**Introduction**

一个著名手写数字数据库：MNIST，利用解码文件decode解压后采用该数据分别做如下实验：

1）用聚类方法对数字数据集进行聚类，计算准确率；2）采用tsne画出散点图分布效果；3）采用SVM分类，计算分类准确率

先利用main文件和解码文件decode导入MNIST数据，并且把图像数据转换为数字编码：

%% 1. 加载MNIST数据

train\_x\_file = 'train-images.idx3-ubyte'; % 60000个训练集图片

test\_x\_file = 't10k-images.idx3-ubyte'; % 10000个测试集图片

train\_y\_file = 'train-labels.idx1-ubyte'; % 60000个训练集图片对应的数字

test\_y\_file = 't10k-labels.idx1-ubyte'; % 10000个测试集图片对应的数字

fprintf('正在加载MNIST数据...\n');

train\_x = decodefile(train\_x\_file, 'image');

test\_x = decodefile(test\_x\_file, 'image');

train\_y = decodefile(train\_y\_file, 'label');

test\_y = decodefile(test\_y\_file, 'label');

% 重塑图像数据

train\_x\_matrix = reshape(train\_x, 28, 28, 60000);

train\_x\_matrix = permute(train\_x\_matrix, [2 1 3]);

test\_x\_matrix = reshape(test\_x, 28, 28, 10000);

test\_x\_matrix = permute(test\_x\_matrix, [2 1 3]);

% 将图像数据转换为二维矩阵（样本×特征）

train\_data = double(reshape(train\_x\_matrix, [], 60000))';

test\_data = double(reshape(test\_x\_matrix, [], 10000))';

train\_labels = double(train\_y);

test\_labels = double(test\_y);

fprintf('训练集: %d个样本, 测试集: %d个样本\n', size(train\_data,1), size(test\_data,1));

**（1）K-means聚类**

**1. 肘部法则分析**

通过计算不同K值对应的簇内平方和（Within-Cluster Sum of Squares, WCSS），寻找最佳的聚类数量K。肘部法则认为，当K值增加到真实聚类数时，WCSS的下降幅度会突然变缓，形成"肘部"。因此我们期望在K=10附近看到明显的拐点，因为MNIST数据集中有10个数字类别。

% 2.1 确定最佳K值 - 肘部法则

fprintf('正在使用肘部法则确定最佳K值...\n');

max\_k = 15;

within\_cluster\_sum = zeros(max\_k, 1);

for k = 1:max\_k

[~, ~, sumd] = kmeans(train\_data, k, 'MaxIter', 100, 'Replicates', 3);

within\_cluster\_sum(k) = sum(sumd);

fprintf('K=%d, 簇内平方和: %.2f\n', k, within\_cluster\_sum(k));

end

% 绘制肘部法则图

figure('Position', [100, 100, 1200, 800]);

subplot(2, 3, 1);

plot(1:max\_k, within\_cluster\_sum, 'bo-', 'LineWidth', 2, 'MarkerSize', 8);

xlabel('簇数量 K');

ylabel('簇内平方和');

title('肘部法则 - 确定最佳K值');

grid on;

% 计算斜率变化来确定肘部点

slopes = diff(within\_cluster\_sum);

slope\_ratios = slopes(1:end-1) ./ slopes(2:end);

[~, optimal\_k\_idx] = max(slope\_ratios);

optimal\_k = optimal\_k\_idx + 2; % 因为diff会减少一个元素

hold on;

plot(optimal\_k, within\_cluster\_sum(optimal\_k), 'ro', 'MarkerSize', 10, 'LineWidth', 2);

legend('簇内平方和', sprintf('建议K值: %d', optimal\_k));

正在使用肘部法则确定最佳K值...

K=1, 簇内平方和: 205706725984.24

K=2, 簇内平方和: 192524742113.23

K=3, 簇内平方和: 183140062051.96

K=4, 簇内平方和: 176038479216.84

K=5, 簇内平方和: 169452575709.56

K=6, 簇内平方和: 164727478679.33

警告: 未能在第 1 次重复期间的 100 次迭代后收敛。

K=7, 簇内平方和: 161254708035.25

K=8, 簇内平方和: 158166404612.59

K=9, 簇内平方和: 155829875866.49

警告: 未能在第 1 次重复期间的 100 次迭代后收敛。

警告: 未能在第 2 次重复期间的 100 次迭代后收敛。

警告: 未能在第 3 次重复期间的 100 次迭代后收敛。   
K=10, 簇内平方和: 153024092320.02

K=11, 簇内平方和: 150859091990.52

K=12, 簇内平方和: 148622838394.25

警告: 未能在第 3 次重复期间的 100 次迭代后收敛。

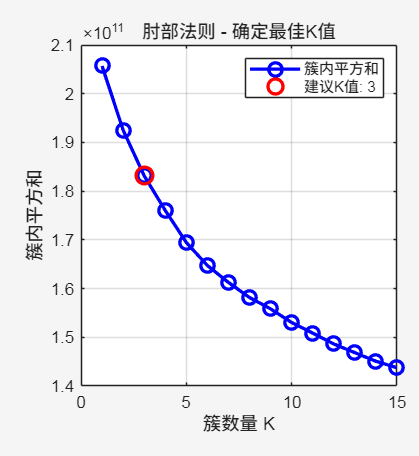
K=13, 簇内平方和: 146839486671.26

警告: 未能在第 1 次重复期间的 100 次迭代后收敛。

警告: 未能在第 2 次重复期间的 100 次迭代后收敛。

K=14, 簇内平方和: 145077164445.17

K=15, 簇内平方和: 143749261737.67



以上的结果表明WCSS随着K值增加持续下降，这是正常现象；在K=1到K=5区间下降幅度相对较大，从K=5开始下降幅度明显减缓，其中在K=3处有最大斜率，但是考虑到我们的任务需要对手写数字进行识别，所以还是建议取K=10

另外，多个K值出现收敛警告，说明MNIST数据聚类难度较大

% 2.2 使用轮廓系数评估聚类质量

fprintf('\n正在计算轮廓系数评估聚类质量...\n');

k\_values = [5, 8, 10, 12, 15];

silhouette\_scores = zeros(length(k\_values), 1);

% 使用部分数据计算轮廓系数（计算量较大）

sample\_size = 2000;

sample\_idx = randperm(size(train\_data, 1), sample\_size);

sample\_data = train\_data(sample\_idx, :);

for i = 1:length(k\_values)

k = k\_values(i);

[idx, ~] = kmeans(sample\_data, k, 'MaxIter', 100);

silhouette\_vals = silhouette(sample\_data, idx);

silhouette\_scores(i) = mean(silhouette\_vals);

fprintf('K=%d, 平均轮廓系数: %.4f\n', k, silhouette\_scores(i));

end

subplot(2, 3, 2);

bar(k\_values, silhouette\_scores, 'FaceColor', [0.2, 0.6, 0.8]);

xlabel('簇数量 K');

ylabel('平均轮廓系数');

title('轮廓系数分析');

grid on;

轮廓系数衡量每个样本点与自身簇的紧密度和与其他簇的分离度，取值范围[-1,1]，值越大表示聚类效果越好。

这里选择 [5, 8, 10, 12, 15] 这几个K值主要是出于计算效率和实际意义的考虑。首先轮廓系数计算复杂度为O(n²)，对于60000个样本计算量极大，所以为了避免大量计算选取一些情况来做分析。其次，MNIST有10个数字类别，所以应该重点关注10附近的K值，选取的这几个数覆盖了小于10、等于10、大于10的情况，能够比较全面地呈现结果。

正在计算轮廓系数评估聚类质量...

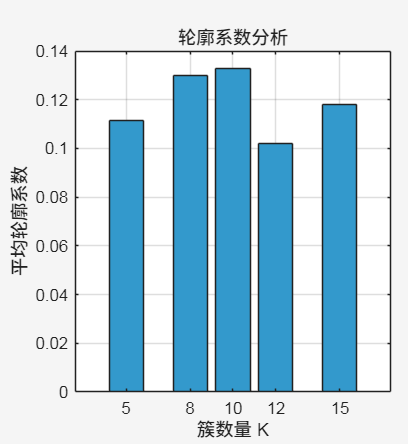
K=5, 平均轮廓系数: 0.1113

K=8, 平均轮廓系数: 0.1298

K=10, 平均轮廓系数: 0.1330

K=12, 平均轮廓系数: 0.1020

K=15, 平均轮廓系数: 0.1181



观察轮廓系数的数据和可视化图像，发现所有轮廓系数都较低（<0.15），说明MNIST数据的聚类结构不够清晰，在K=10时获得最高轮廓系数0.1330，验证了选择K=10的合理性。

轮廓系数整体偏低反映了手写数字数据的复杂性和重叠性。

% 2.3 执行最终K-means聚类

final\_k = 10; % MNIST有10个数字类别

fprintf('\n执行最终K-means聚类 (K=%d)...\n', final\_k);

[idx, centers, sumd, distances] = kmeans(train\_data, final\_k, ...

'MaxIter', 200, 'Replicates', 5, 'Display', 'final');

% 计算聚类准确率

accuracy = calculate\_clustering\_accuracy(idx, train\_labels, final\_k);

fprintf('K-means聚类准确率: %.2f%%\n', accuracy \* 100);

%% 辅助函数

function accuracy = calculate\_clustering\_accuracy(cluster\_labels, true\_labels, k)

% 将聚类标签与真实标签进行最佳匹配

accuracy = 0;

for i = 1:k

cluster\_mask = (cluster\_labels == i);

if sum(cluster\_mask) > 0

true\_labels\_in\_cluster = true\_labels(cluster\_mask);

most\_common\_label = mode(true\_labels\_in\_cluster);

cluster\_accuracy = sum(true\_labels\_in\_cluster == most\_common\_label) / sum(cluster\_mask);

accuracy = accuracy + cluster\_accuracy \* sum(cluster\_mask);

end

end

accuracy = accuracy / length(cluster\_labels);

end

执行最终K-means聚类 (K=10)...

第 1 次重复，56 次迭代，总距离 = 1.52993e+11。

第 2 次重复，39 次迭代，总距离 = 1.53455e+11。

第 3 次重复，200 次迭代，总距离 = 1.53455e+11。  
警告: 未能在第 3 次重复期间的 200 次迭代后收敛。

第 4 次重复，140 次迭代，总距离 = 1.53455e+11。

第 5 次重复，170 次迭代，总距离 = 1.53455e+11。  
距离的最佳总和 = 1.52993e+11

K-means聚类准确率: 59.07%

最后，使用K=10对全部60000个训练样本进行K-means聚类，并评估聚类准确率。期望获得较高的聚类准确率，理想情况下应超过70%，但是现在只有约60%，说明模型还有很大的提升空间。

% 2.4 可视化聚类结果

% 显示聚类中心

subplot(2, 3, 3);

montage\_array = zeros(28, 28, 1, final\_k);

for i = 1:final\_k

center\_img = reshape(centers(i, :), 28, 28);

montage\_array(:, :, 1, i) = center\_img;

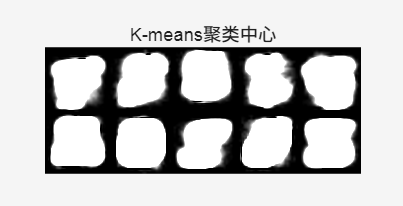
end

montage(montage\_array, 'Size', [2, 5]);

title('K-means聚类中心');

聚类中心可视化显示了10个聚类中心的图像（28×28像素），期望看到的是每个聚类中心能代表一个数字的"平均"模样，但是由于本实验获得的聚类准确率为59.07%，意味着每个聚类中约40%的样本属于其他数字类别。当计算聚类中心（各特征维度的平均值）时，这些错误分类的样本会"稀释"真正数字的特征，导致中心图像模糊。

K-means基于欧氏距离，但像素空间的欧氏距离不能很好捕捉数字的语义相似性，两个形状相似但位置略有偏移的数字，在像素空间可能距离很远。



% 显示每个聚类的样本分布

subplot(2, 3, 4);

cluster\_counts = zeros(final\_k, 1);

for i = 1:final\_k

cluster\_counts(i) = sum(idx == i);

end

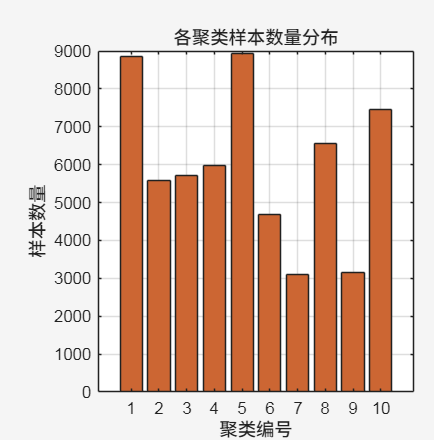
bar(1:final\_k, cluster\_counts, 'FaceColor', [0.8, 0.4, 0.2]);

xlabel('聚类编号');

ylabel('样本数量');

title('各聚类样本数量分布');

grid on;



发现样本数量差距较大，不是很均衡，可能很多都被错判了

% 显示聚类纯度（每个聚类中主要数字的占比）

subplot(2, 3, 5);

purity\_scores = zeros(final\_k, 1);

for i = 1:final\_k

cluster\_indices = (idx == i);

true\_labels\_in\_cluster = train\_labels(cluster\_indices);

if ~isempty(true\_labels\_in\_cluster)

most\_common\_label = mode(true\_labels\_in\_cluster);

purity\_scores(i) = sum(true\_labels\_in\_cluster == most\_common\_label) / length(true\_labels\_in\_cluster);

end

end

bar(1:final\_k, purity\_scores \* 100, 'FaceColor', [0.4, 0.8, 0.4]);

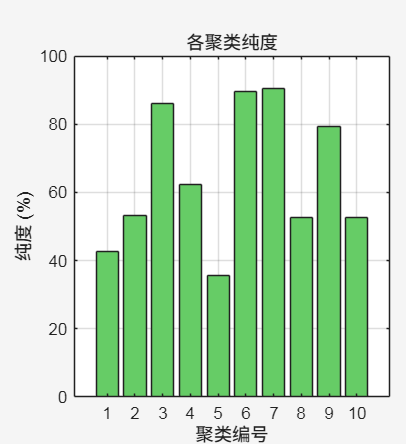
xlabel('聚类编号');

ylabel('纯度 (%)');

title('各聚类纯度');

ylim([0, 100]);

grid on;



有超过半数的类别中纯度较低，同样反映了模型性能不佳的问题

% 显示距离热图

subplot(2, 3, 6);

center\_distances = pdist2(centers, centers);

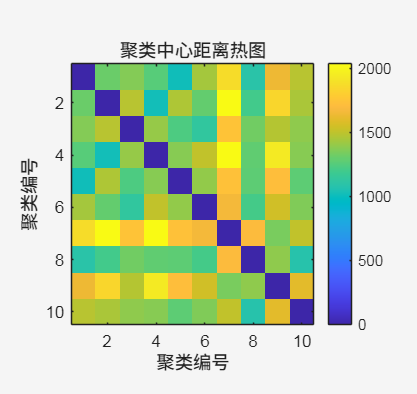
imagesc(center\_distances);

colorbar;

title('聚类中心距离热图');

xlabel('聚类编号');

ylabel('聚类编号');



这幅图反映了聚类中心之间的欧氏距离，理论上来讲相似数字（如3和8）的中心距离较近，但从实际结果看似乎并不完全服从这个规律。

总结一下结果不好可能的原因。

首先，K-means作为无监督学习方法，完全依赖数据本身的分布特性进行聚类，无法利用样本的真实标签信息，这构成了根本性的性能天花板。

其次，我直接使用784维原始像素作为特征，相邻像素间的高度相关性造成了大量冗余信息，而大量零值背景像素稀释了关键笔画特征的重要性。

% 检查原始数据维度

fprintf('train\_x原始大小: %s\n', mat2str(size(train\_x)));

fprintf('train\_x\_matrix大小: %s\n', mat2str(size(train\_x\_matrix)));

fprintf('train\_data大小: %s\n', mat2str(size(train\_data)));

% 检查一个样本的像素值分布

sample\_idx = 1;

sample\_pixels = train\_data(sample\_idx, :);

fprintf('样本%d - 像素值范围: [%.1f, %.1f]\n', sample\_idx, min(sample\_pixels), max(sample\_pixels));

fprintf('样本%d - 零像素比例: %.1f%%\n', sample\_idx, sum(sample\_pixels == 0) / 784 \* 100);

train\_x原始大小: [47040000 1]  
train\_x\_matrix大小: [28 28 60000]  
train\_data大小: [60000 784]

样本1 - 像素值范围: [0.0, 255.0]  
样本1 - 零像素比例: 78.8%

例如这里随机选取了一个样本（标签是“5”，算是笔画比较多的），其零像素比例达到了78.8%，这能很好的体现出上述问题。

还有一个很重要的问题，在像素空间中使用欧氏距离无法捕捉数字的语义相似性。两个形状相同但位置略有偏移的数字，在像素空间中可能距离很远；而两个不同数字如果恰好有相似的像素分布，则会被错误地聚在一起。

最后，在某些情况下，同一数字的不同写法之间的差异（如倾斜的“1”与直立的“1”）可能大于不同数字间的差异（如“1”与“7”），这与K-means的基本假设严重冲突。

**（2）t-SNE可视化分析**

t-SNE可视化的主要目的是将高维的像素数据投影到二维平面，直观展示不同数字在特征空间中的分布情况，从而深入理解数据的内在结构特性。

%% 3. 任务2: t-SNE可视化

% 使用PCA先降维到50维，再t-SNE（提高速度和稳定性）

fprintf('使用PCA预处理后进行t-SNE...\n');

[coeff, score\_pca, ~] = pca(test\_data, 'NumComponents', 50);

test\_data\_pca = test\_data \* coeff(:, 1:50);

首先对测试集的10000个样本进行预处理。原始样本包含784个像素特征，在这种高维空间中进行t-SNE计算会很慢，所以采用PCA方法先将数据维度从784维降至50维，去除数据中的噪声干扰，同时保留了绝大部分的特征信息。

% 执行t-SNE

fprintf('正在进行t-SNE降维...\n');

Y = tsne(test\_data\_pca, 'NumDimensions', 2, 'Perplexity', 30, 'Verbose', 1);

完成PCA预处理后再使用t-SNE算法进行非线性降维。

参数设置：

* 目标维度为2维，便于在平面中可视化展示；
* Perplexity参数设置为30，这是一个经验值，能够较好地平衡局部结构和全局结构的关系；
* 开启进度输出功能，便于监控算法运行状态。

经过计算得到10000个样本在二维空间中的坐标。

|==============================================|  
| ITER | KL DIVERGENCE | NORM GRAD USING |  
| | FUN VALUE USING | EXAGGERATED DIST|  
| | EXAGGERATED DIST| OF X |  
| | OF X | |  
|==============================================|  
| 20 | 2.815870e+01 | 6.616107e-06 |

| 40 | 2.254794e+01 | 1.026015e-03 |

| 60 | 2.095453e+01 | 7.041628e-04 |

| 80 | 2.018212e+01 | 3.040564e-04 |

| 100 | 3.557494e+00 | 1.100903e-04 |

| 120 | 3.092140e+00 | 5.010873e-05 |

| 140 | 2.827739e+00 | 4.192055e-05 |

| 160 | 2.641382e+00 | 2.560621e-05 |

| 180 | 2.504309e+00 | 2.348897e-05 |

| 200 | 2.399976e+00 | 2.502613e-05 |

| 220 | 2.318748e+00 | 2.200505e-05 |

| 240 | 2.253958e+00 | 1.876998e-05 |

| 260 | 2.180151e+00 | 1.310452e-05 |

| 280 | 2.091773e+00 | 4.189834e-05 |

| 300 | 2.033150e+00 | 1.479593e-05 |

| 320 | 1.994252e+00 | 1.116862e-05 |

| 340 | 1.966768e+00 | 1.381216e-05 |

| 360 | 1.947470e+00 | 6.310674e-05 |

| 380 | 1.932587e+00 | 1.390967e-05 |

| 400 | 1.920335e+00 | 1.036378e-05 |

| 420 | 1.910433e+00 | 2.071109e-05 |

| 440 | 1.902352e+00 | 1.041383e-05 |

| 460 | 1.895145e+00 | 9.488835e-06 |

| 480 | 1.888123e+00 | 1.139032e-05 |

| 500 | 1.881358e+00 | 9.907134e-06 |

| 520 | 1.874711e+00 | 9.075541e-06 |

| 540 | 1.868733e+00 | 1.165953e-05 |

| 560 | 1.862870e+00 | 1.253739e-05 |

| 580 | 1.856688e+00 | 1.127679e-05 |

| 600 | 1.850556e+00 | 1.037117e-05 |

| 620 | 1.844963e+00 | 9.928059e-06 |

| 640 | 1.839412e+00 | 1.086930e-05 |

| 660 | 1.834115e+00 | 1.251254e-05 |

| 680 | 1.828853e+00 | 8.947699e-06 |

| 700 | 1.824418e+00 | 9.376897e-06 |

| 720 | 1.820188e+00 | 8.687382e-06 |

| 740 | 1.816706e+00 | 1.021662e-05 |

| 760 | 1.813741e+00 | 8.243819e-06 |

| 780 | 1.811022e+00 | 9.038826e-06 |

| 800 | 1.808371e+00 | 1.062850e-05 |

| 820 | 1.805877e+00 | 1.861914e-05 |

| 840 | 1.803504e+00 | 1.010233e-05 |

| 860 | 1.801372e+00 | 8.919175e-06 |

| 880 | 1.799468e+00 | 8.754370e-06 |

| 900 | 1.797852e+00 | 1.408720e-05 |

| 920 | 1.796392e+00 | 8.075294e-06 |

| 940 | 1.794746e+00 | 8.374377e-06 |

| 960 | 1.792791e+00 | 8.634672e-06 |

| 980 | 1.790693e+00 | 8.134524e-06 |

| 1000 | 1.788564e+00 | 1.160227e-05 |

同步输出了t-SNE算法运行时的详细迭代日志，包含三个主要指标：ITER表示迭代次数，KL DIVERGENCE FUN VALUE是核心优化目标函数的值，NORM GRAD表示梯度的大小。

从迭代过程可以看出明显的两个阶段。前100次迭代属于第一阶段，KL散度值从28.16快速下降到3.56，梯度范数也相对较大，这说明算法正在快速学习数据的主要结构。从第100次迭代开始进入第二阶段，KL散度从3.56缓慢下降到最终的1.79，梯度范数也保持在较低水平。这个阶段算法在进行精细调整，逐步优化低维空间中数据点的相对位置。在整个1000次迭代过程中，KL散度始终在持续下降，梯度范数也保持正值，这表明算法运行正常且收敛状态良好。最终KL散度值稳定在1.79左右，说明t-SNE成功地将高维数据的内在结构在二维空间中进行了合理的保持。

% 可视化t-SNE结果

figure('Position', [100, 100, 1400, 600]);

% 子图1: 按真实标签着色

subplot(1, 2, 1);

gscatter(Y(:,1), Y(:,2), test\_labels, [], 'o', 6, 'on', 'northeastoutside');

title('t-SNE可视化 - 按真实标签着色');

xlabel('t-SNE 1'); ylabel('t-SNE 2');

% 子图2: 按数字类别显示不同的标记

subplot(1, 2, 2);

colors = lines(10);

for digit = 0:9

mask = (test\_labels == digit);

scatter(Y(mask, 1), Y(mask, 2), 30, colors(digit+1, :), 'filled', 'Marker', get\_marker(digit+1));

hold on;

end

title('t-SNE可视化 - 不同标记和颜色');

xlabel('t-SNE 1');

ylabel('t-SNE 2');

legend('0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '位置', 'northeastoutside');

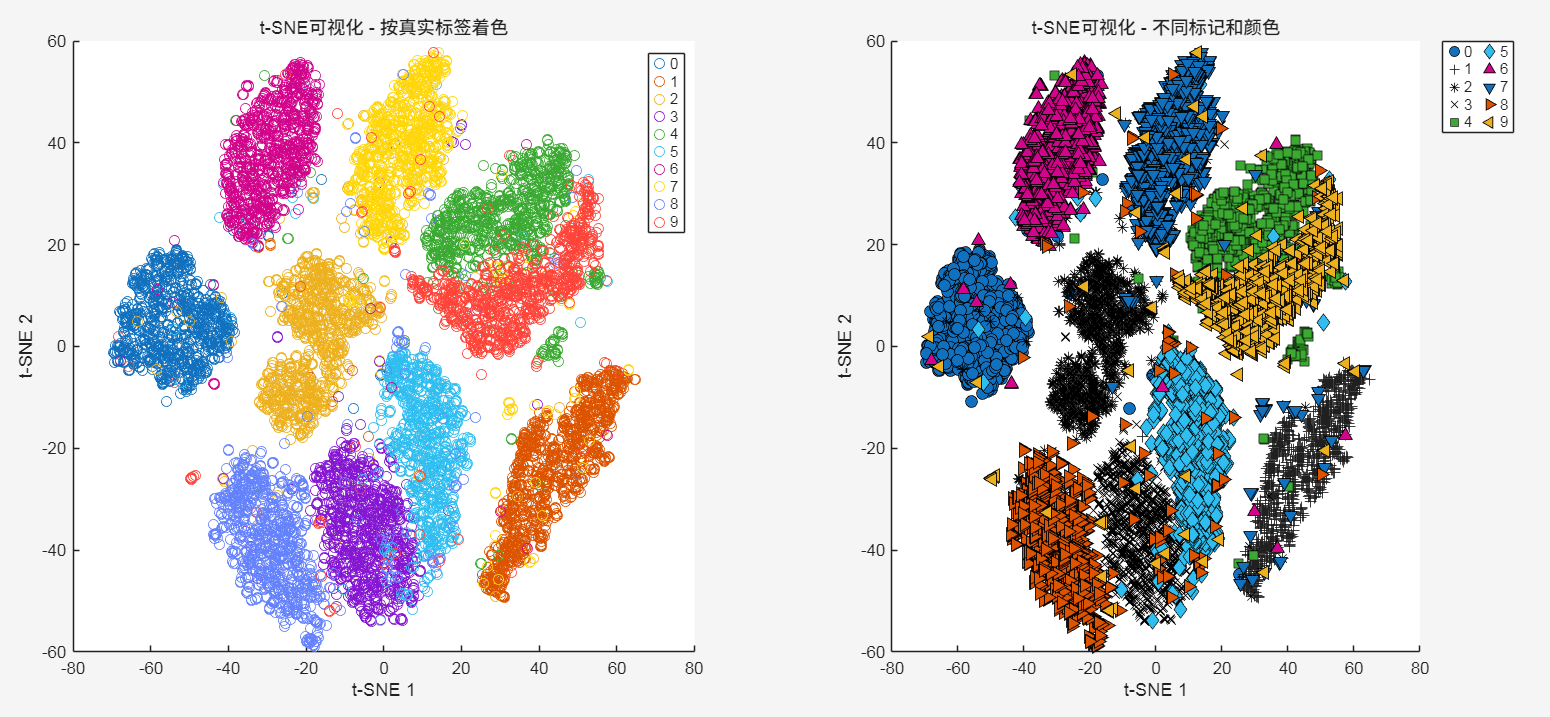
function marker = get\_marker(index)

markers = {'o', '+', '\*', 'x', 's', 'd', '^', 'v', '>', '<'};

marker = markers{mod(index-1, length(markers)) + 1};

end

第一个图采用颜色区分策略，使用gscatter函数按数字标签自动分配不同颜色，所有样本统一使用圆形标记，并自动生成图例说明；第二个图则采用形状区分策略，手动为每个数字分配独特的标记形状和颜色。



**（3）SVM分类**

%% 4. 任务3: SVM分类 - 详细分析

% 使用完整的训练测试集划分

X\_train = train\_data;

y\_train = train\_labels;

X\_test = test\_data;

y\_test = test\_labels;

fprintf('数据集划分: 训练集%d, 测试集%d\n', size(X\_train,1), size(X\_test,1));

数据集划分: 训练集60000, 测试集10000

直接使用MNIST原始的训练集和测试集划分，训练集包含60000个手写数字样本，测试集包含10000个样本。这种划分方式保持了数据集的原始分布，便于与其他研究结果进行对比。

% 4.1 尝试不同的SVM核函数

fprintf('\n比较不同SVM核函数的性能...\n');

kernels = {'linear', 'rbf', 'polynomial'};

accuracies = zeros(length(kernels), 1);

training\_times = zeros(length(kernels), 1);

for i = 1:length(kernels)

kernel = kernels{i};

fprintf('训练 %s 核SVM...\n', kernel);

tic;

switch kernel

case 'linear'

template = templateSVM('KernelFunction', 'linear', 'Standardize', true);

case 'rbf'

template = templateSVM('KernelFunction', 'rbf', 'Standardize', true, 'KernelScale', 'auto');

case 'polynomial'

template = templateSVM('KernelFunction', 'polynomial', 'Standardize', true, 'PolynomialOrder', 3);

end

model = fitcecoc(X\_train, y\_train, 'Learners', template, 'Coding', 'onevsone');

training\_time = toc;

training\_times(i) = training\_time;

y\_pred = predict(model, X\_test);

accuracy = sum(y\_pred == y\_test) / length(y\_test);

accuracies(i) = accuracy;

fprintf('%s 核SVM - 准确率: %.2f%%, 训练时间: %.2f秒\n', ...

kernel, accuracy \* 100, training\_time);

end

比较不同SVM核函数的性能...  
训练 linear 核SVM...

linear 核SVM - 准确率: 93.63%, 训练时间: 870.70秒  
训练 rbf 核SVM...

rbf 核SVM - 准确率: 95.93%, 训练时间: 227.32秒  
训练 polynomial 核SVM...

polynomial 核SVM - 准确率: 11.04%, 训练时间: 4194.37秒

比较三种SVM核函数在MNIST数据集上的表现：

线性核函数达到93.63%的准确率但训练时间较长；

RBF核函数表现最佳，准确率95.93%且训练效率最高；

多项式核函数表现异常，准确率仅11.04%（可能由于多项式阶数设置不当或数据标准化问题）

% 可视化不同核函数的性能比较

figure('Position', [100, 100, 1200, 500]);

subplot(1, 2, 1);

bar(accuracies \* 100, 'FaceColor', [0.3, 0.5, 0.9]);

set(gca, 'XTickLabel', kernels);

ylabel('准确率 (%)');

title('不同SVM核函数性能比较');

grid on;

subplot(1, 2, 2);

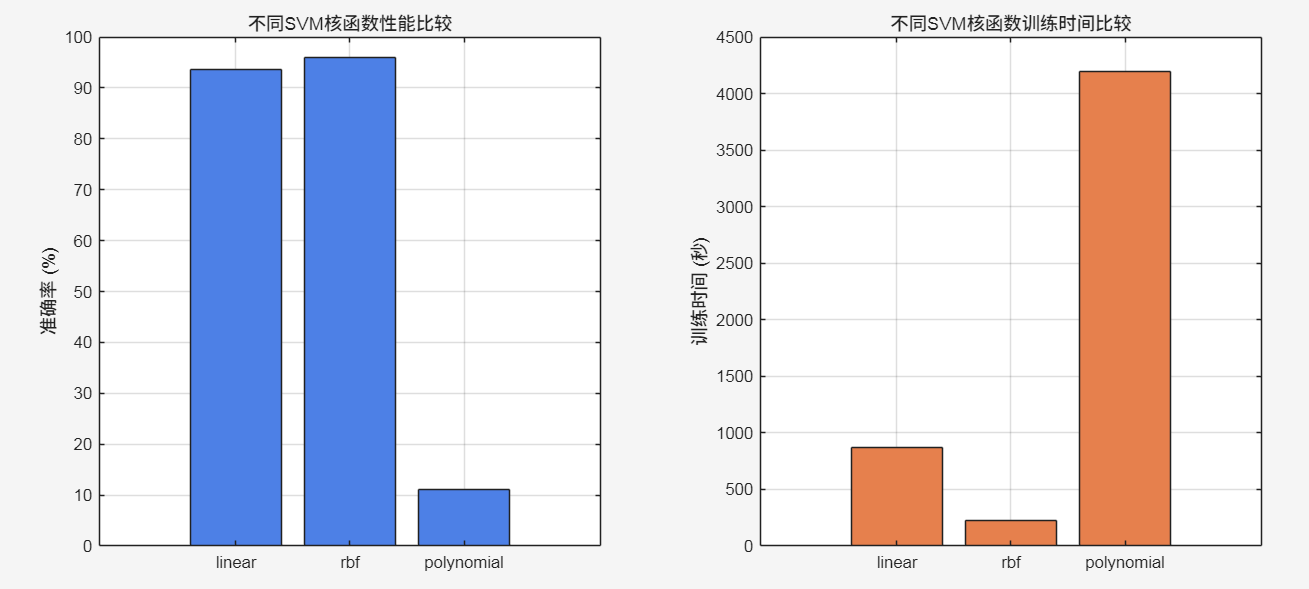
bar(training\_times, 'FaceColor', [0.9, 0.5, 0.3]);

set(gca, 'XTickLabel', kernels);

ylabel('训练时间 (秒)');

title('不同SVM核函数训练时间比较');

grid on;



生成的两张图显示：左侧准确率比较图中RBF核高于其他两种核函数；右侧训练时间图中多项式核显著长于其他方法，RBF核训练时间最短。（所以显然应该选择RBF核）

% 4.2 使用最佳核函数进行最终训练

fprintf('\n使用最佳核函数进行最终训练...\n');

[best\_accuracy, best\_idx] = max(accuracies);

best\_kernel = kernels{best\_idx};

switch best\_kernel

case 'linear'

final\_template = templateSVM('KernelFunction', 'linear', 'Standardize', true);

case 'rbf'

final\_template = templateSVM('KernelFunction', 'rbf', 'Standardize', true, 'KernelScale', 'auto');

case 'polynomial'

final\_template = templateSVM('KernelFunction', 'polynomial', 'Standardize', true, 'PolynomialOrder', 3);

end

final\_model = fitcecoc(X\_train, y\_train, 'Learners', final\_template, 'Coding', 'onevsone');

y\_test\_pred = predict(final\_model, X\_test);

final\_accuracy = sum(y\_test\_pred == y\_test) / length(y\_test);

基于性能比较结果，自动选择RBF核函数作为最佳模型并重新训练！

% 4.3 显示详细分类报告

fprintf('\n=== 详细分类报告 ===\n');

class\_accuracy = zeros(10, 1);

for digit = 0:9

mask = (y\_test == digit);

if sum(mask) > 0

digit\_accuracy = sum(y\_test\_pred(mask) == digit) / sum(mask);

class\_accuracy(digit+1) = digit\_accuracy;

fprintf('数字 %d 的准确率: %.2f%% (%d/%d)\n', ...

digit, digit\_accuracy \* 100, sum(y\_test\_pred(mask) == digit), sum(mask));

end

end

=== 详细分类报告 ===  
数字 0 的准确率: 97.45% (955/980)  
数字 1 的准确率: 97.97% (1112/1135)  
数字 2 的准确率: 96.61% (997/1032)  
数字 3 的准确率: 96.04% (970/1010)  
数字 4 的准确率: 96.33% (946/982)  
数字 5 的准确率: 95.29% (850/892)  
数字 6 的准确率: 96.35% (923/958)  
数字 7 的准确率: 95.43% (981/1028)  
数字 8 的准确率: 94.25% (918/974)  
数字 9 的准确率: 93.66% (945/1009)

详细分类报告显示模型对各个数字的识别精度存在差异。数字1和0的识别率最高（超过97%），表明这些数字的特征最为鲜明；数字8和9的识别率相对较低（约94%），可能由于这些数字的形状容易与其他数字混淆，如8与3、5、6的相似性，9与4、7的相似性。

不管怎样，这比kmeans的结果要好太多了。

% 显示混淆矩阵和各类别准确率

figure('Position', [100, 100, 1400, 600]);

subplot(1, 2, 1);

confusionchart(y\_test, y\_test\_pred);

title(sprintf('SVM分类混淆矩阵 (%s核, 准确率: %.2f%%)', best\_kernel, final\_accuracy \* 100));

subplot(1, 2, 2);

bar(0:9, class\_accuracy \* 100, 'FaceColor', [0.2, 0.7, 0.4]);

xlabel('数字类别');

ylabel('准确率 (%)');

title('各数字类别分类准确率');

ylim([0, 100]);

grid on;

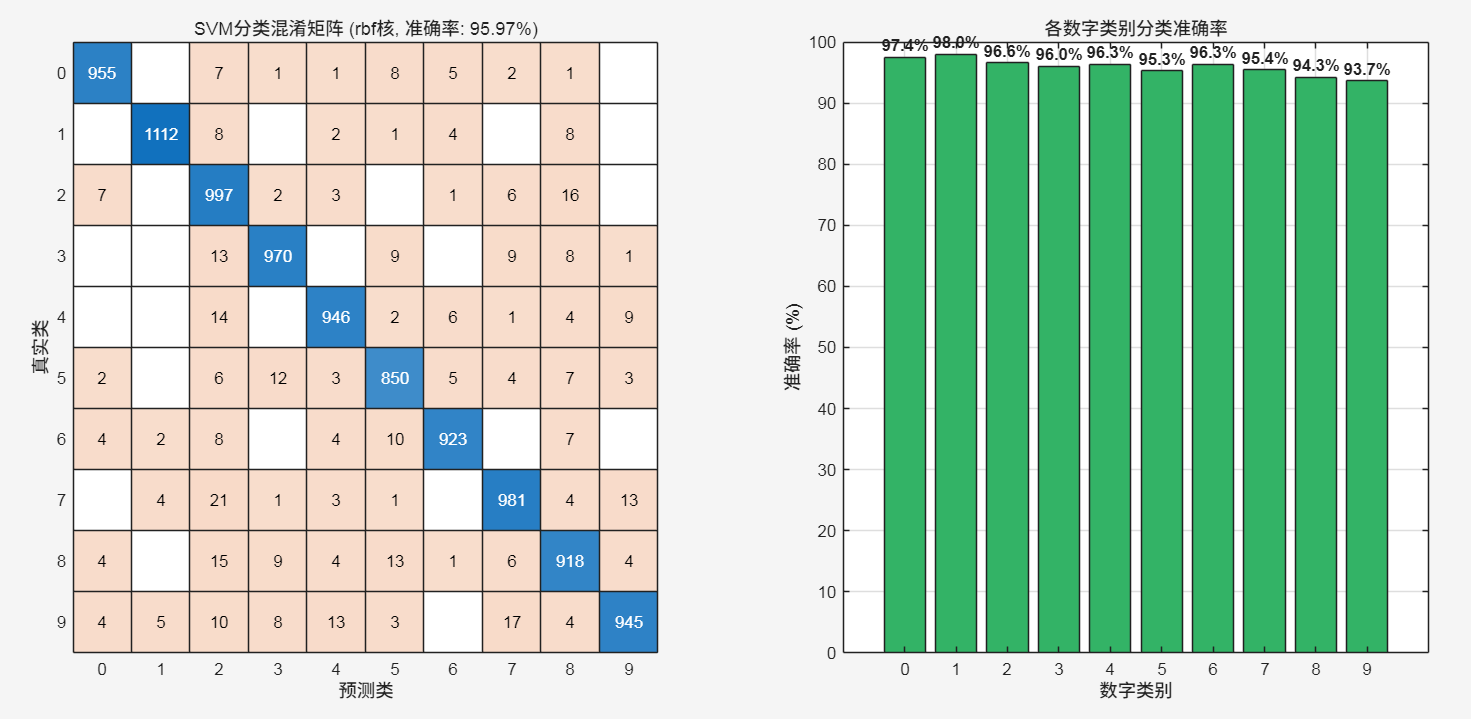
% 添加数值标签

for i = 1:10

text(i-1, class\_accuracy(i)\*100 + 2, sprintf('%.1f%%', class\_accuracy(i)\*100), ...

'HorizontalAlignment', 'center', 'FontWeight', 'bold');

end



左侧混淆矩阵显示主要错误集中在形状相似的数字对之间，如2-7、2-8等；右侧柱状图清晰展示各数字类别的分类精度，柱顶的数值标签便于精确读取分类准确率百分比（题目要求）。

本实验为机器学习方法选择提供了实用参考：在计算资源有限且需要快速部署的场景下，SVM+RBF核函数组合在MNIST这类图像分类任务中能够达到接近深度学习的性能，而计算成本远低于深度神经网络。同时，实验也验证了从简单方法入手、逐步复杂的分析流程在实际问题解决中的有效性。