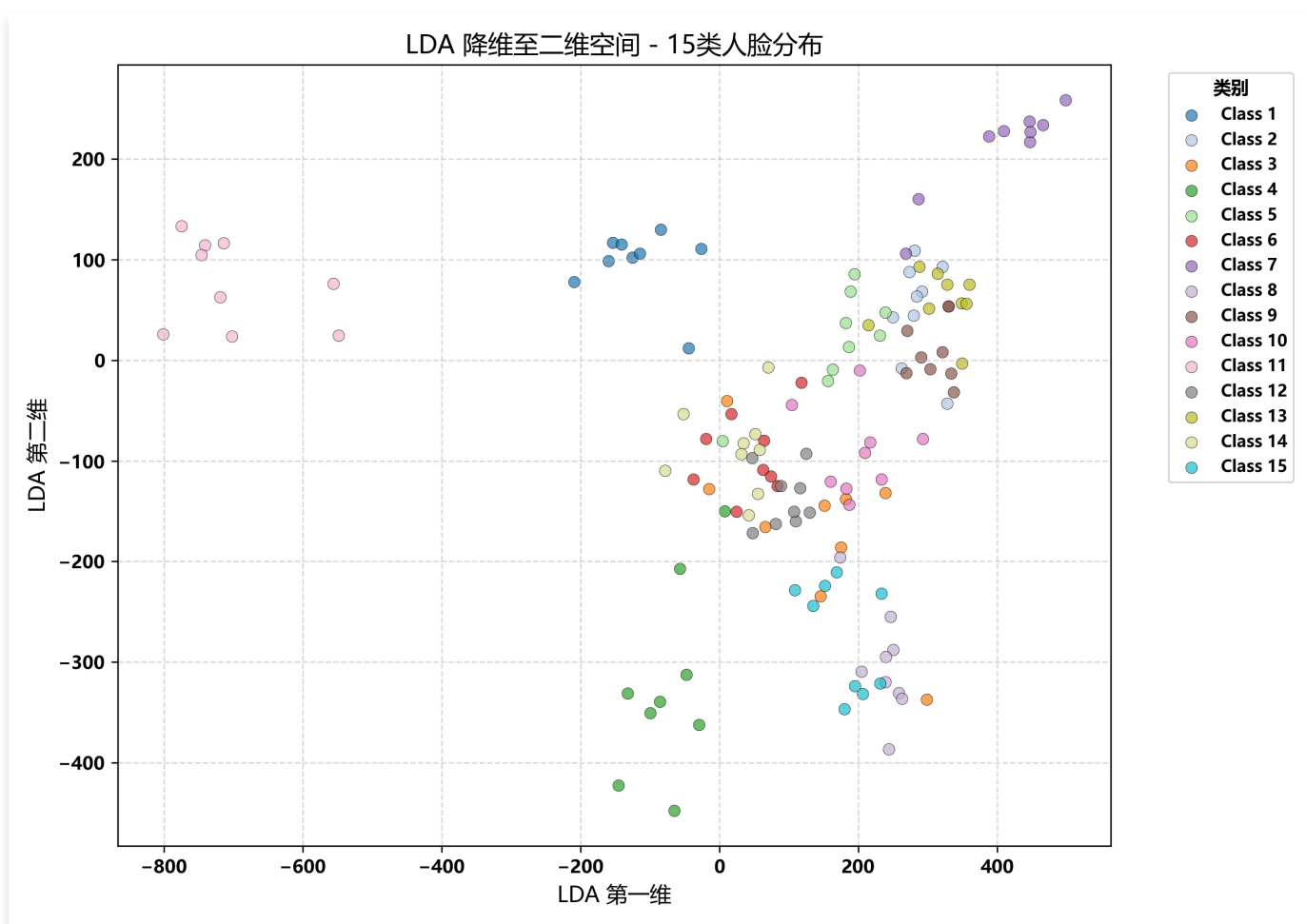


图4 LDA降维至二维空间-15类人脸分布

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。

```
1  # ===== KNN 类 =====
2  class KNNClassifier:
3      def __init__(self, k=3):
4          self.k = k
5
6      def fit(self, X, y):
7          self.X_train = X
8          self.y_train = y
9
10     def predict(self, X):
11         y_pred = []
12         for x in X:
13             distances = np.linalg.norm(self.X_train - x, axis=1)
14             k_indices = distances.argsort()[:self.k]
15             k_labels = self.y_train[k_indices]
16             y_pred.append(np.bincount(k_labels).argmax())
17         return np.array(y_pred)
18
19     def score(self, X, y):
20         y_pred = self.predict(X)
21         return np.mean(y_pred == y)
```



```

1  # ===== KNN 分类器 =====
2  results = []
3  knn = KNNClassifier(k=3)
4
5  for dim in dims:
6      # PCA+KNN 准确率
7      train_pca = pca.transform(train_data, dim=dim)
8      test_pca = pca.transform(test_data, dim=dim)
9      knn.fit(train_pca, train_label)
10     acc_pca = knn.score(test_pca, test_label)
11     pca_knn_acc.append(acc_pca)
12     results.append(f"PCA + KNN Accuracy (dim={dim}): {acc_pca:.4f}")
13     print(f"PCA + KNN Accuracy (dim={dim}): {acc_pca:.4f}")
14
15     # LDA+KNN 准确率
16     train_lda = lda.transform(train_data, dim=min(dim, class_number - 1))
17     test_lda = lda.transform(test_data, dim=min(dim, class_number - 1))
18     knn.fit(train_lda, train_label)
19     acc_lda = knn.score(test_lda, test_label)
20     if dim > class_number - 1:
21         lda_knn_acc.append(None)
22     else:
23         lda_knn_acc.append(acc_lda)
24     results.append(f"LDA + KNN Accuracy (dim={min(dim, class_number - 1)}): {acc_lda:.4f}")
25     print(f"LDA + KNN Accuracy (dim={min(dim, class_number - 1)}): {acc_lda:.4f}")
26     results.append("\n")
27     # 将结果写入文件
28     with open("result/accuracy.txt", "w", encoding="utf-8") as f:
29         for line in results:
30             f.write(line + "\n")
31
32     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。

```

1  # ===== SVM 分类器 =====
2  results = []
3  svm = SVC(kernel='linear', C=1.0, random_state=42)
4
5  for dim in dims:
6      # PCA+SVM 准确率
7      train_pca = pca.transform(train_data, dim=dim)
8      test_pca = pca.transform(test_data, dim=dim)
9      svm.fit(train_pca, train_label)
10     acc_pca_svm = svm.score(test_pca, test_label)
11     pca_svm_acc.append(acc_pca_svm)
12     results.append(f"PCA + SVM Accuracy (dim={dim}): {acc_pca_svm:.4f}")
13     print(f"PCA + SVM Accuracy (dim={dim}): {acc_pca_svm:.4f}")
14
15     # LDA+SVM 准确率
16     train_lda = lda.transform(train_data, dim=min(dim, class_number - 1))
17     test_lda = lda.transform(test_data, dim=min(dim, class_number - 1))
18     svm.fit(train_lda, train_label)
19     acc_lda_svm = svm.score(test_lda, test_label)
20     if dim > class_number - 1:
21         lda_svm_acc.append(None)
22     else:
23         lda_svm_acc.append(acc_lda_svm)
24     results.append(f"LDA + SVM Accuracy (dim={min(dim, class_number - 1)}): {acc_lda_svm:.4f}")
25     print(f"LDA + SVM Accuracy (dim={min(dim, class_number - 1)}): {acc_lda_svm:.4f}")
26     results.append("\n")
27
28     # 将结果写入文件
29     with open("result/accuracy.txt", "a", encoding="utf-8") as f:
30         for line in results:
31             f.write(line + "\n")
32
33     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。

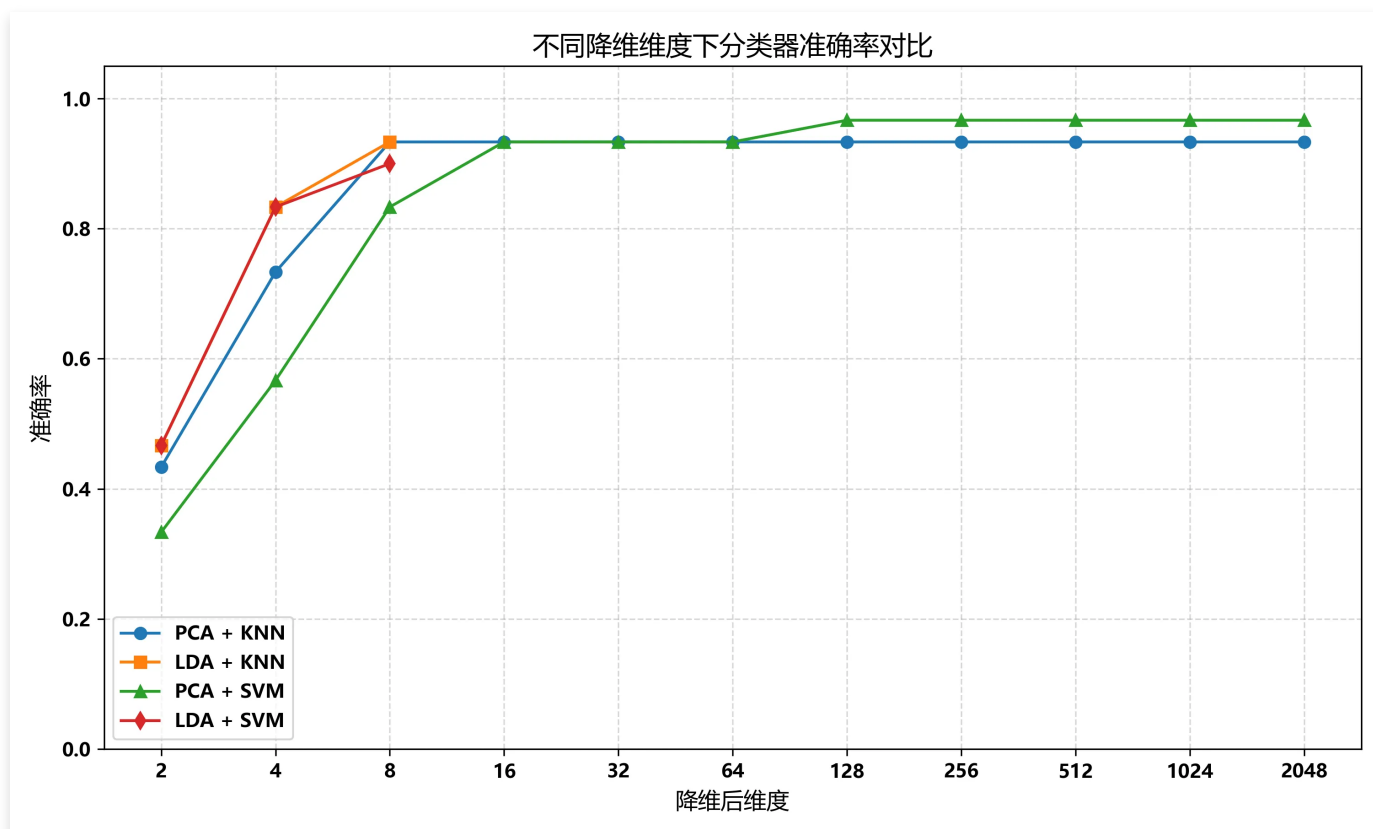


图5 不同降维维度下分类器准确率对比

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 能持续提升维度，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。