同济大学

TONGJI UNIVERSITY

大作业报告

基于 SVM 的手写数字识别任务

学　　院：电子与信息工程学院

专　　业：数据科学与大数据技术

学生姓名：胡逸凡

学　　号：2153592

2023 年 12 月

目录

1 任务描述

1.1 背景

基本介绍 手写数字识别 (Handwritten Digit Recognition) 是模式识别学科的一个传统研究领域。主要研究如何利用电子计算机自动辨认手写在纸张上的阿拉伯数字。随着信息化的发展，手写数字识别的应用日益广泛，研究高识别率、零误识率和低拒识率的高速识别算法具有重要意义。手写数字识别任务需要克服两大难点：一是识别精度需要达到更高的水平。手写数字识别没有上下文，数据中的每一个数据都至关重要。而数字识别经常涉及金融、财会领域，其严格性更是不言而喻。因此，国内外众多的学者都在为提高手写数字的识别率，降低误识率而努力。二是识别的速度要达到很高的水平。数字识别的输入通常是很大量的数据，而高精度与高速度是相互矛盾的，因此对识别算法提出了更高的要求。

研究意义与理论价值 由于手写数字识别本身的特点，对它的研究有重要的理论价值：

- 阿拉伯数字是唯一被世界各国通用的符号，对手写体数字识别的研究基本上与文化背景无关，各地的研究工作者基于同一平台开展工作，有利于研究的比较和探讨。
- 手写数字识别应用广泛，如邮政编码自动识别，税表系统和银行支票自动处理等。这些工作以前需要大量的手工录入，投入的人力物力较多，劳动强度较大。手写数字识别的研究适应了无纸化办公的需要，能大大提高工作效率。
- 由于数字类别只有 10 个，较其他字符识别率较高，可用于验证新的理论和做深入的分析研究。许多机器学习和模式识别领域的新理论和算法都是先用手写数字识别进行检验，验证理论的有效性，然后才应用到更复杂的领域当中。这方面的典型例子就是上次实验用到的人工神经网络和这次实验用到的支持向量机 (Support Vector Machine)。
- 手写数字的识别方法很容易推广到其它一些相关问题，如对英文之类拼音文字的识别。事实上，很多学者就是把数字和英文字母的识别放在一起研究的。

1.2 任务要求

MNIST 数据集是手写数字识别问题中中最经典的数据集之一，许多模型在该数据集上取得了良好效果。在上次采用 CNN 神经网络对手写数字数据集进行预测的基础上，采用传统机器学习中的 SVM 再次进行分类，对比不同核函数与松弛变量惩罚参数，以达到最高的准确率。

针对手写数字识别的技术难点，我准备了易错测试样例如图。以下是我准备的数据集中一些识别出错的样例，如图 1 所示。

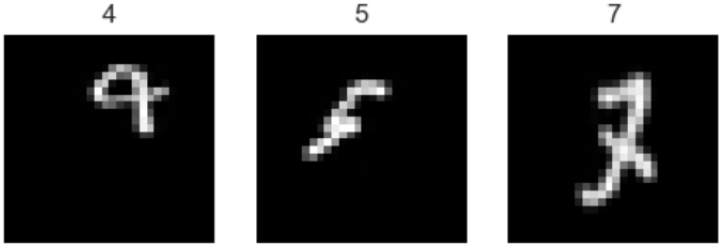


图 1: 易错样例

1.3 相关工作

支持向量机（support vector machines, SVM）是一种分类模型，它的基本模型是定义在特征空间上的间隔最大的线性分类器，间隔最大使它有别于感知机；SVM 还包括核技巧，这使它成为实质上的非线性分类器。SVM 的的学习策略就是间隔最大化，可形式化为一个求解凸二次规划的问题，也等价于正则化的合页损失函数的最小化问题。SVM 的的学习算法就是求解凸二次规划的最优化算法。具体的算法推导和算法应用将在之后的部分详细说明。

本次实验主要将 SVM 用于数字图像识别，通过训练不同核函数与松弛变量惩罚参数的 SVM 模型，从而对手写数字图像进行分类预测。

2 数据集来源

数据集为为经典的 MNIST 数据集，包括 60000 张 28 像素 ×28 像素的灰度图，通过展开得到 784 长度列表的一行数据，数据大小为 (60000, 784)。

经典 MNIST 数据集中部分示例如图 2 所示。

图 2: 经典 MNIST 数据集示例

3 方法

3.1 数据集处理方法

我们需要对经典 MNIST 数据集预处理，直接通过 tensorflow.keras.datasets 导入 mnist 数据集即可。其中共有 60000 条数据，包括 (60000,28,28) 与 (60000,)，我们将其转为 (60000,785) 大小的 ndarray 格式，即可作为训练集与验证集。

3.2 模型结构设计

采用了 scikit-learn 库 sklearn.svm 中的 SVC 函数，以及自己手写实现的 SVM 模型。

scikit-learn 中 SVC 函数 SVC 是 scikit-learn 中对 SVM 的封装 [Pedregosa et al.(2011)]，主要参数有：C、kernel、gamma 等。

```
svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0,
shrinking=True, probability=False, tol=0.001, cache_size=200,
 class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr',
     random_state=None)
```

SVC 基本参数如下 [Chang and Lin(2011)]：

- C: 惩罚参数。对于在边界内的点有惩罚系数 C，C 的取值在 $0-+\infty$ 之间，默认值为 1.0。C 越大代表这个分类器对在边界内的噪声点的容忍度越小，分类准确率高，但是容易过拟合，泛化能力差。所以一般情况下，应该适当减小 C，对在边界范围内的噪声有一定容忍。

- kernel: 核函数，默认是'rbf'，可以是'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'，具体对比如表一所示。

表 1: The Comparison among Different Kernel.

Kernel	公式	适用场景
Linear (线性核函数)	$K(x_1, x_2) = x_1^T x_2$	数据线性可分时，或者特征维度较高 但数据线性相关时，只能解决线性问题
Poly (多项式核函数)	$K(x_1, x_2) = (\gamma \cdot x_1^T x_2 + r)^d$	可以处理一些非线性关系，具有多项式 阶数参数，解决非线性问题和线性问题， 但是偏线性
RBF (高斯核函数)	$K(x_1, x_2) = \exp(-\gamma \|\|x_1 - x_2\|\|^2)$	适用于任意维度的数据，可以捕捉复杂 的非线性关系，解决线性和非线性问题， 偏非线性
Sigmoid (Sigmoid 核函数)	$K(x_1, x_2) = \tanh(\gamma \cdot x_1^T x_2 + r)$	通常用于神经网络等模型
Precomputed (允许用户提供预计算的核矩阵)	用户自定义	当用户有自定义核函数，并且能够 提供预计算的核矩阵时使用

- degree: 多项式核的阶数，默认为 3。对其他核函数不起作用。

- gamma: 核函数系数，只对'rbf','poly','sigmoid' 起作用。默认为'scale'(注意从 0.22 版本，默认从'auto' 改为了'scale')，可以是'scale', 'auto', 或直接输出非负浮点数作为值。

$$gamma = \begin{cases} 1/(n\_features * X.var()), & when \ \ 'scale' \\ 1/n\_features, & when \ \ 'auto' \\ non-negative \ float, & when \ \ float \end{cases}$$

- coef0: 核函数的常数项，默认为 0.0，只对'poly','sigmoid' 有用。

- shrinking: 是否启用启发式收缩方式，默认为 True。启发式收缩方式就是：如果能预知哪些变量对应着支持向量，则只要在这些样本上训练就够了，其他样本可不予考虑，这不影响训练结果，但降低了问题的规模并有助于迅速求解，起到一个加速训练的效果。

- probability: 默认为 False，决定最后是否按概率输出每种可能的概率，需注意最后的预测函数应改为 clf.predict_proba。

- tol: 停止训练的误差精度，默认值为 1e-3。

- cache_size: float, 默认为 200(MB)，kernel cache 的大小。

- class_weight: 默认为 None，给每个类别分别设置不同的惩罚参数 C，如果没有给，则会给所有类别都给 C=1，即前面指出的参数 C。

- verbose: 是否启用详细输出，默认为 Flase

- max_iter: int，默认为-1，表示最大迭代次数，-1 表示不限制

- decision_function_shape: 决定了分类时，是一对多的方式来构建超平面，还是一对一，属于'ovo' 或'ovr'，默认为'ovr'。
  a. 一对多法（one-versus-rest, 简称 1-v-r SVMs）。训练时依次把某个类别的样本归为一类, 其他剩余的样本归为另一类，这样 k 个类别的样本就构造出了 k 个 SVM。分类时将未知样本分类为具有最大分类函数值的类。
  b. 一对一法（one-versus-one, 简称 1-v-1 SVMs）。其做法是在任意两类样本之间设计一个 SVM，因此 k 个类别的样本就需要设计 $\frac{k(k-1)}{2}$ 个 SVM。当对一个未知样本进行分类时，最后得票最多的类别即为该未知样本的类别。Libsvm 中的多类分类就是根据这个方法实现。

- break_ties: bool，默认值为 False(0.22 版本首次引入)。只在 decision_function_shape='ovr' 且类别数量大于 2 的情况下起作用。当 break_ties=True, decision_function_shape='ovr' 且类别数量大于 2 时，predict 函数在处理预测值相等（存在 tie）的情况时，将根据 decision_function 的置信度值来打破 tie。如果 break_ties=False，在存在 tie 的情况下，predict 函数将简单地返回 tie 中的第一个类别。需要注意的是，打破 tie 会相对较高地增加计算成本，因为它需要额外的计算来比较置信度值。

- random_state：默认为 None，在混洗数据时用于概率估计，影响很小。

SVC 常用 attributes 如下 [Platt(1999)]：

- predict: 返回一个数组表示个测试样本的类别。

- predict_probe：返回一个数组表示测试样本属于每种类型的概率。

- decision_function：返回一个数组表示测试样本到对应类型的超平面距离。

- get_params：获取当前 svm 函数的各项参数值。

- score：获取预测结果准确率。

- set_params：设置 SVC 函数的参数。

- clf.n_support_: 各类的支持向量的个数。

- clf.support_：各类的支持向量在训练样本中的索引。

- clf.support_vectors_：全部支持向量。

手写实现 SVM 模型 手写了 Linear 核函数的 SVM 模型，SVM 对超平面的划分模型图如图 1 所示。我采用了合页损失函数，具体实现代码如下：

```
class SVM(object):
    def __init__(self, C=1.0):
        self._support_vectors = None
        self.C = C
        self.W = None # 权重向量，形状=(d,)
        self.b = None # 偏置项
        self.x = None # 训练数据，形状=(n,d)
        self.y = None # 训练标签，形状=(n,)
```

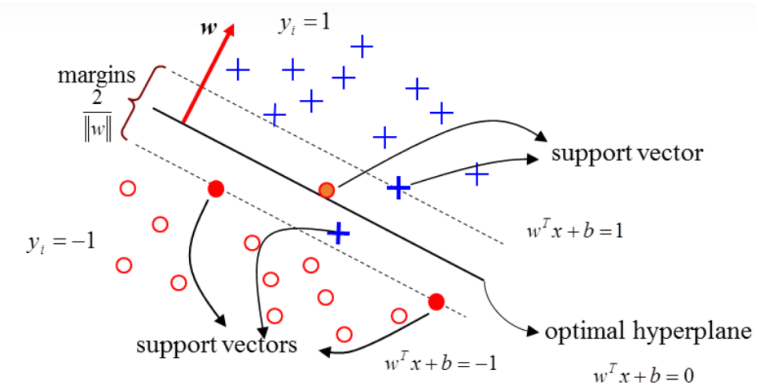


图 3: SVM 超平面划分

```
        self.n = 0 # 样本数
        self.d = 0 # 特征维度

    def __decision_function(self, X): # 决策函数: W * X + b
        return X.dot(self.W) + self.b

    def __margin(self, X, y): # 间隔: y * (W * X + b)
        return y * self.__decision_function(X)

    def __cost(self, margin): # 合页损失函数
        return (1 / 2) * self.W.dot(self.W) + self.C * np.sum(np.maximum(0, 1 - margin))

    def fit(self, X, y, lr=1e-3, epochs=500):
        self.n, self.d = X.shape[0], X.shape[1]
        self.W = np.random.rand(self.d) # 随机初始化权重向量
        self.b = np.random.rand()  # 随机初始化偏置项

        self.x = X
        self.y = y
        losses = []

        for i in range(epochs):
            margin = self.__margin(X, y)
            loss = self.__cost(margin)
            losses.append(loss)

            # 找到误分类点
            missclassified_pts_idx = np.where(margin < 1)[0]

            # 根据误分类点更新权重
            d_W = self.W - self.C * y[missclassified_pts_idx].dot(X[missclassified_pts_idx])
            self.W = self.W - lr * d_W

            # 根据误分类点更新偏置项
            d_b = -self.C * np.sum(y[missclassified_pts_idx])
            self.b = self.b - lr * d_b

```

```
        # 识别支持向量（间隔小于1的点）
        self._support_vectors = np.where(self.__margin(X, y) < 1)[0]

    def predict(self, X):
        # 基于决策函数进行预测
        return np.sign(self.__decision_function(X))

    def score(self, X, y):
        # 计算模型在给定数据集上的准确度
        P = self.predict(X)
        return np.mean(P == y)
```

3.3 模型算法

SVM 的基本模型是定义在特征空间上的间隔最大的线性分类器，间隔最大使它有别于感知机；SVM 还包括核技巧，这使它成为实质上的非线性分类器。SVM 的的学习策略就是间隔最大化，可形式化为一个求解凸二次规划的问题，也等价于正则化的合页损失函数的最小化问题。SVM 的的学习算法就是求解凸二次规划的最优化算法。

SVM 学习的基本想法是求解能够正确划分训练数据集并且几何间隔最大的分离超平面，$\boldsymbol{w}\cdot x+b=0$ 即为分离超平面。对于线性可分数据集和非线性问题的模型算法分别如下：

- 对于线性可分的数据集来说，这样的超平面有无穷多个（即感知机），但是几何间隔最大的分离超平面却是唯一的。
  输入：训练数据集 $T=\{(\mathbf{x_1},y_1),(\mathbf{x_2},y_2),...,(\mathbf{x_n},y_n)\}$，
  其中 $\mathbf{x_i}\in\mathcal{R}^{\mathcal{N}},y_i\in label\{0,1,...,9\},i=1,2,...,n$
  输出：SVM 分离超平面和分类决策函数
  (1) 选择惩罚参数 $C>0$，构造求解凸二次优化问题

$$min_\alpha\ \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j(\mathbf{x_i}\cdot\mathbf{x_j})-\sum_{i=1}^{n}\alpha_i$$
$$s.t.\ \sum_{i=1}^{n}\alpha_i y_i=0,\ 0\le\alpha_i\le C,\ i=1,2,...,n$$

  得到最优解：$\alpha^*=(\alpha_1^*,\alpha_2^*,...,\alpha_n^*)^T$
  (2) 计算更新 SVM 超平面参数

$$\omega^*=\sum_{i=1}^{n}\alpha_i^* y_i\mathbf{x_i}$$
$$b^*=y_j-\sum_{i=1}^{n}n\alpha_i^* y_i(\mathbf{x_i}\cdot\mathbf{x_j})$$

  (3) 得到分离超平面 $\omega^*\cdot\mathbf{x}+b^*=0$ 与分类决策函数 $f(\mathbf{x})=sign(\omega^*\cdot\mathbf{x}+b^*)$

- 对于输入空间中的非线性分类问题，可以通过非线性变换将它转化为某个维特征空间中的线性分类问题，在高维特征空间中学习线性支持向量机。由于在线性支持向量机学习的对偶问题里，目标函数和分类决策函数都只涉及实例和实例之间的内积，所以不需要显式地指定非线性变换，而是用核函数替换当中的内积。核函数表示，通过一个非线性转换后的两个实例间的内积。具体而言，$K(x,z)$ 是一个正定核函数，则存在一个从输入空间到特征空间的映射 $\phi(x)$，对

任意输入空间中 x, z 有 $K(x,z) = \phi(x) \cdot \phi(z)$

由此其算法过程与对线性可分数据集相同，但得到的分类决策函数变为

$$f(\mathbf{x}) = sign(\sum_{i=1}^{n} \alpha_i^* y_i K(\mathbf{x}, \mathbf{x_i}) + b^*)$$

以高斯核函数 RBF 为例，分类决策函数为

$$f(\mathbf{x}) = sign(\sum_{i=1}^{n} \alpha_i^* y_i exp(-\frac{||x - x_i||^2}{2\sigma^2}) + b^*)$$

另外，我的手写实现 SVM 的伪代码如 Algorithm1 中所示，在每一个 epoch 的训练中，我们首先从训练集中得到 $x^{(784)}, y$ 数据，之后计算样本数据点到决策边界的距离，再使用计算得到的距离值通过合页损失函数计算损失，然后根据距离小于 1 的数据点来找出误分类的样本点索引，之后修改权重 W 与偏置 b 使得上一步找出的错误样本点正确分类，最后更新 SVM 模型参数 self.W 与 self.b。

另外我采用 GridSearchCV() 即网格搜索进行超参数调优。

Algorithm 1: The learning algorithm of Hyf's SVM for digit recognition

Input : training set $\mathrm{x}^{(n\times 785)}, y^n$

Parameters: Weight W, bias b, penalty trem C, learning rate η

Output : classification label 0-9

Initialize *learning rate* η as 1e-3;
Initialize *max_epoch* as 500;
Initialize Weight W, bias b randomly;
for *epoch* $= 1$ *to max_epoch* **do**
    Calculate the margin (distance to decision boundary);
    Calculate the loss based on the margin;
    Find misclassified points;
    Update weights based on misclassified points :
        $d_w = W - C \cdot (y_{misclassified} \cdot x_{misclassified})$
        W=W-ηd_w;
    Update bias based on misclassified points :
        $d_b = -C \cdot sum(y_{misclassified})$
        b=b-ηd_b;
    Identify support vectors (points with margin less than 1);
end

4 实验

4.1 实验环境

python version : 3.10.13

pycharm version : PyCharm 2023.2.4 (Professional Edition)

scikit-learn : 1.3.1

4.2 数据集描述

总体数据集大小为由经典 MNIST 数据集得到的 60000×785 大小是数据（28×28=784 个像素值，以及一列 label）。

实验主要分为两步：预训练和微调。首先考虑到运行时间，用经典 MNIST 数据集中 5000 条数据进行模型预训练，对于另外 1000 条数据的预测效果；之后微调，考虑到运行时间因素，训练集中有 20000 条数据时，准确度已经达到较高水平，之后的训练接为 (20000,785)，测试集为 (7000,784)。

另外，我在选取 n 条数据时，采用对 0-9 总共 10 个 label 随机各取 $[\frac{n}{10}]$ 条数据，保证数据分布均匀，选取数据代码如下：

```
def select_data(num, x, y, printed=False):
    selected_x = []
    selected_y = []

    # Iterate over each unique label (0 to 9)
    for label in range(10):

        indices = np.where(y == label)[0]
        np.random.shuffle(indices)
        selected_indices = indices[:num//10]

        # Append the selected data points to the final arrays
        selected_x.append(x[selected_indices])
        selected_y.append(y[selected_indices])
    # Concatenate the selected data points to get the final training set
    selected_x = np.concatenate(selected_x)
    selected_y = np.concatenate(selected_y)
    counts = np.bincount(selected_y)
    if printed:
        for i, count in enumerate(counts):
            print(f"Count of {i} in selected_y_train: {count}")

    return selected_x, selected_y
```

4.3 模型代码片段描述

训练部分，详细内容见注释。

```
def train(kernel, c, train_x, train_y):
    svm_model = svm.SVC(C=c, kernel=kernel, probability=True) # 超参数
    svm_model = OneVsRestClassifier(svm_model)
    svm_model.fit(train_x, train_y)                    # 建模
    return svm_model
```

测试部分，详细内容见注释，在预测后采用准确度、混淆矩阵、分类模型评估报告进行评估。

```
def test(svm_model, test_x, test_y, filename=None):
    # 预测
    predictions = svm_model.predict(test_x)
    # 计算准确度
```

```
    accuracy = accuracy_score(test_y, predictions)*100
    print(f"Accuracy: {accuracy:.6f}%")
    # 混淆矩阵
    draw_confusion_matrix(label_true=test_y,
                          label_pred=predictions,
                          label_name=["0", "1", "2", "3", "4", "5", "6","7","8","9"],
                          title="Confusion Matrix")
    # 分类模型评估报告
    report_dict = classification_report(test_y, predictions, output_dict=True)
    report_df = pd.DataFrame(report_dict).transpose()
    if filename:
        report_df.to_csv(filename, index=False)

    print(report_df)
```

对比不同的核函数与超参数设置。

```
kernel_list = ['sigmoid', 'rbf', 'poly', 'linear']
C = [0.01, 0.1, 1, 10]

train_x, train_y = select_data(5000, x_train.values, y_train)
test_x, test_y = select_data(1000, x_test.values, y_test)

for kernel in kernel_list:
    for c in C:
        print("\nKernel={} C={} start training".format(kernel, c))
        start = time.perf_counter()

        svm_model = train(kernel, c, train_x, train_y)

        end = time.perf_counter()
        print("Kernel={} C={} Training spent {:.6f}s.\n".format(kernel, c, (end - start)))

        test(svm_model, test_x, test_y, 'SVM Kernel={} C={}.csv'.format(kernel, c))
```

网格搜索，超参数调优。

```
# 定义C的候选值
# param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100]} 输出10
# param_grid = {'C': [2, 4, 6, 8, 10, 12, 14]} 输出10
# param_grid = {'C': [9.0, 9.5, 10.0, 10.5, 11.0]} 输出9.5
param_grid = {'C': [9.1, 9.3, 9.5, 9.7, 9.9]} # 输出9.5
kernel = 'poly'
# 创建SVM模型
svm_model = SVC(kernel=kernel, probability=True)
# 使用GridSearchCV进行网格搜索
grid_search = GridSearchCV(svm_model, param_grid, cv=5)
grid_search.fit(x_train, y_train)
# 输出最佳参数
print("Best C:", grid_search.best_params_)
best_c = grid_search.best_params_
```

4.4 模型训练

实验主要分为两步：预训练和微调。首先考虑到运行时间，用经典 MNIST 数据集中 5000 条数据进行模型预训练，对比 4 种 Kernel 函数 (Linear, Poly, RBF, Sigmoid) 分别在惩罚系数 C 在 C = [0.01, 0.1, 1, 10] 总共 4 个不同数量级上，对于另外 1000 条数据的预测效果，得到采用'poly' 核函数和 C 大致范围；之后通过 GridSearchCV() 网格搜索对 C 进行超参数调优，通时，考虑到训练时间因素，我想找到可以达到较高准确度时的最小训练集，绘制预测准确度的随训练集数据条数增加的变化图，得出在训练集有 20000 条数据时，准确度已达到较高水平。

另外，需要注意的一点是，预训练过程中 RBF 核相比 Poly 核准确度稍低，但大概在同一水平，我还对 RBF 核进行了参数微调，在 20000 条数据即较大训练集上时，需要对惩罚参数 C 和 gamma 同时调优，这是因为，RBF 核函数的参数 gamma 定义了单个样本的影响波及范围，gamma 比较小的话，其影响较小；gamma 比较大的话，影响范围较大。可以从以下公式考虑：

$$K(x_1, x_2) = \exp(-\gamma||x_1 - x_2||^2) = \exp(-\frac{||x_1 - x_2||^2}{\epsilon})$$

在 gamma 较大时，正态分布的 ϵ 越小，此时样本分布较为集中，呈现出高瘦的曲线，一个样本对其周围的样本影响自然也就更大了。反之亦然。

对于 SVM 的模型训练需要意识到，在模型准确率与模型复杂度之间取得一个平衡，否则为了追求准确度的稍微提升，复杂度可能会大幅增加，训练耗时大大增加。

4.5 实验结果分析

首先对于 4 种不同的核函数 (Sigmoid,RBF,Poly,Linear) 以及不同数量级的惩罚参数 C(0.01,0.1,1,10) 构建 SVM 模型，对 1000 条数据进行预测，再相比之前的 CNN 模型，得到如下准确度与时间对比如表 2、表 3，最终选择 Ploy 核，再通过 GridSearchCV() 网格搜索对 C 进行超参数调优，得到最优参数值为 9.5。

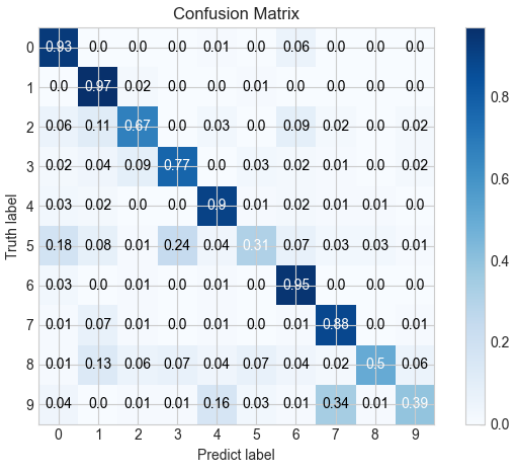
表 2: The Comparison among Accuracy of Different Kernels.

accuracy(%) C / Kernel	0.01	0.1	1.0	10.0
Sigmoid	**72.7**	73.5	67.8	63.2
RBF	84.5	89.8	95.5	**96.4**
Linear	**79.5**	77.7	79.3	79.3
Poly	88.0	92.9	**96.2**	95.5

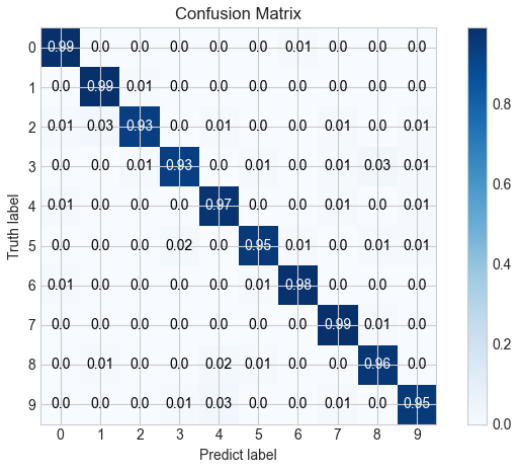
表 3: The Comparison among Training Time of Different Kernel.

time/s C / Kernel	0.01	0.1	1.0	10.0
Sigmoid	92.07	80.25	52.08	**37.36**
RBF	102.93	80.89	51.25	**51.14**
Linear	**164.11**	374.02	518.77	515.97
Poly	85.67	57.98	46.84	**40.41**

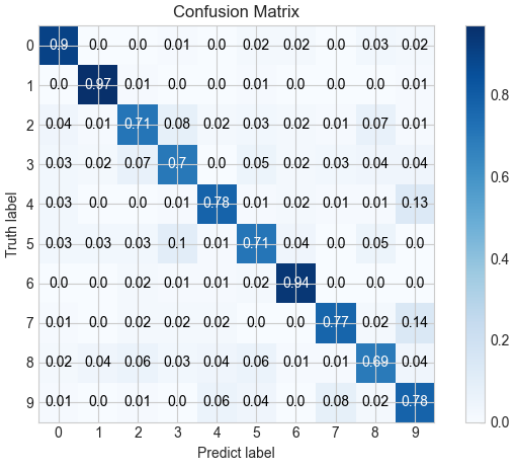
使用每个核函数在准确率最高时的混淆矩阵如图：



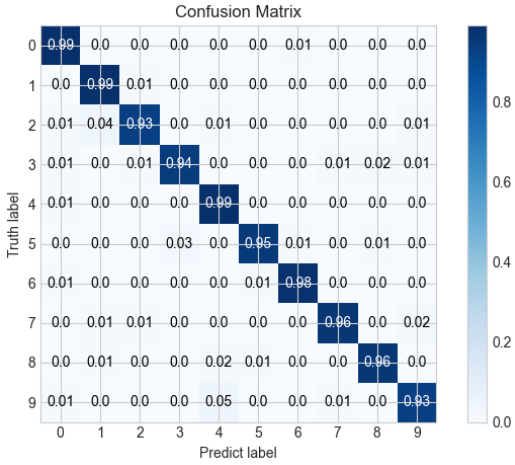
(a) Sigmod.



(b) RBF.



(c) Linear.



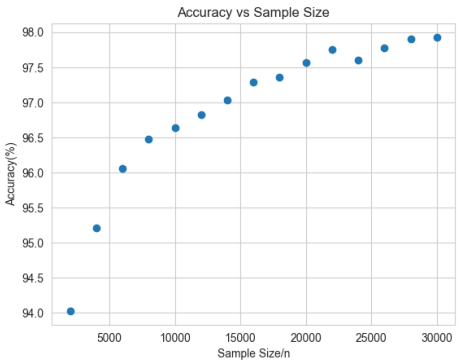
(d) Poly.

图 4: The Comparison among Different Kernel.

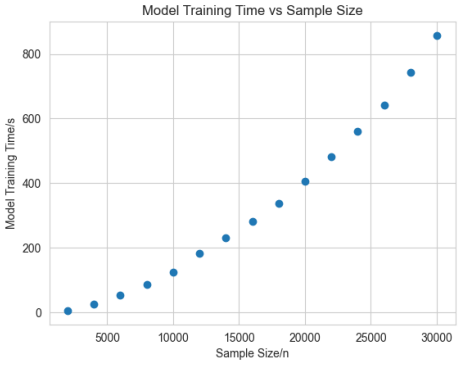
之后，我们知道，SVM 时间复杂度高，训练时间长，为了达到准确率与训练时间的平衡，争取在较高准确度时使用最小训练集，我绘制预测准确度的随训练集数据条数增加的变化图，以及训练时间随训练集数据条数增加的变化图。从图 5(a) 中可以看出在 20000 条数据之前，准确度随数据量增加不断上升，在 20000 条数据之后，准确度随数据量条数有小波动，但始终保持在 97.5% 以上，但是从图 5(b) 看出，随着数据量增加，训练时间呈指数级递增，在数据量为 25000 时，训练时间已经达到 10 分钟以上。最终综合考虑准确度与训练时间，我得出在训练集有 20000 条数据时，准确度已达到较高水平。

综合平衡准确率与时间损耗，使用 Ploy 核函数，惩罚系数 C 为 9.5 时，对 7000 条数据的测试接进行预测，准确度达到 97.91%，在本地电脑上耗时 885.957067s。绘制该模型的混淆矩阵如图 6 所示，分类评估报告如表 4 所示，在分类评估报告中，由于我在测试集数据选择时，人为保证了每一种 label 的数据个数相同，故 support 值均为 700，且 marco avg 与 weight avg 相同，通过观察分类评估报告可以发现，我的 SVM 模型对大部分 label 分类效果较好，对 label 为 5 分类效果最不佳，但准确率也达到 96.89%，是未来改进的方向。另外，相比上一次作业采用 CNN 预测，准确度为 98.36%，效果均较好。

另外，对于书写不规范的数字也有较好的泛化能力。将我准备的易错数据预测结果可视化如图 7 所示。



(a) Accuracy vs Sample Size.



(b) Model Training Time vs Sample Size.

图 5: Change of Sample Size.

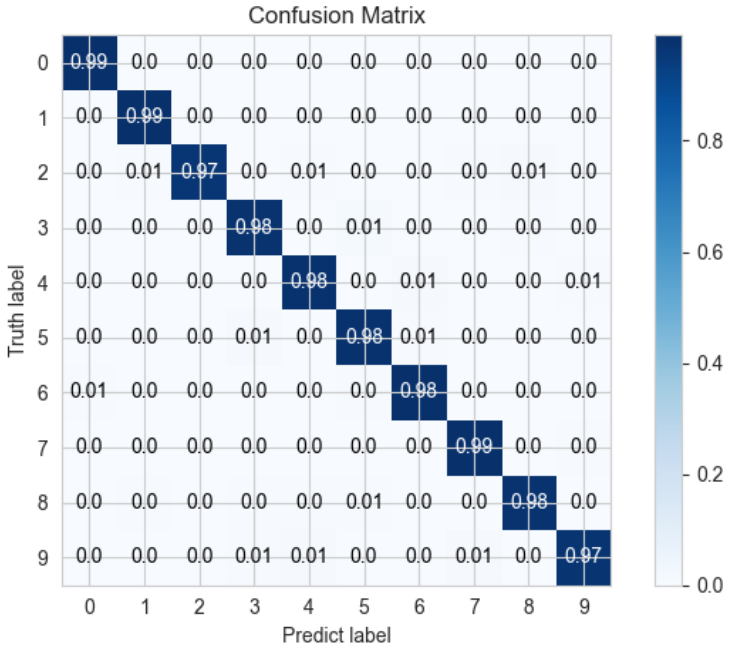


图 6: Confusion Matrix of My Best SVM Model

5 结论和展望

5.1 结论

我的实验首先通过 SVM 模型，采用不同的核函数与惩罚系数，对 MNIST 原始数据集进行训练，在预测集上准确率约为 97.91%。之后，通过可视化可以发现，在扩充数据集后，对于一些比较特殊的数字书写也能有较好的泛化能力，数据集扩充和模型训练有较好的效果。

正如我在报告中写的，通过本次实验，我更加熟悉了经典的 SVM 机器学习算法，在其数学推导、核函数数学应用，以及算法流程与 scikit-learn 库中 SVC 函数的参数，尤其在参数 C 与 gamma 对超平面划分的影响与核技巧方面，我都有很大收获。

5.2 展望

本次手写数字识别是机器学习最基础的应用之一，机器学习具有更强大的性能和更好的前景。[Choudhary et al.(2022)] 在深度学习出现之前，SVM 与集成学习等机器学习算法效果始终出众，如

表 4: Classification Report of My Best SVM Model.

label	precision	recall	f1$_s$*core*	support
0	**0.988571**	0.988571	**0.988571**	700
1	0.982979	**0.990000**	0.986477	700
2	0.986861	0.965714	0.976173	700
3	0.972934	0.975714	0.974322	700
4	0.974395	0.978571	0.976479	700
5	0.968883	0.978571	0.973703	700
6	0.980085	0.984286	0.982181	700
7	0.978723	0.985714	0.982206	700
8	0.975749	0.977143	0.976445	700
9	0.982583	0.967143	0.974802	700
accuracy	0.979143	0.979143	0.979143	0.979143
macro avg	0.979177	0.979143	0.979136	7000
weight avg	0.979177	0.979143	0.979136	7000

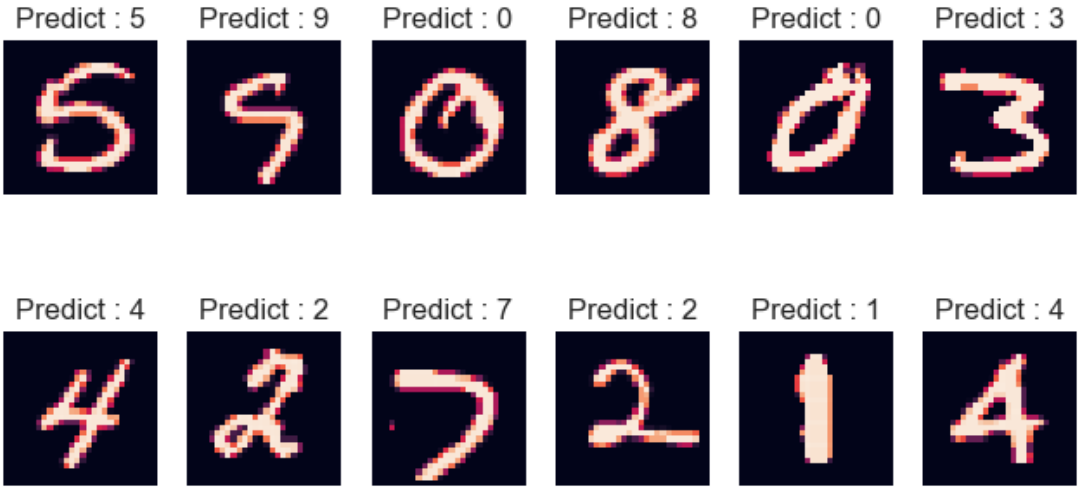


图 7: Visualization of error-prone number prediction results.

今深度学习已经机器学习和人工智能研究的最新趋势之一。它也是当今最流行的科学研究趋势之一。深度学习方法为计算机视觉和机器学习带来了革命性的进步。

如今的深度学习模型有 CNN,RNN,LSTM,Transformer,DRL,GPT 等等，也有在各个领域的各种 sota 模型，深度学习在图像分类与识别，视频分类，序列生成，序列预测，缺陷分类，文本、语音、图像和视频处理，动作识别，歌曲合成等等表现出强大的性能。新的深度学习技术正在不断诞生，超越最先进的机器学习甚至是现有的深度学习技术。随着硬件性能的提升和算法的不断改进，我们可以期待看到更大、更复杂的深度学习模型。新的网络结构和学习算法将进一步提高模型的性能和泛化能力。尽管机器学习（ML）比以往任何时候都更快地推进了世界的发展，但仍有许多方面值得我们去研究。我们仍然无法完全地理解机器学习，我们如何让机器变得更聪明，更接近或比人类更聪明，或者像人类一样学习。ML 一直在解决许多问题，同时将技术应用到方方面面。但是人类仍然面临着许多难题，我们希望机器学习将更加致力于改善人类的生活质量，通过开展最困难的科学研究。愿我们的世界变得更加美好！

参考文献

[Chang and Lin(2011)] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A Library for Support Vector Machines. *ACM Trans. Intell. Syst. Technol.* 2, 3, Article 27 (may 2011), 27 pages. https://doi.org/10.1145/1961189.1961199

[Choudhary et al.(2022)] Kamal Choudhary, Brian DeCost, Chi Chen, Anubhav Jain, Francesca Tavazza, Ryan Cohn, Cheol Woo Park, Alok Choudhary, Ankit Agrawal, Simon J. L. Billinge, Elizabeth Holm, Shyue Ping Ong, and Chris Wolverton. 2022. Recent advances and applications of deep learning methods in materials science. *npj Computational Materials* 8, 1 (apr 2022). https://doi.org/10.1038/s41524-022-00734-6

[Pedregosa et al.(2011)] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[Platt(1999)] John Platt. 1999. Probabilistic Outputs for Support vector Machines and Comparisons to Regularized Likelihood Methods. https://api.semanticscholar.org/CorpusID:56563878