同济大学
计算机科学与技术系

数据挖掘课程实验报告

学　　号　1752919

姓　　名　祁好雨

专　　业　计算机科学与技术

授课老师　向阳

二〇二〇年十二月

一、问题描述

以 MNIST 手写数据集划分训练集和测试集，利用支持向量机模型实现对 MNIST 数据集中手写数字的识别。

二、数据集说明

MNIST 手写数字数据集包含 60000 张用于训练的手写数字图片和 10000 张用于测试的手写数字图片。所有的图片具有相同的尺寸（28x28 pixels），数字均位于图片中央。

数据集链接地址如下：http://yann.lecun.com/exdb/mnist/。

原训练集各类别分布和示例样本如下图所示：

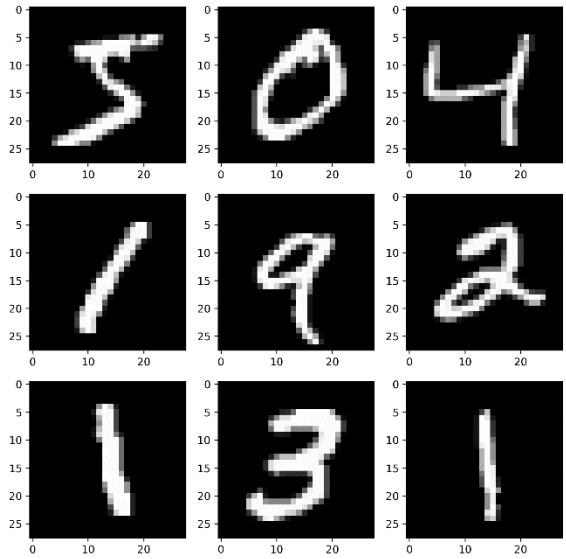

图 1 训练集图片示例

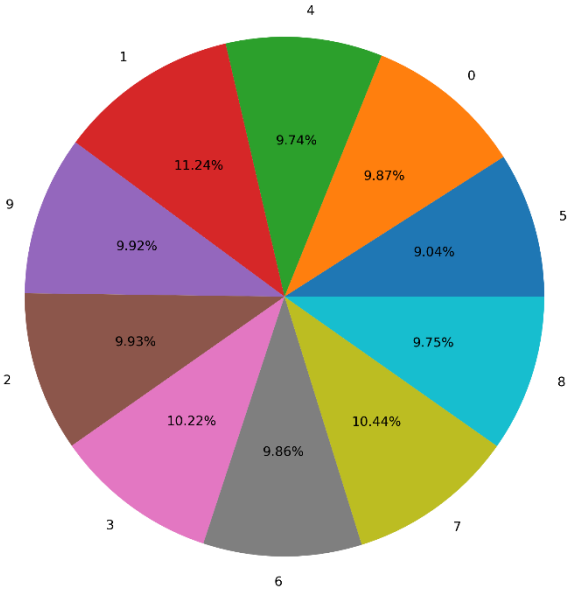

图 2 训练集样本分布

在本实验中，原数据集的测试集作为测试集使用，用于评估模型性能。

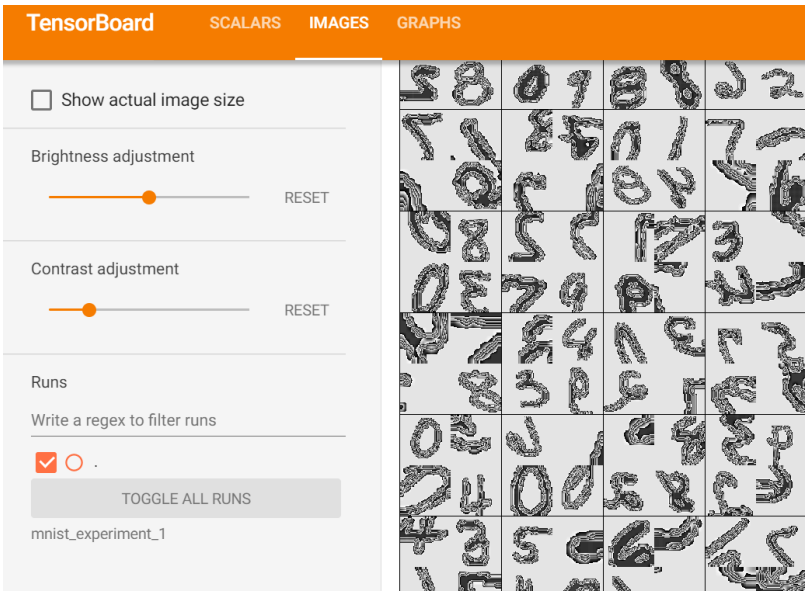

图 3 TensorBoard makegrid 看到的输入图片

三、算法代码

算法概述：

本项目主要使用基于高斯径向基核函数的支持向量机模型来对手写数字样本进行分类，实现利用了 scikit-learn 包中的 SVM.svc 函数，该函数支持的接口如下：

sklearn.svm.SVC

class sklearn.svm.**SVC**(*, C=1.0, kernel='rbf', degree=3, gamma='scale', 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', break_ties=False, random_state=None) [source]

图 4 svm.SVC 函数声明

下面我简单回顾一下支持向量机模型以及核函数的概念来解释一下此函数的部分参数的作用。（支持向量机模型及核函数概念参考自周志华西瓜书[1]）。

如果原样本空间线性可分，则支持向量机模型的目的是找到一个超平面，使得不同类别的样本被这个超平面分开并且不同类别的样本距离最大。

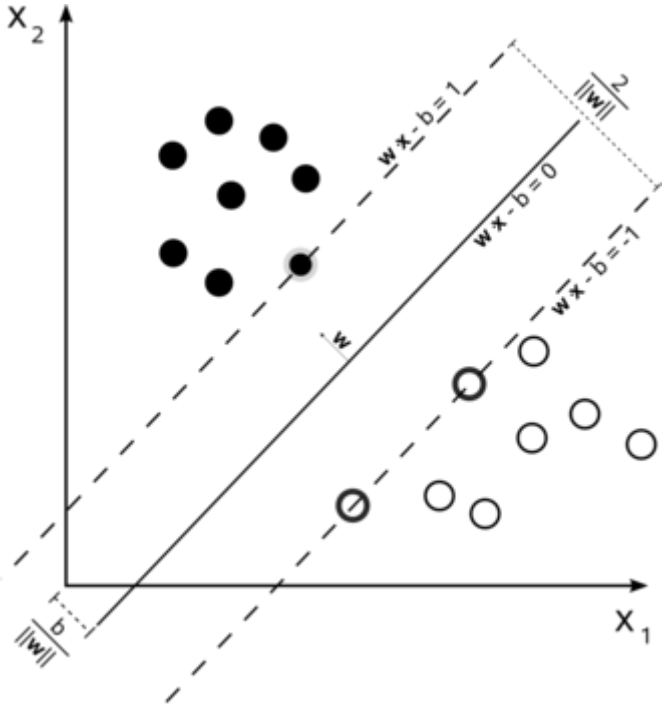

图 5 线性可分的支持向量机模型示意图

如果原样本空间线性不可分，则需要利用数学方法将低维空间的样本映射到高维空间，使之成为线性可分的情况。因此，支持向量机模型也就是一个优化问题，我们可以通过核函数来简化这个优化问题的求解，在本项目中，使用高斯径向基核函数（kernel=‘rbf’）。

$$\min \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\zeta_i$$

$$s.t. \quad y^{(i)}(w^T x^{(i)} + b) \geq 1 - \zeta_i$$

$$\zeta_i \geq 0$$

图 6 优化问题视角下的支持向量机模型

上图的两个限制保证了不同类别(y)的样本点(x)到间隔超平面的距离，其中 C 为软间隔参数，它允许存在一定数量的样本点出现在间隔范围内。数值更小的 C 允许模型拟合出一个简单的划分函数，即使模型的准确率下降；数值更大的 C 则会让模型拟合出一个复杂的划分函数，出现在间隔范围内的样本点更少。它也对应了 svm.SVC 函数中的参数 C。而另一个参数 gamma 则是高斯核函数的参数，它决定了单个样本点对整个模型的影响。

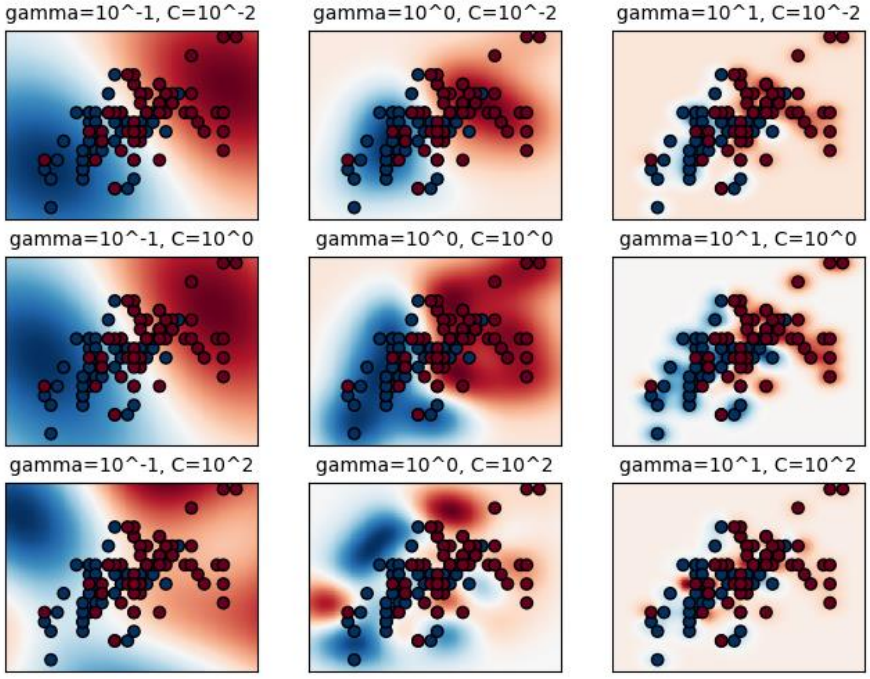

图 7 不同的 gamma 和 C 设置对分类结果的影响[2]

本项目的支持向量机模型中，主要参数设置为 C=1，gamma=‘scale’也就是特征数*方差的倒数，使用高斯径向基核函数。由于支持向量机方法仅支持数据集的维数小于等于 2，因此需要将训练集处理为每一张图片按行主序的方式缩成一维。

使用说明：

由于本项目使用的数据集 MNIST 样本数较多，训练集有 60000 张图片，每张图片的尺寸为[28,28]，处理后训练数据的尺寸为[60000,768]。在 scikit-learn 官方文档中也说明了由于 SVM 方法需要计算点与点之间的最

短距离或者点和点的核函数值，时间复杂度大于 O（n^2），其实不适合用于数据量过大的数据集。在本项目中，为了加速 SVM 的训练，我在预训练阶段将训练集使用 MinMaxScaler 将像素值的范围缩至[0,1]之间；同时，将 SVM.svc 方法使用的 cache 的容量扩大至 10000。（当 cache 不足时，此方法的时间复杂度甚至会超过 O（n^3））。

第三方包安装：

Pytorch
Scikit-learn
Matplotlib

1）导入库

```
[2]  M↓
from sklearn import svm, metrics
import matplotlib.pyplot as plt

[3]  M↓
import torch
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader
import torchvision
```

2）数据集加载

```
[4]  M↓
trainset = torchvision.datasets.MNIST('/home/432/qihaoyu/data/MNIST',train=True,download=False)

[5]  M↓
testset = torchvision.datasets.MNIST('/home/432/qihaoyu/data/MNIST',train=False,download=False)

[14] M↓
clf = svm.SVC(cache_size=10000)

[7]  M↓
trainset.data.shape

torch.Size([60000, 28, 28])

[8]  M↓
trainset.targets.shape

torch.Size([60000])
```

3）数据预处理

```
[9]
n_samples = len(trainset)
X = trainset.data.reshape(n_samples,-1)
Y = trainset.targets
X.shape,Y.shape

(torch.Size([60000, 784]), torch.Size([60000]))

[11]
from sklearn.preprocessing import MinMaxScaler
scaling = MinMaxScaler(feature_range=(0,1)).fit(X)
X_train = scaling.transform(X)
X_test = scaling.transform(testset.data.reshape(len(testset),-1))
```

4）构建分类器及训练

```
[15]
clf.fit(X_train,Y)

SVC(cache_size=10000)

[17]
y_test = clf.predict(X_test)
```

5）模型评估

```
[18]
print("Classification report for classifier %s:\n%s\n"
      % (clf, metrics.classification_report(testset.targets, y_test)))

Classification report for classifier SVC(cache_size=10000):
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.98      0.97      0.98      1032
           3       0.97      0.98      0.98      1010
           4       0.98      0.98      0.98       982
           5       0.99      0.98      0.98       892
           6       0.99      0.99      0.99       958
           7       0.98      0.97      0.97      1028
           8       0.97      0.98      0.97       974
           9       0.97      0.96      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

```
[19]
    print("Confusion matrix:\n%s" % metrics.confusion_matrix(testset.targets, y_test))

Confusion matrix:
[[ 973    0    1    0    0    2    1    1    2    0]
 [   0 1126    3    1    0    1    1    1    2    0]
 [   6    1 1006    2    1    0    2    7    6    1]
 [   0    0    2  994    0    3    0    5    5    1]
 [   0    0    5    0  961    0    3    0    2   11]
 [   2    0    0    9    0  871    4    1    4    1]
 [   6    2    0    0    2    3  944    0    1    0]
 [   0    6   11    1    1    0    0  996    2   11]
 [   3    0    2    6    3    2    2    3  950    3]
 [   3    4    1    7   10    2    1    7    4  970]]
```

四、实验分析

模型在测试集上的评估结果为：

Classification report for classifier SVC(cache_size=10000):

	precision	recall	f1-score	support
0	0.98	0.99	0.99	980
1	0.99	0.99	0.99	1135
2	0.98	0.97	0.98	1032
3	0.97	0.98	0.98	1010
4	0.98	0.98	0.98	982
5	0.99	0.98	0.98	892
6	0.99	0.99	0.99	958
7	0.98	0.97	0.97	1028
8	0.97	0.98	0.97	974
9	0.97	0.96	0.97	1009
accuracy			0.98	10000
macro avg	0.98	0.98	0.98	10000
weighted avg	0.98	0.98	0.98	10000

可以看出模型的性能在各类样本间大致是均衡的，模型也在精准率和召回率的指标中取了平衡，模型的综合准确率为 0.98，与前一次实验神经网络的性能较为接近。模型的最终 confusion matrix 如下图所示：

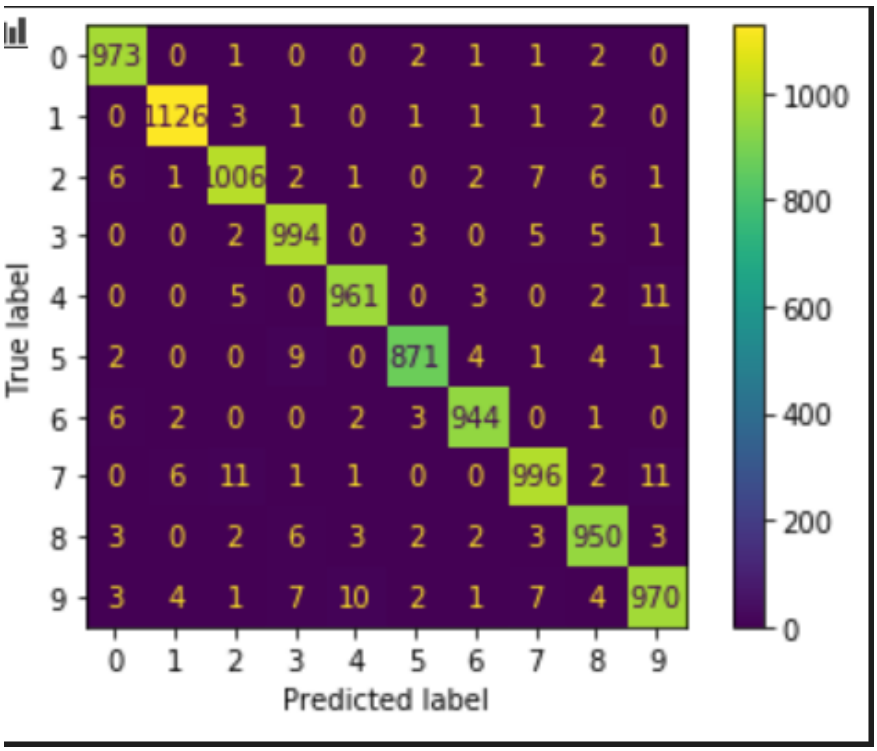

图 8 confusion matrix

从上图可以看出，模型最易混淆的数字对为 4 和 9,2 和 7 对于模型来说也较难判断，与神经网络模型得出的 confusion matrix 结果较为相似。

上传 10 张自己手写的图片，模型判断的结果如下：

```
    y_test_display = clf.predict(X_test[:10,:])
    y_test_display,testset.targets[:10]

(array([7, 2, 1, 0, 4, 1, 4, 9, 6, 9]), tensor([7, 2, 1, 0, 4, 1, 4, 9, 5, 9]))
```

图 9 手写数字识别

参考文献

[1]周志华. 机器学习. 清华大学出版社，2016

[2] https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

[3]维基百科编者. 支持向量机[G/OL]. 维基百科, 2020(20201208)[2020-12-08]. https://zh.wikipedia.org/w/index.php?title=%E6%94%AF%E6%8C%81%E5%90%91%E9%87%8F%E6%9C%BA&oldid=63111341.

附录

完整代码：

```
# To add a new cell, type '# %%'
# To add a new markdown cell, type '# %% [markdown]'
# %%
from sklearn import svm, metrics
import matplotlib.pyplot as plt


# %%
import torch
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader
import torchvision


# %%
trainset = torchvision.datasets.MNIST('/home/432/qihaoyu/data/MNIST',tr
ain=True,download=False)


# %%
testset = torchvision.datasets.MNIST('/home/432/qihaoyu/data/MNIST',tra
in=False,download=False)


# %%
clf = svm.SVC(cache_size=10000)


# %%
trainset.data.shape


# %%
trainset.targets.shape


# %%
n_samples = len(trainset)
X = trainset.data.reshape(n_samples,-1)
Y = trainset.targets
X.shape,Y.shape
```

```
# %%
from sklearn.preprocessing import MinMaxScaler
scaling = MinMaxScaler(feature_range=(0,1)).fit(X)
X_train = scaling.transform(X)
X_test = scaling.transform(testset.data.reshape(len(testset),-1))


# %%
clf.fit(X_train,Y)


# %%
y_test = clf.predict(X_test)


# %%
print("Classification report for classifier %s:\n%s\n"
      % (clf, metrics.classification_report(testset.targets, y_test)))


# %%
print("Confusion matrix:\n%s" % metrics.confusion_matrix(testset.target
s, y_test))


# %%
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(clf, X_test, testset.targets)


# %%
y_test_display = clf.predict(X_test[:10,:])
y_test_display,testset.targets[:10]


# %%
```