统计与数学学院

实验报告

班　　级：应用统计 1701

姓　　名：曹奕涵

学　　号：201720190024

课　　程：非参数统计学

2019~2020 第一学期

使用 LaTeX 撰写于 2020 年 1 月 11 日

摘　要

本实验主要研究了图像分类在 R 中的实现方法以及应用。本实验依赖于 $Fashion-MNIST$ 数据集，探讨了 pooling 降维方法在 R 中的实现以及三种非参数统计方法对图像分类实现的可能性，并在最后给出了三种非参数统计方法的分类正确率。本实验发现，对图像数据处理效果最好的机器学习算法是随机森林算法，在 20000 比 2500 的训练集-测试集划分下，表现出 0.684 的测试集准确率，如果 R 能够处理更大的数据集，那么这个结果将会更好。经过本实验的研究，我发现 R 并不是无法处理图像数据，而是难以处理维度过高的图像数据，只要使用了合适的降维方法以及合适的算法来提升计算性能，在 R 中进行高维图像数据分类并不是一件难事。

关键字：　高维数据处理　图像分类　机器学习

目录

实验一　非参数统计基本介绍

1.1　非参数统计与经典统计

非参数统计学是统计学的一个重要分支。它与经典统计最不相同的地方就是，经典统计主要测算、估计、检验的是模型的参数，而非参数统计研究的则是数据的分布。这也决定了，非参数统计的应用，往往比经典统计要广得多。在摆脱了正态分布的必要条件之后，非参数统计对于数据的要求并不像经典统计那样苛刻，使得更多的数据能够被利用起来。当然，非参数统计也并没有完全脱离经典统计的思想。在古典的非参数统计中，很多对于模型分布检验的思想都来源于经典统计，这两者是相辅相成的。

1.2　非参数统计的基本思想

非参数统计的基本思想就是研究数据的分布。随着计算机的发展，bootstrap 这类的模型得以实现，算法也逐渐落地，使得非参数统计在数据分析中取得了前所未有的进步。在经典的非参数统计中，常常通过假设检验的方式来检测样本是否来自某一分布；直到 20 世纪，机器学习、算法、计算机得到发展，核方法、样条回归的研究开始兴起，此时，非参数统计突破了检验，可以真正地估计出数据的分布。一时间，各种统计学类期刊上被各种各样的 kernel 相关的论文覆盖，直到今天，核估计的各个分支，仍然是统计学家们热衷讨论的内容。

1.3　近年非参数统计的应用与发展

近年来，非参数统计学迎来了重大的突破。自 bootstrap 方法的实现之后，非参数方法的估计结果更加的稳健，也因此被各行各业应用到实践中去。20 世纪末，燕乐存[?] 等人发展了机器学习的方法，将其应用到著名的手写数字识别案例中，从此开始，机器学习的方法开始在各个领域大展身手。机器学习、深度学习的发展，也促进了非参数统计的研究。

近年来，得益于网络爬虫技术的发展，图片与文本这类的高维度数据的获得更加方便，这就要求有更加便捷、精确的处理技术。目前，针对高维数据的分布、推断，已经有了很多有趣的结果。例如，目前有一种非常有意思的方法，能够通过研究已有分布，生成一堆“假样

本”。这是一种被称为是“对抗生成网络”[?] 的方法，其核心思想就是建立一个拟合分布，再建立一个优化器，拟合分布的目标，就是要不断地更新分布，使其更接近真实值。然后，从拟合的分布中，生成一批假样本，让优化器去区分假样本，优化器的目标就是尽量精确地区分假样本与真样本。当循环到优化器已经几乎无法分辨假样本与真样本时，对于该分布的拟合就是完成了。这种方法可以很有效地去逼近样本地分布，比起往常的一步步地检验样本是否服从了某一分布，更加直接。

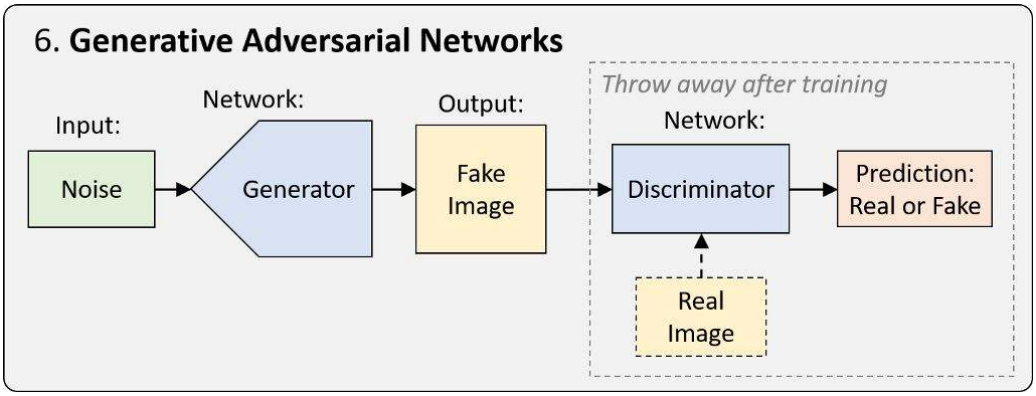

图 1.1: 对抗生成样本

目前，不仅仅是对抗生成这一种方法，还有变分自编码器等等各种各样的方法，运行速度快、效率高、在高维数据上表现良好。可见，非参数统计的方法，在数据科学、大数据等方面有着广泛的应用。

1.4 实验分析

本实验主要探究了高维图像数据在 R 中处理方法。引入了 $Fashion - MNIST$ 数据集，并且综合运用了非参数统计的方法，对交叉样本进行训练、拟合，寻找最优的解决高维图像数据难以处理的方法。

本实验的具体实验步骤如下：

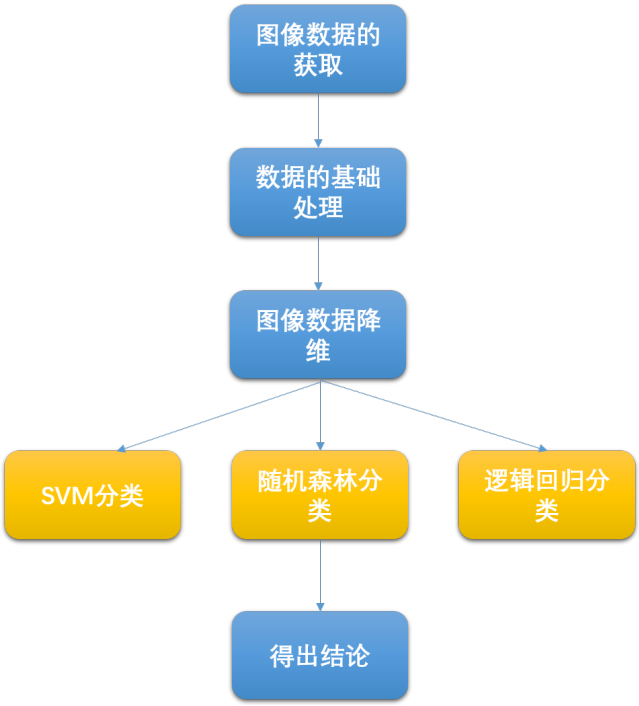

图 1.2: 实验步骤

本实验首先利用 python 对数据进行采集，将图像数据转换成可处理的数值型数据，然后对图像数据进行降维，这样，可以让图像数据在 R 中处理的更快。然后，使用朴素的非参数方法对数据进行拟合，比较各个模型的拟合优度，最后，得出图像数据在 R 中处理的结论。

实验二　数据预处理

2.1　数据介绍

本实验使用的图片数据是著名的 $Fashion-MNIST$ 数据集[?]。$Fashion-MNIST$ 属于常用的深度学习数据集，由 Zalando，一家德国时尚科技公司旗下的研究部门提供，涵盖了来自 10 种类别的共 7 万个不同商品的正面图片。$Fashion-MNIST$ 的大小、格式和训练集、测试集划分与原始的 $MNIST$ 完全一致，包含 60000 条训练集数据以及 10000 条测试集，所有图片的格式都为 28×28 的灰度图片。

$Fashion-MNIST$ 是 $MNIST$ 手写数据集的改进。随着计算机处理数据的能力增强，$MNIST$ 在算法测试上的表现越来越捉襟见肘，正如其他的研究者所说，如果一个算法在 $MNIST$ 上行不通，那它根本没用。如果在 $MNIST$ 上行得通，也不一定有用。$Fashion-MNIST$ 扩展了 $MNIST$ 的数据集包含信息的程度，使得分类更加困难。下图是 $Fashion-MNIST$ 数据集的部分样本图片。

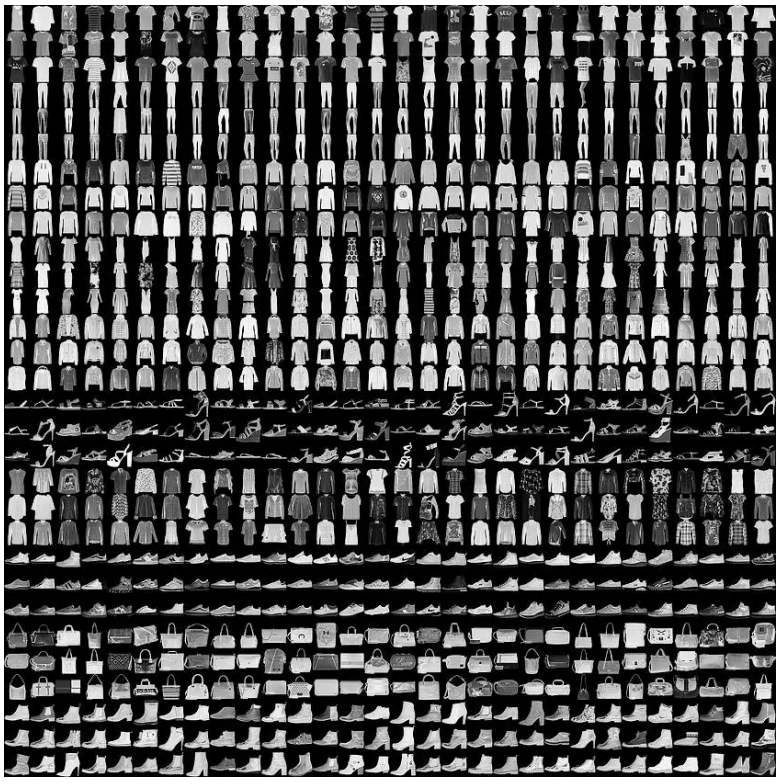

图 2.1: Fashion-MNIST

图像的类别如下表：

表 2.1: Fashion-MNIST 标注解释

标注	描述
0	T-shirt
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	AnkleBoot

其余的更多信息可以参考该数据集作者 Han Xiao 等发表的论文：$Fashion-MNIST:aNovelImageDatasetforBenchmarkingMachineLearningAlgorithms.HanXiao, KashifRasul, Roland$ 1708.07747.

2.2 数据加载与整合

数据的加载与整合通过 pytorch 完成。首先通过 pytorch 下载获得数据集，通过 *opencv* 工具，将图像数据转为 *tensor* 矩阵，使得 R 可以读取。再将数据集从 python 中，通过 *nparray* 的格式输出。

在 pytorch 中，需要首先定义 $get_fashion_mnist$ 函数，将 $Fashion-MNIST$ 的标签更改为文字类型，防止计算机将分类任务误认为回归任务。然后，通过简单的循环函数，将每一张 28×28 的张量矩阵，“拉直为”行向量，这可以通过 pytorch 中的 view 函数实现，并将拉直后的行向量，添加到已经设置好的列表中。这样，就得到了最原始的数据集。最原始的训练集格式为 784×60000，测试集格式为 784×10000。

可以看到，即使是小型图片，图片数据的维度都已经达到了 784 维。虽然高性能的计算机能够处理大型数据，但是为了节省时间，数据降维的处理还是必须的。

此段代码见附件 1。

2.3 维度降低方法及原理

前面说到，为了节省数据处理的时间，需要对数据进行降维。在图像数据处理中，数据降维的方法有很多，像 PCA 这类的方法也可以使用，但是使用的时候就失去了图像处理的意义，也就是说，PCA 进行的降维难以用图片的形式很好地表示出来。这里我们使用卷积神经网络中经常使用的池化方法。

池化的思想非常简单，这里简要介绍一下。就是使用一个固定大小的过滤器，在图片上连续采取最具有代表性的样本。池化具有很好的性质，可以在一定程度上提高空间不变性，也就是说只对图像做了细微的变换，输出的结果仅仅损失了很少的信息。

在我们平时看图片的时候，我们常常注意到的是图像中最明显的部分，也就是图像的主角，而像图像中的边缘部分，比如合照中人物脚边的蚂蚁，可能拍了进去，但是几乎无法发现，像这样的信息实际上就是无效的，因为我们的图像识别只需要最突出的部分。所以，就有了如下的池化算法：

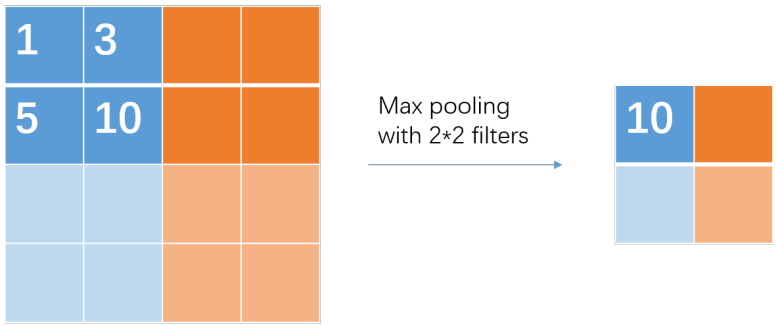

图 2.2: 最大池化算法

这一算法非常简单，图中，将一个 4×4 的图像数据用一个 2×2 的过滤框来选择，看起来像是分为了 4 个部分。在这些过滤器部分中，过滤器每到一个池化区域，就选择这个区域内最大的值，作为这个区域的代表值。这跟我们之前所说的原理非常相近，在矩阵表示的灰度图像中，首先，数据不为 0 的地方说明此处是有颜色的, 如果数值越大，那么也就是说明此处的颜色越明显，那么就越具有代表性。图片在池化之后，实际上就是尺寸变小，变得更模糊了。

当然，池化方法也有求平均的方法，基本的思想与最大化池化一样。本实验中的图片数据为 28×28，所以使用 7×7 的过滤器，最后得到 16 个特征。这两种方法都包含在代码中。具体 R 代码见附录二。

实验三　模型建立与比较分析

由于 R 缺少深度学习环境，所以没有选取深度学习的方法进行测试。一般深度学习的基础分类方法，在该数据集上的测试准确率为 0.86。并且由于 60000 的数据集训练需要消耗时间过长，所以这里将训练集与测试集缩减为：

3.1　SVM 模型

3.1.1　原理介绍

SVM 是一个学界研究了很久的模型，其分类原理就是在两类的边缘之间画一条线，尽量与两边的数据距离都较远，书面的说法就是形成一个特征空间上间隔最大的分类器。SVM 由感知机发展而来，我们知道，在感知机算法中，存在一个可供调整的“核”部分，这个部分也 SVM 保留了下来，也就是核技巧。由于 SVM 是求解凸二次规划问题，不同于其他的机器学习与深度学习算法，SVM 的求解更为简单一点。

当数据难以使用线性分类器进行划分时，一般的线性分类器常常会失效，而 SVM 则能够通过将数据投影到超平面上，然后再超平面中形成一个分类平面，对数据进行有效的分类。

3.1.2　参数调整与效果

由于使用 tune.out 进行调参消耗时间较长，所以直接给出参数 cost 为 1,0.1，10 的测试集与训练集拟合优度：

表 3.1: SVM 图片数据拟合优度

核	gamma	cost	训练集准确度	测试集准确度
线性核	\	1	0.63	0.622
	\	0.1	0.616	0.612
	\	10	0.634	0.626
高斯核	0.5	0.1	0.265	0.261
	0.5	10	0.259	0.255
	1	1	0.201	0.206
	1	10	0.203	0.201

可以看出，拟合优度较高的是使用线性核，损失为 10 的 SVM 方法。由于训练集和测试集都被缩小为 15000、2500 的大小，所以拟合优度可能不如实际的拟合优度。总的来说，SVM 在该数据集的拟合优度已经非常不错了，测试集优度约在 0.625。

3.2 随机森林模型

3.2.1 原理介绍

树是一个离散数学中的概念。决策树对于一个问题，通过提出一个有一个的决策分类问题，根据样本训练其判别的准则，得到答案，再连接到下一个问题。如下图所示：

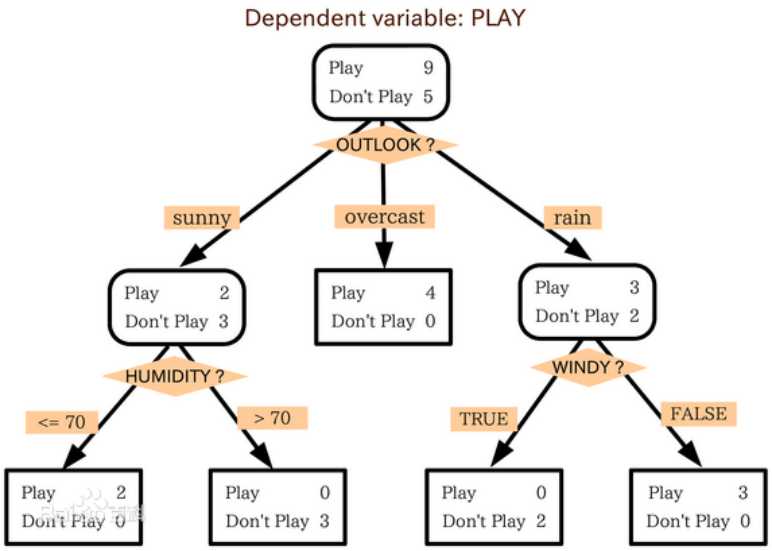

图 3.1: 决策树

决策树整体呈离散数学中定义的树状，遍历整个树的分支，都代表了一个决策点，在每个决策点的分支，都代表了这个决策点的取值。每个分支有着约束条件跟权重。输入数值，根据数值的大小，决策树最后可以给出一个决策值。决策树常常被用来做分类问题，但是也可以用来做回归问题，原理是一样的。

随机森林跟决策树很像，它们本质是差不多的。随机森林是一种集成算法，集成算法通常把很多单独的模型聚合到一起进行训练，最后依据投票机制得出结果。集成学习大体可以分成两类，一类是各个个体学习器之间常常有很强的相互关系，需要串行进行学习。另一类则是个体学习器之间是完全独立的，这样的学习器能够支持并行训练，所以速度也会更快一些。同样的, 随机森林也可以做回归，虽然随机森林的构造简单，但是通常也能获得很好的效果。

3.2.2 模型准确度

模型中可以调整的主要为随机森林中决策树的个数。一般来说，决策树的个数越多，随机森林的性能也越好，对模型中异常点的鲁棒性就越好，模型的泛化能力越强，当然，如果引入的树太多，第一，浪费资源，时间很长，第二，会出现过拟合的问题，泛化能力也会降低。此时，可以选择减少树的数量或者在模型中加入噪音。

表 3.2: 随机森林拟合优度

ntree	训练集准确度	测试集准确度
10	0.973	0.648
50	0.989	0.684
500	0.991	0.683
200	0.99	0.683
1000	0.991	0.681

上表说明，随机森林出现了较大的过拟合的问题，不过拟合的优度还是不错的。当树的个数达到 200 以上时，模型的预测准确率基本稳定在 0.683，从 1000 开始继续下降，最优的参数是树约为 50 个左右，此时过拟合问题稍微缓解，模型的拟合优度达到 0.684，优于 SVM。

3.3 逻辑回归模型

3.3.1 原理介绍

逻辑回归虽然看起来是一种回归算法，但实际上是一种分类算法。逻辑回归通过在线性概率模型中，引入 softmax 函数，让整个曲线呈符合人类认知的“中间概率大、两边概率小”的形式，并且将阈值限定在 0 到 1 的区间内。其中，softmax 函数的表达式为：

$$g(z) = \frac{1}{1 + e^{-z}}$$

这个函数既可以称为 softmax 函数，也可以称为 logic 函数。像这样的函数，常常具有很强的鲁棒性，能够很好地适应数据中的异常点。估计这个模型的方法也很有趣，它虽然是线性模型，但并不适用最小二乘的方法，而是使用最大似然与梯度上升、梯度下降相结合的方法来估计参数，就是在曲线上按照曲线的切线斜率，不停地逼近局部最优点，在一定的循环之后，可以收敛到一个固定值，模型就得到了求解。

3.3.2 模型效果

逻辑回归中没有需要调整的参数，故计算的速度也非常快。其在训练集上的优度为 0.632，在测试集上的优度为 0.6352，训练结果都不错，如果能够引入神经网络的方法，其拟合优度应该还会更高。

3.4 对比分析

三种模型的综合准确率如下：

表 3.3: 综合准确率

模型	训练集准确度	测试集准确度
SVM	0.634	0.625
随机森林	0.989	0.684
逻辑回归	0.632	0.635

由上表可知，随机森林的测试准确率最高，其次是逻辑回归与 SVM。数据集是在经过了 Maxpooling 方法的降维下进行的，这也说明,maxpooling 的方法有效地帮助了训练，减少了训练的时间。在 R 语言中，训练一个高维的模型常常要花大量的时间，这也是 R 语言无法处理图像数据的问题之一，通过 pooling 的方法，可以有效地对图像数据进行降维，从而很快地训练图像数据。但是目前，图像数据的维度越来越高，图片也越来越复杂，像 pooling 这样的方法，就很难适用于 $ImageNet$ 这样大型的图像数据，所以，在 R 中进行深度学习、图像处理，还需要进一步的研究。

实验感想

本实验基于 pooling 降维思想的基础，对 $Fashion-MNIST$ 图像数据进行了处理，讨论了 R 语言处理高维图像数据的方法以及不足之处。由于时间的限制，本实验没有能够完全地利用 60000 条训练集数据，导致训练结果在 0,6-0.7 之间，并不是非常好。实际上，通过神经网络的方法能够很快地提升训练的速度。在本次实验中，我尤其感受到 SVM 算法训练的速度很慢，而逻辑回归的算法运行的非常快，在研究这两者的源码之后，我发现 nnet 包中的逻辑回归算法使用的是梯度下降来估计参数，所以，算法训练的速度很快。这也说明，其实 R 语言的运行速度并不慢，也并不是不适合深度学习以及更深层次的非参数方法的开发，而是由于 R 语言作为统计领域开源的社区，很难获得计算机中 Python、java 或者是商业化的 matlab 这样的支持，很多算法更新不够迅速，方法不全，所以才会导致运行较慢，这对于高维数据的处理也是一个挑战。

通过这次试验，我对 SVM、随机森林、逻辑回归这三种方法的使用以及调参有了更明确的了解，也对 pooling 算法进行了 R 语言上的实现。希望在不久的将来，非参数统计的方法能够在图像领域进一步拓展。

参 考 文 献

[1] Yann LeCun,Leon Bottou,Yoshua Bengio,Patrick Haffner. Gradient-based learning applied to document recognition.Proceedings of IEEE.1998.DOI 10.1109/5.726791.

[2] Ian Goodfellow, Jean Pougetabadie, Mehdi Mirza, Bing Xu, David Wardefarley, Sherjil Ozair, Aaron Courville, Yoshua Bengio.Generative Adversarial Nets.2014 Neural Information Processing Systems.2014.2673-2680.

[3] Han Xiao, Kashif Rasul, Roland Vollgraf,Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. arXiv:1708.07747.

附录 A 导出数据 python 程序代码

```
import torch
import torchvision
from torchvision import transforms

import pandas as pd
import numpy as np

mnist_train = ...
    torchvision.datasets.FashionMNIST(root='¬/Datasets/FashionMNIST', ...
    train=True, download=True, transform=transforms.ToTensor())
mnist_test = ...
    torchvision.datasets.FashionMNIST(root='¬/Datasets/FashionMNIST', ...
    train=False, download=True, transform=transforms.ToTensor())

def get_fashion_mnist_labels(labels):
    text_labels = ['t-shirt', 'trouser', 'pullover', 'dress', 'coat',
                   'sandal', 'shirt', 'sneaker', 'bag', 'ankle boot']
    return [text_labels[int(i)] for i in labels]

samples = []
labels = []
for sample in mnist_train:
    feature,label = sample
    feature = np.array(feature.view(feature.shape[0],-1)[0])
    samples.append(feature)
    label = np.array(label)
    labels.append(label)

test_samples = []
test_labels = []
for sample in mnist_test:
    feature,label = sample
    feature = np.array(feature.view(feature.shape[0],-1)[0])
```

```
30      test_samples.append(feature)
31      label = np.array(label)
32      test_labels.append(label)
33
34  new_samples = np.array(test_samples)
35  data_feature = pd.DataFrame(new_samples)
36  data_feature.to_csv('C:/Users/Administrator/Desktop/test_feature.csv',encoding ...
        = 'gbk')
37
38  new_labels = np.array(test_labels)
39  data_label = pd.DataFrame(new_labels)
40  data_label.to_csv('C:/Users/Administrator/Desktop/test_label.csv',encoding ...
        = 'gbk')
41
42  new_samples = np.array(samples)
43  data_feature = pd.DataFrame(new_samples)
44  data_feature.to_csv('C:/Users/Administrator/Desktop/feature.csv',encoding = ...
        'gbk')
45
46  new_labels = np.array(labels)
47  data_label = pd.DataFrame(new_labels)
48  data_label.to_csv('C:/Users/Administrator/Desktop/train_label.csv',encoding ...
        = 'gbk')
```

附录 B Pooling 程序代码

```
###########Pooling###########
####Attention:Pooling should be a function here!####
pooling <- function(data,poolSize,func){
  in_row = length(data[,1])
  in_col = length(data[1,])

  out_row = in_row/poolSize
  out_col = in_col/poolSize

#max pooling
  temp = rep(1,length = out_row*out_col)
  Resultmap = matrix(temp,out_row,out_col)
  for (i in 1:out_row){
    for (j in 1:out_col){
      startX = poolSize*(i-1)+1
      startY = (j-1)*poolSize+1
      endX = startX+poolSize-1
      endY = startY+poolSize-1
      poolField = data[startX:endX,startY:endY]
      if(func == 'max'){
        poolResult = max(poolField)
      }else{
        poolResult = mean(poolField)
      }
      Resultmap[i,j]=poolResult
    }
  }
  return(Resultmap)
}

vecpooling <- function(data,poolSize){
  n_row = length(data[,1])
  n_col = length(data[1,])/(poolSize*poolSize)
```

```
34    result <- matrix(1,n_row,n_col)
35    for(item in 1:length(data[,1])){
36      single = matrix(data[item,],28,28)
37      new_single = as.vector(pooling(single,poolSize,'max'))
38      result[item,]=new_single
39    }
40    return(result)
41  }
```

附录 C 模型训练程序代码

```
X_train <- ...
    read.csv("C:/Users/Administrator/Desktop/train_feature.csv",header = ...
    FALSE)
y_train <- read.csv("C:/Users/Administrator/Desktop/train_label.csv",header ...
    = FALSE)
X_test <- read.csv("C:/Users/Administrator/Desktop/test_feature.csv",header ...
    = FALSE)
y_test <- read.csv("C:/Users/Administrator/Desktop/test_label.csv",header = ...
    FALSE)

#########训练集数据#########
X_train <- as.matrix(X_train)
x_train <- vecpooling(X_train,7)
x_train <- as.data.frame(x_train)
train_data <- data.frame(x_train[1:15000,],y_train[1:15000,])
colnames(train_data) = ...
    c('V1','V2','V3','V4','V5','V6','V7','V8','V9','V10','V11','V12','V13','V14','V15','
library(e1071)
########参数调整，训练模型#########
svmfit1 = svm(label~.,data=train_data,kernel='linear',cost=1,scale=FALSE)
svmfit2 = svm(label~.,data=train_data,kernel='linear',cost=0.1,scale=FALSE)
svmfit3 = svm(label~.,data=train_data,kernel='linear',cost=10,scale=FALSE)

########查看测试集拟合优度##########
res1 = ...
    table(predict(svmfit1,train_data[1:15000,1:16])==train_data[1:15000,17])
res2 = ...
    table(predict(svmfit2,train_data[1:15000,1:16])==train_data[1:15000,17])
res3 = ...
    table(predict(svmfit3,train_data[1:15000,1:16])==train_data[1:15000,17])

########查看训练集拟合优度#########
X_test <- as.matrix(X_test)
```

```
x_test <- vecpooling(X_test,7)
x_test <- as.data.frame(x_test)
test_data <- data.frame(x_test[1:2500,],y_test[1:2500,])

table(predict(svmfit1,test_data[,1:16])==test_data[,17])
table(predict(svmfit2,test_data[,1:16])==test_data[,17])
table(predict(svmfit3,test_data[,1:16])==test_data[,17])

#########高斯核#######
svmfit4 = svm(label~.,data=train_data,kernel='sigmoid',gamma=0.5,cost=0.1)
svmfit5 = svm(label~.,data=train_data,kernel='sigmoid',gamma=0.5,cost=1)
svmfit6 = svm(label~.,data=train_data,kernel='sigmoid',gamma=0.5,cost=10)
svmfit7 = svm(label~.,data=train_data,kernel='sigmoid',gamma=1,cost=1)
svmfit8 = svm(label~.,data=train_data,kernel='sigmoid',gamma=1,cost=10)

res4 = ...
    table(predict(svmfit4,train_data[1:15000,1:16])==train_data[1:15000,17])
res5 = ...
    table(predict(svmfit5,train_data[1:15000,1:16])==train_data[1:15000,17])
res6 = ...
    table(predict(svmfit6,train_data[1:15000,1:16])==train_data[1:15000,17])
res7 = ...
    table(predict(svmfit7,train_data[1:15000,1:16])==train_data[1:15000,17])
res8 = ...
    table(predict(svmfit8,train_data[1:15000,1:16])==train_data[1:15000,17])


tes1 = table(predict(svmfit4,test_data[,1:16])==test_data[,17])
tes2 = table(predict(svmfit6,test_data[,1:16])==test_data[,17])
tes3 = table(predict(svmfit7,test_data[,1:16])==test_data[,17])
tes4 = table(predict(svmfit8,test_data[,1:16])==test_data[,17])

#Randomforest
library(randomForest)
rfmodel = randomForest(label~.,data=train_data,importance=TRUE)
rfres1 = table(predict(rfmodel,train_data[,1:16])==train_data[,17])
rftes1 = table(predict(rfmodel,test_data[,1:16])==test_data[,17])
```

```
57
58 rfmodel2 = randomForest(label~.,data=train_data,importance=TRUE,ntree=200)
59 rfres2 = table(predict(rfmodel2,train_data[,1:16])==train_data[,17])
60 rftes2 = table(predict(rfmodel2,test_data[,1:16])==test_data[,17])
61
62 rfmodel3 = randomForest(label~.,data=train_data,importance=TRUE,ntree=1000)
63 rfres3 = table(predict(rfmodel3,train_data[,1:16])==train_data[,17])
64 rftes3 = table(predict(rfmodel3,test_data[,1:16])==test_data[,17])
65
66 rfmodel4 = randomForest(label~.,data=train_data,importance=TRUE,ntree=50)
67 rfres4 = table(predict(rfmodel4,train_data[,1:16])==train_data[,17])
68 rftes4 = table(predict(rfmodel4,test_data[,1:16])==test_data[,17])
69
70 rfmodel5 = randomForest(label~.,data=train_data,importance=TRUE,ntree=10)
71 rfres5 = table(predict(rfmodel5,train_data[,1:16])==train_data[,17])
72 rftes5 = table(predict(rfmodel5,test_data[,1:16])==test_data[,17])
73
74 #LogisticRegression
75 library(nnet)
76 model1 <- multinom(label~.,data = train_data)
77 table(predict(model1,train_data[,1:16])==train_data[,17])
78 table(predict(model1,test_data[,1:16])==test_data[,17])
```