数据挖掘第二次报告

1. 任务描述：

本次作业是基于 MNIST 数据集加上自己添加的数据，进行一次简单的手写数字识别的任务，要求有很多，比如：

（1）数据采集与处理部分要求：收集自己用手机拍照 0 到 9 的手写数字图片，将其处理为 mnist 格式（28x28），添加进训练数据集；

（2）模型训练部分要求：使用上述数据集进行模型训练，得到手写数字识别的 SVM 模型，并要求选用不同的核函数和松弛变量惩罚参数，以选择准确率最高的模型作为最终模型；

（3）模型交叉验证部分要求：使用混淆矩阵，提升交叉验证的效果，要求在可视化部分展示出添加自己拍照的手写数字前后，（即扩充 mnist 数据集前后）的混淆矩阵，以及 5 折交叉验证效果；

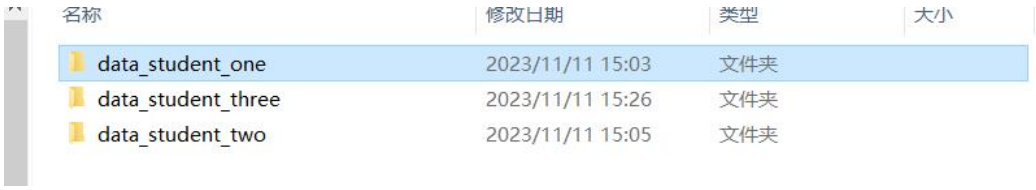
（4）结果可视化验证部分要求：使用新的手写数字图片检验模型效果，并进行可视化展示。

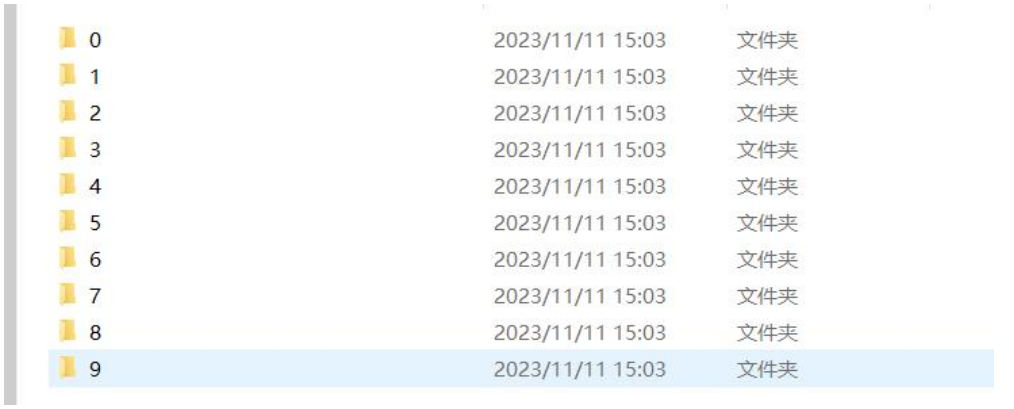
2. 数据集来源：

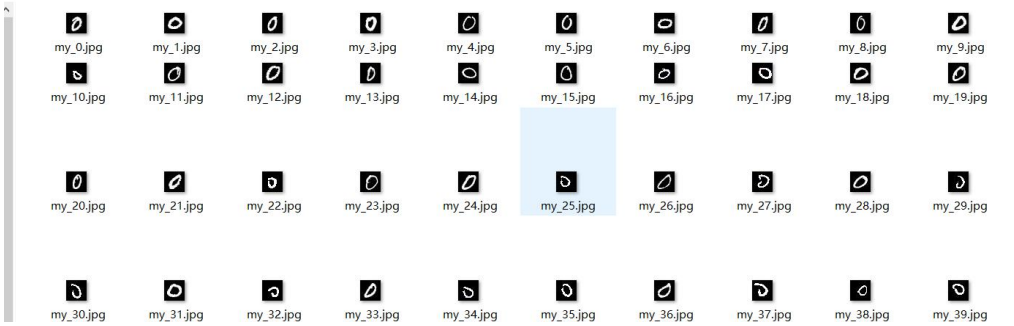
数据集的说明，我们采用的数据集主要来自以下两个途径：
（1）一是在官网上下载的 MNIST 数据集，这是通过下面代码获得的：

```
import numpy as np
from PIL import Image
import tensorflow as tf
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt


def data_process():

    print('导入MNIST官方数据集.....................')
    # 获取脚本文件的目录
    script_dir = os.path.dirname(os.path.realpath(__file__))
    # 将工作目录切换到脚本文件的目录
    os.chdir(script_dir)

    # 加载MNIST数据集
    mnist = tf.keras.datasets.mnist
    (train_images, train_labels), (test_images, test_labels) = mnist.load_data()

```

（2）二是我们小组合作获得的数据集：







3. 数据集：

（1）数据集的处理方法

在 MNIST 官网上获得的数据集比较规范，并不需要太多的处理，所以我们主要做的处理是将自己制取的数据集进行一个处理，主要的过程就是如下：

```
# 4. 处理每张图片
for image_file in image_files:
    # 打开图片
    img = Image.open(image_file)

    # 重新调整大小为28x28像素
    img = img.resize((28, 28))

    # 将图片转为NumPy数组
    img_array = np.array(img)

    # 设置阈值（这里使用128作为阈值，你可以根据需要进行调整）
    threshold = 128

    # 对图片进行阈值化处理
    img_array[img_array > threshold] = 255
    img_array[img_array <= threshold] = 0

    # 对图片进行反相处理
    img_array = 255 - img_array

    # 将处理后的图片添加到列表中
    processed_images.append(Image.fromarray(img_array))
```

首先将图片打开，并调整到对应的大小，之后我们将其转为 np 数组然后进行一个简单的二值化处理，这里我们选择 256 的中间值作为二值化的阈值，于是就产生了二值化的黑白图片。

（2）数据集整理和划分：

首先我们导入官网的 MNIST 数据集

```
print('导入MNIST官方数据集.....................')
# 获取脚本文件的目录
script_dir = os.path.dirname(os.path.realpath(__file__))
# 将工作目录切换到脚本文件的目录
os.chdir(script_dir)

# 加载MNIST数据集
mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images/255.0
test_images = test_images/255.0

train_images, val_images, train_labels, val_labels = train_test_split(train_images, train_labels, test_size=0.2, random_state=42)

# 打印数据集的形状
print("训练集图像形状:", train_images.shape)
print("训练集标签形状:", train_labels.shape)
print("验证集图像形状:", val_images.shape)
print("验证集标签形状:", val_labels.shape)
print("测试集图像形状:", test_images.shape)
print("测试集标签形状:", test_labels.shape)
print()
```

上述的结果为：

```
导入MNIST官方数据集.....................
训练集图像形状: (48000, 28, 28)
训练集标签形状: (48000,)
验证集图像形状: (12000, 28, 28)
验证集标签形状: (12000,)
测试集图像形状: (10000, 28, 28)
测试集标签形状: (10000,)
```

之后我们要导入自己的数据集：

这里我们为了观察最终的结果，所以决定拿 MNIST 数据集，学生 1 和学生 3 的数据作为训练集和验证集，然后将学生 2 的数据作为最后的测试集，以观察自己导入的数据对最终结果的影响。

因此我们要首先处理学生 1 的数据：

```
print('导入自己的数据集.....................')
# 处理学生一的数据
in_folder = "data/data_student_one/"
student_data = [[] for _ in range(10)]

# 遍历0-9的文件夹
```

```
    # 遍历0-9的文件夹
    for digit in range(10):
        digit_folder = os.path.join(in_folder, str(digit))

        # 遍历文件夹中的文件
        for filename in os.listdir(digit_folder):
            # 构建完整的文件路径
            file_path = os.path.join(digit_folder, filename)

            # 打开图像
            image = Image.open(file_path)
            image = image.resize((28, 28))
            image = image.convert('L')
            image_array = np.array(image)
            image_array = image_array / 255.0

            # 将图像和标签添加到列表中
            student_data[digit].append(image_array)
```

然后我们处理学生 3 的数据：

```
    # 处理学生三的数据
    in_folder = "data/data_student_three/"
    number = [2, 3, 5, 7]

    # 遍历0-9的文件夹
    for digit in number:
        digit_folder = os.path.join(in_folder, str(digit))

        # 遍历文件夹中的文件
        for filename in os.listdir(digit_folder):
            # 构建完整的文件路径
            file_path = os.path.join(digit_folder, filename)

            # 打开图像
            image = Image.open(file_path)
            image = image.resize((28, 28))
            image = image.convert('L')
            image_array = np.array(image)
            image_array = image_array / 255.0

            # 将图像和标签添加到列表中
            student_data[digit].append(image_array)
```

然后我们需要整合上面两个学生的数据与 MNIST 数据相结合，构成最终的训练集和测试集：

```
student_data = [np.array(queue) for queue in student_data]
student_label = [[] for _ in range(10)]
```

```
#展示处理的同学的数据
for i in range(10):
    for j in range(student_data[i].shape[0]):
        student_label[i].append(i)
    student_label[i] = np.array(student_label[i])


for i in range(10):
    m = len(student_data[i])
    n = int(m*0.7)
    p = int(m*0.2)
    q = m-n-p
    array_data = student_data[i][:n]
    array_label = student_label[i][:n]
    train_images = np.concatenate((train_images, array_data))
    train_labels = np.concatenate((train_labels, array_label))


    array_data = student_data[i][n:n+p]
    array_label = student_label[i][n:n+p]
    val_images = np.concatenate((val_images, array_data))
    val_labels = np.concatenate((val_labels, array_label))
```

整合之后，我们需要再次打乱上面的训练集和测试集：

```
random_state = 42
np.random.seed(random_state)

# 生成打乱的索引
shuffle_index_1 = np.random.permutation(len(train_images))
# 根据打乱的索引重新排序数据和标签
train_images = train_images[shuffle_index_1]
train_labels = train_labels[shuffle_index_1]


# 生成打乱的索引
shuffle_index_2 = np.random.permutation(len(val_images))
# 根据打乱的索引重新排序数据和标签
val_images = val_images[shuffle_index_2]
val_labels = val_labels[shuffle_index_2]
```

最后我们观察自己处理的结果：

```
导入自己的数据集.....................
训练集图像形状: (52125, 28, 28)
训练集标签形状: (52125,)
验证集图像形状: (13177, 28, 28)
验证集标签形状: (13177,)
测试集图像形状: (10596, 28, 28)
测试集标签形状: (10596,)
```

4. 模型与训练：

（1）模型结构设计：

本题的模型比较简单，就是一个简单的 SVM 模型，然后要求选用不同的核函数和松弛变量惩罚参数，以选择准确率最高的模型作为最终模型：

我们首先定义参数网格，然后一一进行训练，寻找准确率最高的模型：

```
# 定义参数网格
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf', 'poly'],
    'gamma': ['scale', 'auto']
}

# 创建SVM模型
model = svm.SVC()

# 使用GridSearchCV进行参数调优和交叉验证
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, scoring='accuracy', cv=5)
```

（2）训练过程：

我们将具体的数据和标签导入，开始进行训练，由于训练时间过长，为了简便的看一下效果，我们就先使用 16 分之一的部分进行训练：

```
train_images = train_images[:len(train_images)//16]
train_labels = train_labels[:len(train_labels)//16]


grid_search.fit(train_images.reshape(train_images.shape[0], -1), train_labels)
```

（3）寻找最优方法：

我们需要选用不同的核函数和松弛变量惩罚参数，因此需要多次训练，并找出准确率最高的一个模型：

```
# 获取最佳模型
best_model = grid_search.best_estimator_

# 保存最佳模型
torch.save(best_model, os.path.join(checkpoints, 'best_model_16.pth'))

# 使用最佳模型进行测试数据的预测
test_predictions = best_model.predict(test_images.reshape(test_images.shape[0], -1))

# 计算混淆矩阵
cm = confusion_matrix(test_labels, test_predictions)
print("混淆矩阵:\n", cm)

file.write("\n混淆矩阵:\n")
file.write(str(cm))


# 使用5折交叉验证评估模型性能
scores = cross_val_score(best_model, train_images.reshape(train_images.shape[0], -1), train_labels, cv=5)
print("交叉验证准确率: ", np.mean(scores))

file.write("\n交叉验证准确率: ")
file.write(str(np.mean(scores)))
```

由于上面的训练时间过长，我们对最初的训练集进行了一个小的划分，进一步观察 不同大小训练集上的最优模型

结果展示：

```
原数据集64分之一
开始执行GridSearchCV...

参数网格大小:18
最佳参数: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
最佳准确率: 0.8906384912519881
混淆矩阵:
[[1000    0    6    5    2   11    7    2    2    0]
 [   2 1155    8    2    0    7    6    1    7    0]
 [  20   54  879   28   18   12    8   15   53    7]
 [   0   77    8  918    0   39    1   15   15    8]
 [   1    7   10    1  921    6   17    4   17   56]
 [   9   69    6   50    4  771    2    6   27   10]
 [  15   11    6    3    2   27  939    1    5    3]
 [   3   77   18    9    7    9    0  954    3   19]
 [   6   10   15   24    7   29   12   10  905   11]
 [  17   11    3   11   38    6    2   26   12  938]]
交叉验证准确率: 0.8906384912519881
```

```
原数据集32分之一
开始执行GridSearchCV...

参数网格大小:18
最佳参数: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
最佳准确率: 0.9047796130250119
混淆矩阵:
[[1009    0    4    2    2    4    8    3    3    0]
 [   0 1170    3    3    0    0    4    3    5    0]
 [  20   66  924    6   11    6    7   17   33    4]
 [   0   58    7  944    0   22    1   16   28    5]
 [   1    7    9    4  949    2   19    2    8   39]
 [  11   56    8   27    8  805    5    5   24    5]
 [  10    9    8    6    2   12  955    2    5    3]
 [   3   58   20    8    6   16    0  958   10   20]
 [   4    5   15   21    9   19   14    8  927    7]
 [  15   11    1   16   27    2    1   12   10  969]]
交叉验证准确率: 0.9047796130250119
```

```
原数据集16分之一
开始执行GridSearchCV...

参数网格大小:18
最佳参数: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
最佳准确率: 0.9229335708160169
混淆矩阵:
[[1013    0    3    2    1    5    5    5    1    0]
 [   0 1170    2    2    0    2    5    6    1    0]
 [  17   56  958    5   10    4    7   12   25    0]
 [   2   58    8  964    1    9    1   24   12    2]
 [   1    1    7    4  989    5    8    1    4   20]
 [   8   52    2   26    4  827    4   17   10    4]
 [  12    7    3    4    4   12  965    0    2    3]
 [   2   67   17    4    8    2    0  987    0   12]
 [   7    8    8   25   11   22   12    8  918   10]
 [  14   10    2   15   32    7    2   10    1  971]]
交叉验证准确率: 0.9229335708160169
```

5. 预测和结果展示：

（1）预测过程：

在完成整体的训练过程，并将训练的模型储存之后，我们就需要进行结果的预测和验证了：

首先我们处理一下学生 2 的数据作为测试集：

```python
def predict(test_images, test_labels, model):

    # 处理学生二的数据
    in_folder = "data/data_student_two/"

    predict_data = [[] for _ in range(10)]

    # 遍历0-9的文件夹
    for digit in range(10):
        digit_folder = os.path.join(in_folder, str(digit))

        # 遍历文件夹中的文件
        for filename in os.listdir(digit_folder):
            # 构建完整的文件路径
            file_path = os.path.join(digit_folder, filename)

            # 打开图像
            image = Image.open(file_path)
            image = image.resize((28, 28))
            image = image.convert('L')
            image_array = np.array(image)
            image_array = image_array / 255.0

            # 将图像和标签添加到列表中
            predict_data[digit].append(image_array)
```

```
    predict_data = [np.array(queue) for queue in predict_data]
    predict_label = [[] for _ in range(10)]

        #展示处理的同学的数据
    for i in range(10):
        for j in range(predict_data[i].shape[0]):
            predict_label[i].append(i)
        predict_label[i] = np.array(predict_label[i])

    test_image = []
    test_label = []

    for i in range(10):
        for j in predict_data[i]:
            test_image.append(j)

    for i in range(10):
        for j in predict_label[i]:
            test_label.append(j)
```

之后我们就开始预测了：

```
    path = 'data\\data_student_two'
    result_number = []
    for i in range(10):
        result_number.append(('{}\\{}\\image_{}.jpg'.format(path, i, i), i))
        print(result_number[i])

    return result_number
```

注意上面的预测要将预测的结果和图片的地址储存起来，后面要进行可视化的。

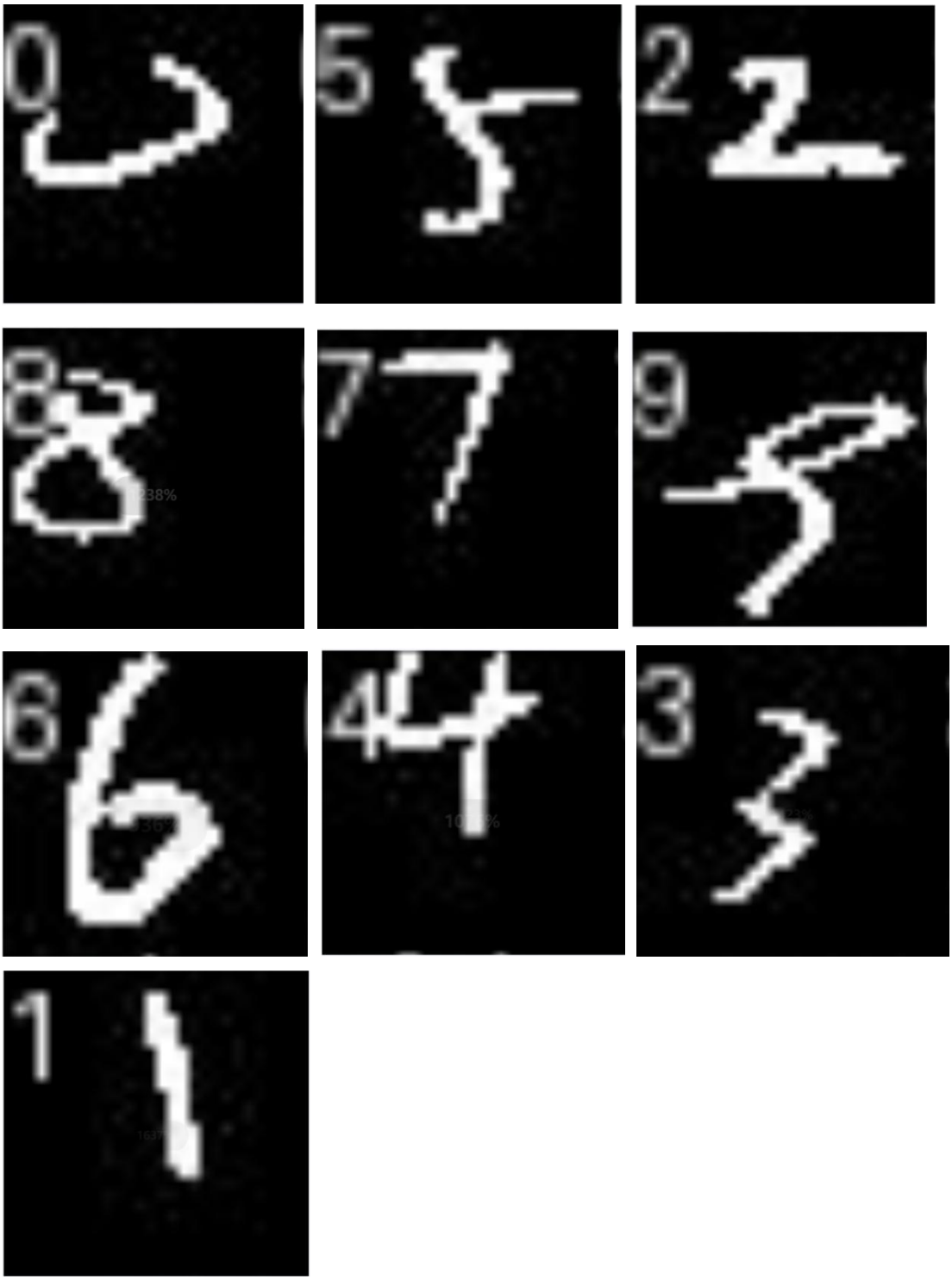
（2）结果可视化：

可视化的过程比较简单，我们将之前储存的预测结果和对应的图片打开并展示出来就可以了：

```
from PIL import Image, ImageDraw, ImageFont
import random

def showresult(predict_num):
    random.shuffle(predict_num)


    for i in predict_num:
        # 打开图片
        image_path = i[0]
        image = Image.open(image_path)

        # 在图片上绘制文本
        draw = ImageDraw.Draw(image)
        text = str(i[1])
        x = 0  # 标注文本的横坐标位置
        y = 0  # 标注文本的纵坐标位置
        draw.text((x, y), text, fill=255)  # 使用默认字体和红色填充文本

        # 显示图片
        image.show()

```

结果展示：

其中左上角是预测的结果。



6. 结论与展望：

（1）结论：

在这个手写数字识别任务中，我们使用了支持向量机（SVM）算法来完成。通过调整不同的核函数和松弛变量惩罚参数，我们选择了准确率最高的模型作为最终模型。

首先，我们对数据进行了预处理，包括对图像进行了标准化处理，并将像素值缩放到 0 和 1 之间。然后，我们将训练集分为训练集和验证集，用于模型的训练和调优。

接下来，我们定义了一个 SVM 模型，并使用 GridSearchCV 进行参数调优和交叉验证。我们通过定义参数网格，包括不同的核函数和松弛变量惩罚参数，来搜索最佳的参数组合。通过交叉验证，我们评估了不同参数组合下模型的性能，并选择了准确率最高的模型。

在训练过程中，我们使用最佳参数组合训练了 SVM 模型，并将其保存为最佳模型。然后，我们使用最佳模型对测试数据进行预测，并计算了混淆矩阵来评估模型的性能。

最后，我们使用 5 折交叉验证对最佳模型进行了评估，并计算了平均准确率作为模型性能的指标。

通过这个任务，我们熟悉了 SVM 算法的应用和调优过程。我们学会了选择合适的核函数和调整松弛变量惩罚参数来提高模型的性能。通过交叉验证和评估指标，我们能够客观地评估模型的性能，并选择最佳模型进行预测。

总的来说，这个手写数字识别任务是一个很好的实践项目，通过使用 SVM 算法，我们获得了令人满意的结果，并学到了在数据挖掘任务中调优模型的技巧和方法。这将为我们进一步探索和应用其他机器学习算法提供宝贵的经验和知识。

（2）展望：

基于手写数字识别任务的成功经验，我们可以进一步探索和应用数据挖掘的方法和技术。以下是一些展望方向：

1. 扩大规模：尝试在更大的数据集上进行实验和训练，以提升模型的泛化能力和鲁棒性。可以探索使用更多的手写数字图像数据集，或者通过数据增强技术生成更多的训练样本。

2. 探索其他网络架构：除了使用 SVM，可以尝试使用其他网络架构，如卷积神经网络（CNN）或循环神经网络（RNN）。这些网络架构在处理具有空间结构或时间序列特征的数据时具有优势，可能进一步提高手写数字识别的性能。

3. 优化性能和效率：通过调整超参数、使用正则化技术、引入预训练模型等方法，优化模型的性能和训练效率。可以尝试不同的核函数和松弛变量惩罚参数的组合，以寻找更好的模型配置。

4. 扩展应用领域：将手写数字识别应用于更广泛的领域，如自动化表单处理、手写输入识别、签名验证等。这些应用场景可以提高工作效率，减少人工错误，并为用户提供更好的交互体验。

5. 关注最新研究和技术发展：持续关注数据挖掘领域的最新研究和技术发展，学习和应用新的算法和工具。例如，可以关注深度学习领域的最新进展，如自注意力机制、Transformer 模型等，以及相关的预训练模型，如 BERT、GPT 等。

总的来说，通过不断学习和实践，我们可以将数据挖掘的知识和技术应用到更广泛的领域和实际问题中，为社会和业务带来更大的价值。手写数字识别任务只是数据挖掘中的一个示例，我们可以探索更多的任务和应用场景，并不断改进

和创新，以提高模型的性能和应用的效果。