摘要

本研究使用 ORL 数据库中的人脸图像进行了人脸识别的实验研究。

针对二分类和多分类问题，设计了基于支持向量机（SVM）的模型。

在二分类问题中，选择了是否戴眼镜作为类标签进行预处理和分类。通过系统地调整 SVM 的核函数和超参数，评估了不同模型在测试数据集上的精度、查准率、查全率和 F1 值。

在多分类问题中，同样进行了类似的实验，并对不同模型的测试精度进行了比较。实验结果显示，适当调整模型参数可以显著提升人脸识别系统的性能和准确度，为进一步研究和应用提供了有益的参考和启示。

KEY WORDS: ORL，人脸识别，SVM，OVO,OVR

第一章 SVM 模型

1.1 历史

原始SVM算法是由苏联数学家弗拉基米尔•瓦普尼克和亚历克塞•泽范兰杰斯于 1963 年发明的。1992 年，伯恩哈德 • E • 博瑟（Bernhard E. Boser）、伊莎贝尔 • M • 盖昂（Isabelle M. Guyon）和瓦普尼克提出了一种通过将核技巧应用于最大间隔超平面来创建非线性分类器的方法。当前标准的前身（软间隔）由科琳娜 • 科特斯和瓦普尼克于 1993 年提出，并于 1995 年发表。

1.2 简介

在机器学习中，支持向量机（英语：support vector machine，常简称为 SVM，又名支持向量网络）是在分类与回归分析中分析数据的监督式学习模型与相关的学习算法。给定一组训练实例，每个训练实例被标记为属于两个类别中的一个或另一个，SVM 训练算法建立一个将新的实例分配给两个类别之一的模型，使其成为非概率二元线性分类器。SVM 模型是将实例表示为空间中的点，这样映射就使得单独类别的实例被尽可能宽的明显的间隔分开。然后，将新的实例映射到同一空间，并基于它们落在间隔的哪一侧来预测所属类别。

除了进行线性分类之外，SVM 还可以使用所谓的核技巧有效地进行非线性分类，将其输入隐式映射到高维特征空间中。

当数据未被标记时，不能进行监督式学习，需要用非监督式学习，它会尝试找出数据到簇的自然聚类，并将新数据映射到这些已形成的簇。将支持向量机改进的聚类算法被称为支持向量聚类，当数据未被标记或者仅一些数据被标记时，支持向量聚类经常在工业应用中用作分类步骤的预处理。

1.3 思想

将数据进行分类是机器学习中的一项常见任务。假设某些给定的数据点各自属于两个类之一，而目标是确定新数据点将在哪个类中。对于支持向量机来说，数据点被视为 p 维向量，而我们想知道是否可以用 $(p-1)$ 维超平面来分开这些点。这就是所谓的线性分类器。可能有

许多超平面可以把数据分类。最佳超平面的一个合理选择是以最大间隔把两个类分开的超平面。因此，我们要选择能够让到每边最近的数据点的距离最大化的超平面。如果存在这样的超平面，则称为最大间隔超平面，而其定义的线性分类器被称为最大间隔分类器，或者叫做最佳稳定性感知器。

第二章　二分类问题

2.1　概述

从网络上下载 ORL 数据集的人脸图片，并在 glass.m 创建了 400 张图片的眼镜标签矩阵 labels（0 代表没戴眼镜，1 代表戴眼镜），并转换为 glass.mat。

在 face_glasses_classification.m 中完成是否佩戴眼镜的二分类问题。

2.2　建模过程

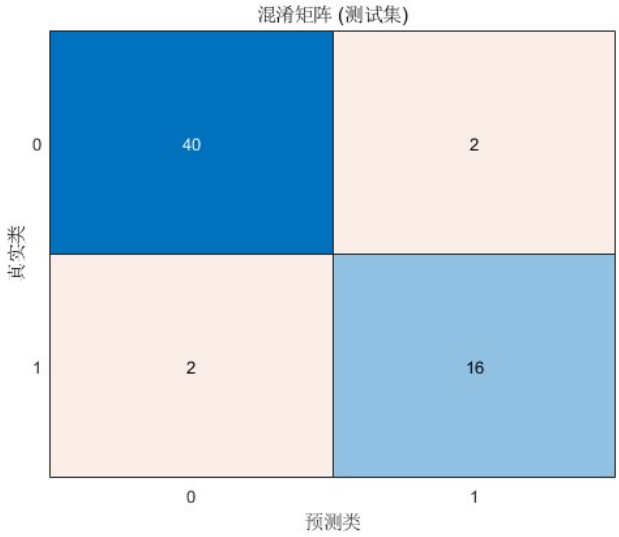
- 加载数据：从两个 MAT 文件中加载数据，fea 是人脸数据特征矩阵，labels 是眼镜标签。
- 数据划分：使用 dividerand 函数将数据划分为训练集、验证集和测试集。
- 超参数：trainRatio、valRatio 和 testRatio 是训练集、验证集和测试集的划分比例。并设置随机种子方便比较核函数的改动对模型的影响。
- 模型训练：fitcsvm 函数用于训练支持向量机模型，根据 trainData 和 trainLabels 学习数据模式。
- 模型评估：使用验证集评估模型在未见过的数据上的表现，计算准确率来评估模型的泛化能力；使用测试集评估模型的最终性能，计算准确率来衡量模型在真实场景中的表现。
- 混淆矩阵：使用 confusionmat 函数计算测试集上的混淆矩阵，显示模型在各个类别上的预测结果。
- 计算查准率、查全率和 F1 值

2.3　实验结果

数据集的划分如下
训练集样本数: 280
验证集样本数: 60
测试集样本数: 60

2.3.1 线性核

验证集准确率: 0.98333
测试集准确率: 0.93333



类别	查全率	查准率	f1
0	0.95238	0.95238	0.95238
1	0.88889	0.88889	0.88889

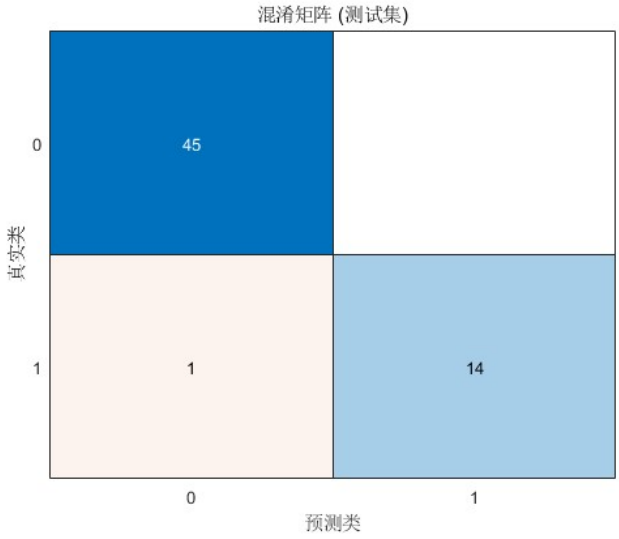
表 2-1 查准率、查全率和 F1 值表

2.3.2 高斯核

验证集准确率: 0.98333
测试集准确率: 0.98333

类别	查全率	查准率	f1
0	0.97826	1	0.98901
1	1	0.93333	0.96552

表 2-2 查准率、查全率和 F1 值表



2.3.3 结论

通过对比可以发现，在其它条件不改变的情况下，在此二分类任务上，高斯核相比线性核有更好的表现。在测试集上的精度高于线性核。相较于线性核，使用高斯核能够获得更高的查全率、查准率和 F1 值。

第三章　多分类问题

3.1　概述

观察人脸图集和数据集可以发现，数据集包含 40 个人的人脸，每个人有 10 张不同的人脸照。依旧利用 SVM 实现这个 N 分类问题（N=40）。并且利用 PCA 方法，防止模型过拟合或者不稳定的问题，尽可能保留原始数据中的信息，以提高模型的效率和性能。

在 face_recognize.m 中完成了这个 N 分类问题。

3.2　建模过程

- 加载数据：从 MAT 文件中加载数据，fea 是人脸数据特征矩阵，gnd 是类别标签。
- 设置随机数种子，以确保结果的可重现性，并用于对比核函数改变后模型的泛化能力。
- 使用循环遍历每个类别，随机打乱每个类别的样本索引，然后将样本划分为训练集、验证集和测试集。
- 使用 pca 函数对训练集数据进行主成分分析。
- 使用 fitcecoc 函数训练多分类的 SVM。在注释提到，可以根据实验需要选择不同的核函数（如线性核、高斯核等）来训练模型。
- 使用训练好的模型对测试集进行预测，并计算测试集的准确率。同时计算并展示混淆矩阵，以评估模型在各个类别上的分类性能。

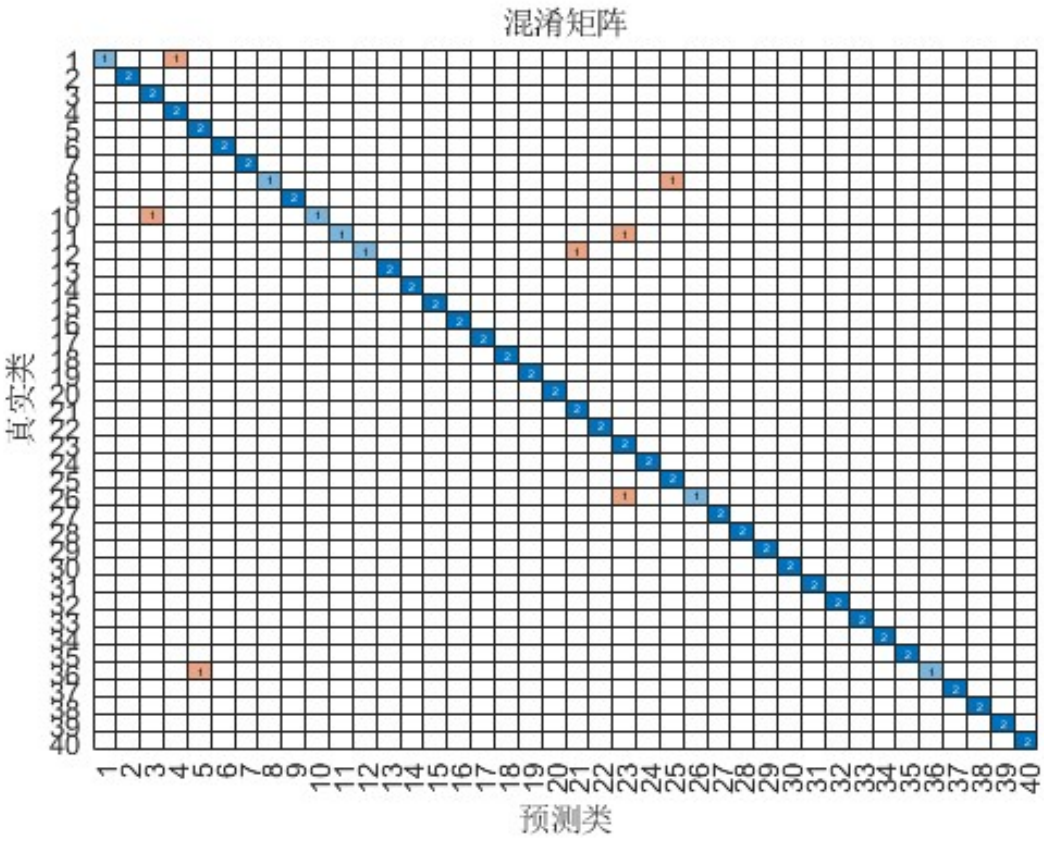
3.3　实验结果

数据划分利用分层取样。每个类 10 张中 6 张用于训练，2 张用于验证，2 张用于测试。

3.3.1　线性核

验证集准确率: 88.75%
测试集准确率: 91.25%

对于多分类的混淆矩阵，第 i 行 j 列代表将第 i 类预测为第 j 类。可以发现对角线上的代表预测正确。



3.3.2 高斯核

网上查阅发现在使用线性核不需要显式指定多分类策略是因为 MATLAB 默认会根据情况选择合适的多类别分类策略。高斯核则需要显式指定多类别分类的策略和相关参数。在此对高斯核利用两种多分类策略分别训练模型。

OVO：

验证集准确率: 96.25%

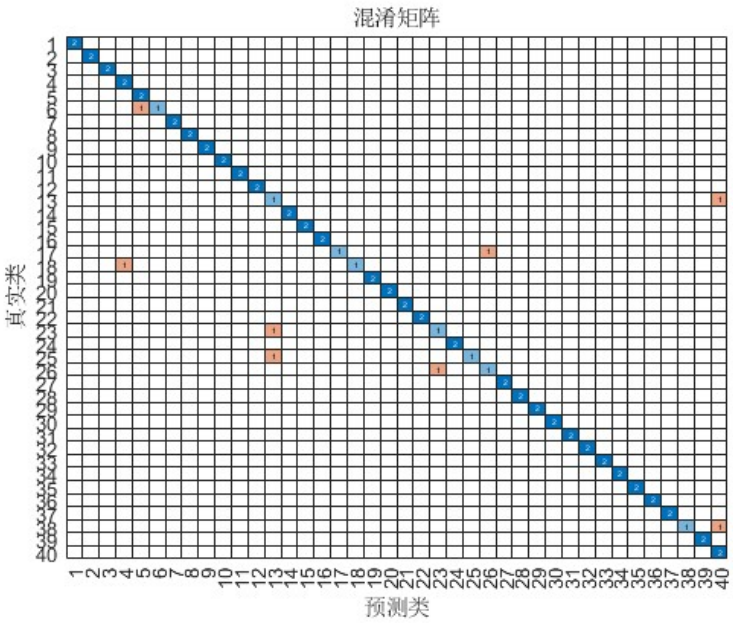
测试集准确率: 90.00%

OVR：

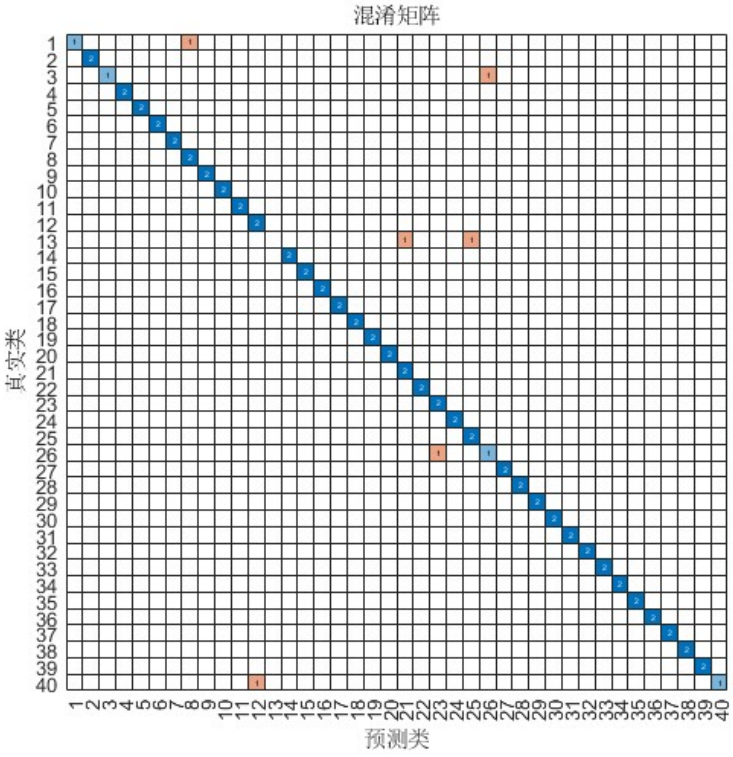
验证集准确率: 98.75%

测试集准确率: 92.50%

OVO 混淆矩阵:



OVR 混淆矩阵：



3.3.3 结论

对比精确率可以发现线性核的表现和高斯核采用 OVO 策略相当，高斯核采用 OVR 策略表现最优。

附录

Listing 1 glass.m

```
    % 类别由人工统计出来
labels = zeros(400, 1);

% 第二组全部
labels(11:20) = 1;

% 第四组 1 2 3 8 9
labels(31:33) = 1;
labels(38:39) = 1;

% 第六组全部
labels(51:60) = 1;

% 第七组 4 5 10
labels(64:65) = 1;
labels(70) = 1;

% 第十三组 1 2 5 6 7 8 9 10
labels(121:122) = 1;
labels(125:130) = 1;

% 第十四组全部
labels(131:140) = 1;

% 第十七组 1 2 5 6 7 8 9 10
labels(161:162) = 1;
labels(165:170) = 1;

% 第十九组 1 2 3 6 10
labels(181:183) = 1;
labels(186) = 1;
labels(190) = 1;

% 第二十组 1 2 3 5 7 8 9 10
labels(191:193) = 1;
labels(195) = 1;
labels(197:200) = 1;
```

```
% 第二十七组全部
labels(261:270) = 1;

% 第二十八组全部
labels(271:280) = 1;

% 第三十一组全部
labels(301:310) = 1;

% 第三十四组全部
labels(331:340) = 1;

% 第三十六组 9 10
labels(359:360) = 1;

% 第三十七组全部
labels(361:370) = 1;

save('glass.mat', 'labels');
```

Listing 2 face_glasses_classification.m

```
    clear; clc;close all;
load('glass.mat'); % 存放眼镜labels标签
load('ORL_32x32.mat', 'fea', 'gnd');

x = fea;
y = labels;

% 划分比例
trainRatio = 0.7;
valRatio = 0.15;
testRatio = 0.15;

% 设置随机种子方便比较超参数改动对模型的影响
rng(1); % 设置随机种子

% 划分数据
[trainInd, valInd, testInd] = dividerand(400,
  trainRatio, valRatio, testRatio);

% 划分输入数据和标签
trainData = x(trainInd, :);
```

```
valData = x(valInd, :);
testData = x(testInd, :);

trainLabels = y(trainInd);
valLabels = y(valInd);
testLabels = y(testInd);

% 输出样本数量
numTrainSamples = length(trainInd);
numValSamples = length(valInd);
numTestSamples = length(testInd);

disp(['训练集样本数: ', num2str(numTrainSamples)]);
disp(['验证集样本数: ', num2str(numValSamples)]);
disp(['测试集样本数: ', num2str(numTestSamples)]);

%% 对比线性核高斯核效果, 高斯核更好
% 训练SVM模型
% 使用线性核函数
%SVMModel = fitcsvm(trainData, trainLabels);
% 使用 高斯核函数
SVMModel = fitcsvm(trainData, trainLabels,
  'KernelFunction', 'rbf', 'KernelScale', 'auto');


% 验证模型
valPredictions = predict(SVMModel, valData);
valAccuracy = sum(valPredictions == valLabels) /
  numel(valLabels);
disp(['验证集准确率: ', num2str(valAccuracy)]);

% 测试模型
testPredictions = predict(SVMModel, testData);
testAccuracy = sum(testPredictions == testLabels) /
  numel(testLabels);
disp(['测试集准确率: ', num2str(testAccuracy)]);

% 计算混淆矩阵
testConfMat = confusionmat(testLabels,
  testPredictions);
disp('混淆矩阵 (测试集):');
disp(testConfMat);
```

```
% 计算查准率、查全率和F1值
numClasses = length(unique(testLabels));
precision = zeros(numClasses, 1);
recall = zeros(numClasses, 1);
f1 = zeros(numClasses, 1);

for i = 1:numClasses
    tp = testConfMat(i,i);
    fp = sum(testConfMat(:,i)) - tp;
    fn = sum(testConfMat(i,:)) - tp;

    precision(i) = tp / (tp + fp);
    recall(i) = tp / (tp + fn);
    f1(i) = 2 * (precision(i) * recall(i)) /
      (precision(i) + recall(i));
end

% 输出查准率、查全率和F1值表格
classLabels = unique(testLabels);
metricsTable = table(classLabels, precision,
  recall, f1);
disp('查准率、查全率和F1值表:');
disp(metricsTable);

% 可视化混淆矩阵
figure;
confusionchart(testLabels, testPredictions);
title('混淆矩阵 (测试集)');
```

Listing 3 face_recognize.m

```
clear;clc;close all;
load('ORL_32x32.mat', 'fea', 'gnd');
X = fea;
y = gnd;

% 设置随机数种子以便复现结果
rng(42);

% 初始化
num_samples = size(X, 1);
num_classes = 40; % ORL数据集有40个人
samples_per_class = 10; % 每个类别的样本数
train_samples_per_class = 6; % 每个类别训练样本数
```

```
val_samples_per_class = 2; % 每个类别验证样本数
test_samples_per_class = samples_per_class -
  train_samples_per_class - val_samples_per_class;
  % 每个类别测试样本数

% 初始化索引
train_idx = [];
val_idx = [];
test_idx = [];

for i = 1:num_classes
    % 找到每个类别的样本索引
    class_idx = find(y == i);

    % 随机打乱每个类别的样本索引
    class_idx =
      class_idx(randperm(length(class_idx)));

    % 将每个类别的样本分为训练集、验证集和测试集
    train_idx = [train_idx;
      class_idx(1:train_samples_per_class)];
    val_idx = [val_idx;
      class_idx(train_samples_per_class +
      1:train_samples_per_class +
      val_samples_per_class)];
    test_idx = [test_idx;
      class_idx(train_samples_per_class +
      val_samples_per_class + 1:end)];
end

% 划分训练集、验证集和测试集
X_train = X(train_idx, :);
y_train = y(train_idx);
X_val = X(val_idx, :);
y_val = y(val_idx);
X_test = X(test_idx, :);
y_test = y(test_idx);

% 进行PCA
[coeff, X_train_pca, ~, ~, explained] =
  pca(X_train);
```

```
%
   选择保留的主成分数目（例如，解释方差达到95%的主成分）
num_components = find(cumsum(explained) >= 95, 1);

% 将训练集、验证集和测试集投影到PCA空间
X_train_pca = X_train * coeff(:, 1:num_components);
X_val_pca = X_val * coeff(:, 1:num_components);
X_test_pca = X_test * coeff(:, 1:num_components);

% 训练多分类SVM分类器
%SVMModel = fitcecoc(X_train_pca, y_train);

% 训练多分类SVM分类器
t = templateSVM('KernelFunction', 'rbf',
   'BoxConstraint', 1, 'KernelScale', 'auto');
% one vs one
% SVMModel = fitcecoc(X_train_pca, y_train,
   'Learners', t, 'Coding', 'onevsone');
% one vs others
SVMModel = fitcecoc(X_train_pca, y_train,
   'Learners', t, 'Coding', 'onevsall');


% 使用验证集进行预测以调整模型参数
y_val_pred = predict(SVMModel, X_val_pca);

% 计算验证集准确率
val_accuracy = sum(y_val_pred == y_val) /
   length(y_val);
fprintf('验证集准确率: %.2f%%\n', val_accuracy *
   100);

% 使用训练好的SVM模型进行测试集预测
y_test_pred = predict(SVMModel, X_test_pca);

% 计算测试集准确率
test_accuracy = sum(y_test_pred == y_test) /
   length(y_test);
fprintf('测试集准确率: %.2f%%\n', test_accuracy *
   100);

% 计算混淆矩阵
confMat = confusionmat(y_test, y_test_pred);
```

```
% 可视化混淆矩阵
figure;
confusionchart(confMat);
title('混淆矩阵');
```