基于 MNIST 数据集和 R 的数字识别

摘要 本文基于 MNIST 数据集建立模型以实现对像素值输入的手写数字数据进行识别。主要工作包含对像素图的基本绘图了解,并比较了不同线性模型与标准神经网络模型在识别能力和运算速度的差异。通过预测正确率和运算时间的综合考虑认为经过主成分分析降维后使用二次判别分析建模得到的模型整体性能较为优秀。

引言

计算机视觉是使用计算机及相关设备对生物视觉的一种模拟。它的主要任务是通过对采集的图片或视频进行处理以获得相应场景的三维信息,就像人类和许多其他类生物每天所做的那样。而此次数字识别就是计算机视觉一个应用的例子，具体地说就是通过一定的算法进行预测像素图中的手写数字具体是多少。

此次使用到的数据集是由 NYU 的 Yann LeCun、Google Labs 的 Corinna Cortes 和 Microsoft Research 的 Christopher J.C. Burges 共同合作完成的 MNIST。每个图片样本都经过尺寸大小的调整和对图片的中心化处理。

一、 内容概要

1.1 数据集的介绍

MNIST(THE MINIST DATABASE of handwritten digits)

主要是由手写数字的像素点数据组成的，每个像素点的取值为[0,255]，并且每个样本像素个数为 28*28，即 784 个。以一个具体的例子说明，如下图 1.1

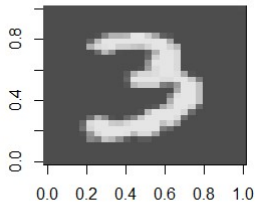

图 1.1 样本像素图示例

1.2 内容介绍

给定训练集（train）其中包含 40,000 条样本信息，其中每种手写数字占总样本的比例约为 10%，训练模型并用于预测另外 2,000 条样本的标识值（label）。

二、 绘制样本图形

我们通过绘制各个数字的样本图像来了解手写数字的一些特点，具体如下图 2.1

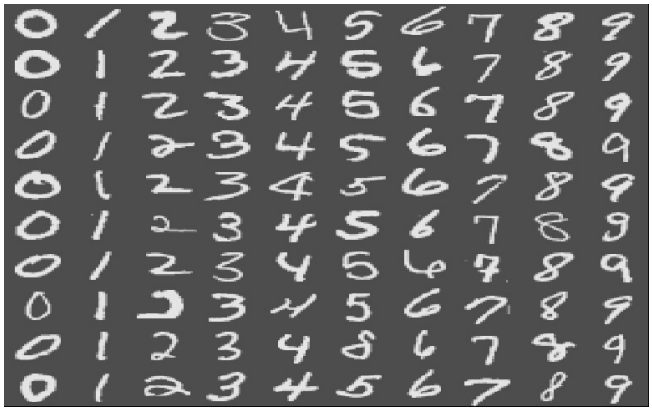

图 2.1 各类数字样本示例

通过图 2.1 我们可以看到，由于是实际的手写数字，因此并不像直接由电脑打印出来的文本数字，各个样本之间存在一定的差异性。但是各个类型的数字之间的区分或者界限还是比较清晰的，这从某种意义上也反映了这个问题的可行性和具体实现的难度。这一点也可以由下面的图 2.2 和图 2.3 来说明

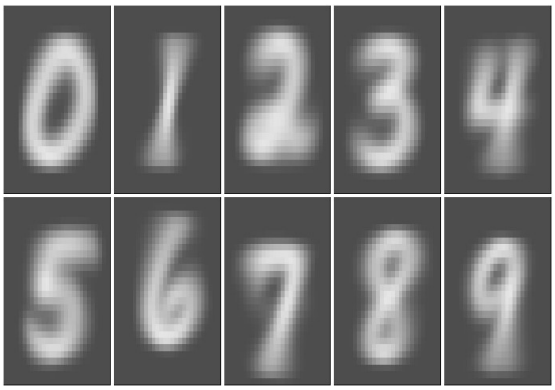

图 2.2 像素均值图

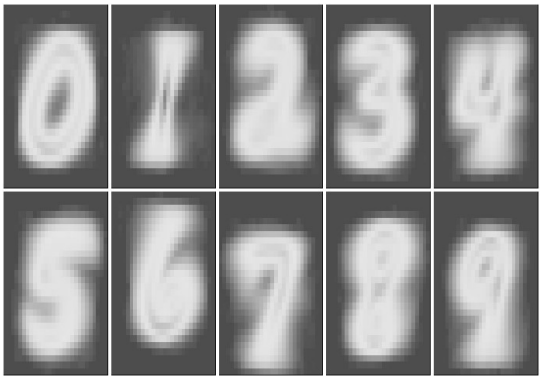

图 2.3 像素方差图

从像素方差图可以看到，各个类别的数字内部的差异性，同时各个数字核心笔画的周围都存在的一定的模糊，这也表示实际手写数字中在这些模糊部分也会出现，只是占总体比例的多少问题，如何正确识别模糊部分的信息也是我们需要考虑的问题。

通过前两部分的阐述，我们可以对此次的数据进行如下几点的概括：

（1） 由于图片本身的特点导致的数据的维度较高和数据量较大

（2） 数据集是由灰度图片得到的，因此存在一定比例的 0，此时数据集也可近似看

作一个稀疏矩阵

（3） 手写数字相对与规范的输出有更多随机的部分，例如 7 中间可能添加一条斜线来标示它具体是 7 还是 1

（4） 部分手写数字 1 可能出现倾斜

通过上面的小结我们在选择模型时需要注意以下的问题：

（1） 选择什么样的模型来实现更高效率的计算和预测？

（2） 是否需要对像素图进行去噪？去噪的比例如何确定？

（3） 若考虑通过降低维度来增加模型的选择范围，那么对于图像数据应该选择怎样的降维方式？

三、 神经网络模型的建立

在不考虑降维的情况下，直接使用原始数据进行建模需要消耗大量的时间，尤其是对于哪些计算复杂度在$o\left(n^2\right)$以上的算法。但是，我们可以尝试在一些开源引擎上运行分布式算法（parallel distributed machine learning algorithms），例如广义线性模型（generalized linear models）、随机森林（Random Forest）和集群环境下的神经网络模型（neural networks (deep learning) within cluster environments）。

本次使用的是在 H20 环境基础上的标准二层神经网络模型，每层的结点设置为 100 个并且 50% dropout ratio，即 50%的隐藏层结点不参与工作。经过十折交叉验证后得到的一个正确率和错误值汇总，如下表 3.1：

表 3.1 十折交叉验证正确率和错误值汇总表

cv_valid	1	2	3	4	5
accuracy	0.9422	0.9465	0.9429	0.9453	0.9423
err	0.0578	0.0535	0.05708	0.05469	0.05768
cv_valid	6	7	8	9	10
accuracy	0.9455	0.9458	0.9405	0.9494	0.9346
err	0.05449	0.05411	0.05942	0.0505	0.0653

最后得到的模型的预测正确率为 94.5%，error 的均值为 0.05647。由于 error 等于偏差、方差和噪声的和，因此 error 越小代表模型的预测能力越强。与此同时，我们也可以看到每一折的 error 都大致在均值附近随机波动，这也说明了这个模型对于预测这个数据集的稳定性较强，即在出现极其偏离训练样本之前，模型对于这类数据具有一定的预测能力，可以在这个模型上对算法进行进一步的优化来扩展它对于其他类型具有偏差数据的预测能力。

四、 线性模型的建立

神经网络模型并未考虑降维，所以可以尝试一些其他的算法进行比较。由于训练集可以看作一个大的稀疏矩阵，同时各类数字内的方差远大于各类数字之间的方差，综合以上两点接下来主要考虑的是先进行数据的去噪和降维，再使用线性模型进行建模。

4.1 剔除噪声

由于像素信息具有一定的冗余，因此首先考虑的是剔除部分噪声，主要使用了两种方法，

它们分别是简单过滤法和奇异值分解。

1. 简单过滤法

首先需要考虑的是保留多少的像素信息，因此我通过一些具体的例子来判断并确定这个比例，如下图 4.1，从左到右依次是保留 100%、75%、50%和 25%：

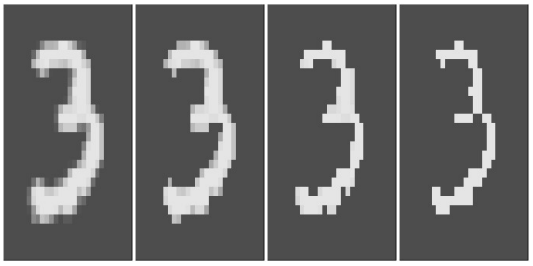

图 4.1 剔除像素信息示例图

当只保留 25%像素信息时数字可能出现缺损，但又要剔除尽可能多的信息，因此综合考虑选择保留 50%的像素信息，并对所有的样本进行相同的处理。对于处理后的样本信息进行输出得到如下图 4.2 所示：

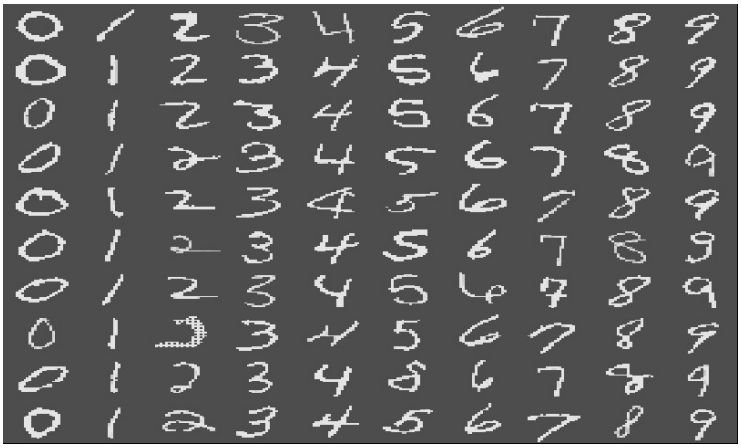

图 4.2 剔除像素信息后各数字样本示例图

通过比较去噪前后的样本方差图来考察去噪的成效，比较去噪前后各类数字的分布是否变更加清晰，如下图 4.3 所示：

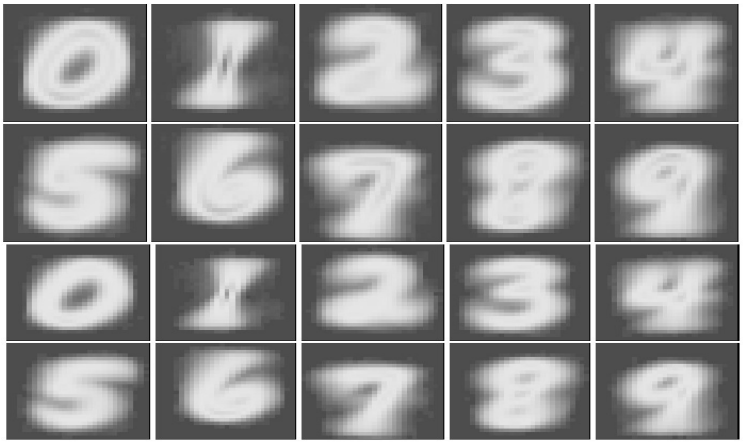

图 4.3 去噪前后方差对比图

可以看到很多数字的棱角开始变得清晰，但是仍然存在一定的模糊，这从另一个方面证明手写数字具有一定范围内“随机”的特点。因此，仅仅通过去噪所带来的对于问题的简化其成效是有限的。

2. 奇异值分解去噪

在简单过滤的基础进行奇异值分解进一步地剔除噪声。相对于特征值分解，奇异值分解可以应用一般的矩阵，可以应用于减噪和降维。其几何意义就是将原有的数据向多个新的正交基进行线性变换，并保留包含多数信息的矩阵来近似原有的矩阵。

这里需要注意的是，一般地，矩阵之间近似程度可以使用 Frobenius 范数来具体衡量，其中矩阵的 Frobenius 范数是矩阵所有元素平方和的平方，将 Frobenius 范数记为 F，则其公式如下(4.1)：

$$F=\sqrt{\sum_{i,j}\left(x_{ij}-y_{ij}\right)}\ \ 1\le i\le m,1\le j\le n\ , \tag{4.1}$$

其中 m,n 表示原矩阵的行、列向量个数，y_{ij} 是 x_{ij} 的近似。

当其为 0 时，两个矩阵严格相等。但是，由于在选择保留多少信息时需要确定 K 值，K 越大，则估计矩阵更接近原矩阵。然而，在 SVD 算法下仅通过 Frobenius 范数来确定 K，K 倾向于尽可能取大，当 K 取太大时则失去了减噪的意义。这也意味着我们需要使用其他的指标来确定 K 值。这里选择的是当两个矩阵单位像素误差减少速度明显减慢时，取该 K 值作为最后的 K 取值，具体公式如下式（4.2）

$$F^{'}=\min_{k}\Delta\left(\frac{\sum_{i,j}\left|x_{ij}-y_{ij}^{(k)}\right|}{m\times n}\right)\ \ 1\le i\le m,1\le j\le n\ \ , \tag{4.2}$$

$y_{ij}^{(k)}$ 表示 $K=k$ 时，x_{ij} 的近似值。

4.2 对去噪后的数据进行降维

由于数据的维度较高，有监督的降维需要消耗大量的时间，因此先尝试使用无监督降维中的主成分分析（Principal Component Analysis, PCA）降维。PCA 的主要思想是用较少的综合变量来代替原来较多的变量，同时要求这几个综合变量尽可能多地反映原来变量的信息，并且彼此之间互不相关。

4.3 再次建立模型

考虑到数据类内方差远小于类间方差，因此考虑使用计算时间相对较少的线性模型。本次使用的模型有线性判别分析、二次判别分析和非线性支持向量机。其中支持向量机尝试了两种核：高斯核和线性核。

五、 模型的评价

对于模型的评价主要从以下两个方面来考虑。首先是预测的正确率，其次是建立模型耗费的时间。具体如下表 5.1：

表 5.1 各模型表现指标汇总表

model	2layers-NN	LDA	QDA	KSVM-Gaussian	KSVM-linear
accuracy	94.50%	86.60%	95.85%	**96.60%**	92.80%
running time(s)	240	22.17	**11.14**	770.08	234.07
feature number	784	58	58	58	58
time in per feat	0.3061224	0.382241	**0.192069**	13.27724138	4.03568966
reduced time	0.00%	9.80%	18.39%	**17.20%**	-1.63%

表 5.1 最后一行表示数据在未进经减噪时直接进行降维和建模所多消耗的时间百分比。

综合考虑 PCA-QDA 是较为理想的方法。虽然 PCA-KSVM（Gaussian）正确率比 PCA-QDA 高 0.75%，但是其运算时间较长，且随着特征数量的增加，其运算时间呈幂级增长。神经网络是目前计算机视觉方向较为流行的算法，其具有的潜力较大。在此基础上进一步增加样本数量，其预测的正确率可以达到 96%及以上，并且其单位特征的运算时间远低于 KSVM 算法。

最后是对这次数字识别实践的一点总结。首先是去噪问题。通过具体的操作发现，去噪本身这个问题可以做非常大的拓展，比如寻找在处理图像时的最佳去噪方式和去噪比例，使算法提高运算速度、预测准确度，同时又不抹去过多的信息导致模型的泛化能力下降。其次是降维，特别是对图像数据，应尽可能地选择特定方法去提取数据的内在结果，而不是基于当下的部分信息去对原始数据做过多的处理。

附录

```
#自定义函数汇总
#--------------------自定义函数 plotDigits()--------------------#
plotDigits <- function(x){
  x <- rev(x)
  m <- matrix(x,nrow = 28)
  m <- apply(m, 2, rev)
  image(m,col = grey.colors(255))
}
#---------------------------------------------------------------#
#------------------自定义函数 plotDigits.eg()-------------------#
plotDigits.eg <- function(x){
  x <- rev(x)
  m <- matrix(x,nrow = 28)
  m <- apply(m, 2, rev)
  image(m,col = grey.colors(255),labels = F,tick = F)
}
#---------------------------------------------------------------#
#--------------自定义函数 plotDigits.output()-------------------#
plotDigits.output <- function(x,file){
  x <- rev(x)
  m <- matrix(x,nrow = 28)
  m <- apply(m, 2, rev)
  png(filename = file,bg = "transparent")
  print(image(m,col = grey.colors(255),labels = F,tick = F))
  dev.off()
}
#---------------------------------------------------------------#
#----------------自定义函数 removeOuter()-----------------------#
removeOuter <- function(x,prob = 0.25)
{
  num <- x[which(x != 0)]
  thres <- quantile(num,probs = 1-prob)
  index <- which(x < thres & x > 0)
  x[index] <- 0
  return(x)
}
#---------------------------------------------------------------#
#------------------自定义函数 accuracy()------------------------#
accuracy <- function(pred,flag)
{
  return(length(which(pred == flag))/length(flag))
}
```

```
#-------------------------------------------------------------#
#------------------自定义函数 recovery()----------------------#
recovery <- function(fac,k,dataset)
{
  dmat <- diag(k)
  diag(dmat) <- fac$d[1:k]
  m <- fac$u[,1:k] %*% dmat %*% t(fac$v[,1:k])
  return(m)
}
#-------------------------------------------------------------#
#------------------自定义函数 svd.setK()----------------------#
svd.setK <- function(fac,k,dataset)
{
  re <- c()
  n <- NCOL(dataset)*NROW(dataset)
  for(i in k) {
    dmat <- diag(i)
    diag(dmat) <- fac$d[1:i]
    m <- fac$u[,1:i] %*% dmat %*% t(fac$v[,1:i])
    re <- c(re,sum(abs(dataset-m))/n)
  }
  return(re)
}
#-------------------------------------------------------------#

#读入数据
raw_dataset <- read.csv("digitRecg_train.csv",header = T,stringsAsFactors = F)
eg <- raw_dataset[10,]
cat(sprintf("数据集中包含的样本个数：%d ",NROW(raw_dataset)))

#将 label 设置成 factor
raw_dataset$label <- as.factor(raw_dataset$label)

#绘制各个数字的样本图形
plotData <- matrix(ncol = NCOL(raw_dataset),nrow = 0)
for(i in 0:9)
{
  plotData <- rbind(plotData,subset(raw_dataset,label == i)[1:10,])
}
plotData <- plotData[,-1]

par(mfcol = c(10,10),mar = rep(0,4))
apply(plotData, 1, plotDigits.eg)
dev.off()
```

```
#绘制均值和方差图
par(mfrow = c(2,5),mar = rep(0.2,4))
for(i in 0:9)
{
  x <- apply(subset(raw_dataset,label == i )[,-1],2,mean)
  plotDigits.eg(x)
}
dev.off()

par(mfrow = c(2,5),mar = rep(0.2,4))
for(i in 0:9)
{
  x <- apply(subset(raw_dataset,label == i)[,-1],2,sd)
  plotDigits.eg(x)
}
dev.off()

#提取 flag
flag <- raw_dataset[,1]

#显示样本中各个数字占的比例
prop.table(table(flag))

#控制 train 和 test 的取值至[0,1]之间，作为 dataset1
dataset1 <- raw_dataset[,-1]
VAR <- apply(dataset1, 2, var)
summary(VAR)

#将 dataset 转换成 0-1 矩阵，作为 dataset2
index <- which(dataset1 > 0,arr.ind = T)
dataset2 <- dataset1
dataset2[index] <- 1

#释放内存空间
rm(i,index,VAR,x)

#Deeplearning applied to all datasets
library(statmod)
library(methods)
library(h2o)
#-------------------自定义函数 deeplearning()--------------------------#
deeplearning_h2o <- function(train,test)
{
```

```
  localH2O = h2o.init(max_mem_size = '2g',
                      nthreads = -1)

  train_h2o <- as.h2o(train)
  test_h2o <- as.h2o(test)

  s <- proc.time()

  model <- h2o.deeplearning(x = 2:785,
                            y = 1,
                            training_frame = train_h2o,
                            activation = "RectifierWithDropout",
                            input_dropout_ratio = 0.2,
                            hidden_dropout_ratios = c(0.5,0.5),
                            balance_classes = TRUE,
                            hidden = c(100,100),
                            momentum_stable = 0.99,
                            nesterov_accelerated_gradient = TRUE,
                            epochs = 15,
                            nfolds = 10,
                            keep_cross_validation_predictions = TRUE,
                            keep_cross_validation_fold_assignment = TRUE)
  print(s - proc.time())
  h2o.confusionMatrix(model)
  h2o_y_test <- h2o.predict(model,test_h2o)

  df_y_test = as.data.frame(h2o_y_test)
  df_y_test = data.frame(ImageId = seq(1,length(df_y_test$predict)),
                         Label = df_y_test$predict)
  return(df_y_test)
}
#-----------------------------------------------------------------------#
#加载数据集
load("D:/R/DR/allDataset.RData")
rm(dataset2,dataset3,dataset4)

#dataset1
dataset1 <- cbind(label = flag,dataset1)
train1 <- dataset1[1:40000,]
test1 <- dataset1[40001:42000,]
set1.dl.pred <- deeplearning_h2o(train1,test1)
accuracy(set1.dl.pred$Label,flag[40001:42000])
#94.5%
```

```
#user  system elapsed
#-1.72   -0.14 -240.44

#err
#[1] "0.05286783"  "0.05116164"  "0.05441504"  "0.05629222"  "0.050616972"
"0.049469966"
#[7] "0.054808423" "0.052056883" "0.053283766" "0.05052995"
#mean err: 0.05255027



#释放空间
rm(dataset1,set1.dl.pred)
rm(train1,test1)

#加载数据集
load("D:/R/DR/allDataset.RData")
rm(dataset1,dataset3,dataset4)

#dataset2
dataset2 <- cbind(label = flag,dataset2)
train2 <- dataset2[1:40000,]
test2 <- dataset2[40001:42000,]
set2.dl.pred <- deeplearning_h2o(train2,test2)
accuracy(set2.dl.pred$Label,flag[40001:42000])
#93.8%

#user  system elapsed
#-2.08   -0.11 -230.32

#释放空间
rm(dataset2,set2.dl.pred)
rm(train2,test2)

#只保留部分像素信息的数据，作为dataset3
#单个样本进行测试
par(mfrow = c(1,4),mar = rep(0.2,4))

plotDigits.eg(as.numeric(eg[-1]))
eg.test <- removeOuter(as.numeric(eg[-1]),0.75)
plotDigits.eg(eg.test)
eg.test <- removeOuter(as.numeric(eg[-1]),0.5)
plotDigits.eg(eg.test)
eg.test <- removeOuter(as.numeric(eg[-1]),0.25)
```

```
plotDigits.eg(eg.test)
dev.off()

#绘制去噪之后各个数字的样本图形
test <- apply(plotData, 1, function(x){removeOuter(x, prob = 0.5)})
test <- t(test)
par(mfcol = c(10,10),mar = rep(0,4))
apply(as.matrix(test), 1, plotDigits.eg)
dev.off()

#应用于原始数据集去噪
dataset3 <- apply(raw_dataset[,-1], 1, function(x){removeOuter(x, prob = 0.5)})
dataset3 <- t(dataset3)
dataset3  <-  as.data.frame(cbind(label  =  as.numeric(as.character(flag)),
dataset3))

#施放部分内存
rm(raw_dataset)

#绘制去噪后的均值和方差图
par(mfrow = c(2,5),mar = rep(0.2,4))
for(i in 0:9)
{
  x <- apply(subset(dataset3,label == i)[,-1],2,mean)
  plotDigits.eg(x)
}
dev.off()

par(mfrow = c(2,5),mar = rep(0.2,4))
for(i in 0:9)
{
  x <- apply(subset(dataset3,label == i)[,-1],2,sd)
  plotDigits.eg(x)
}
dev.off()

#将 dataset3 转换成 0-1 矩阵，作为 dataset4
index <- which(dataset3 > 0,arr.ind = T)
dataset4 <- dataset3
dataset4[index] <- 1

rm(eg.test,test,i,x,index)
rm(plotData)
```

```
#保存相应的数据集（注意修改相应的保存目录）
save.image("D:/R/DR/allDataset.RData")

rm(dataset2,dataset3,dataset4)

#奇异值分解
library(sp)
library(raster)
library(jpeg)

#找到分解后的特征
s <- proc.time()
model.set1.svd <- svd(dataset1)
print(s-proc.time())

#print(s-proc.time())
#用户      系统      流逝
#-149.45    -0.81 -153.69

k <- seq(from = 50, to = 250, by = 50)
Kvect <- svd.setK(model.set1.svd,k,dataset1)

plot(Kvect)
dataset1.svd <- recovery(model.set1.svd,k = 250,dataset1)

save.image("D:/R/DR/set1_svd.RData")

#lib
library(lattice)
library(ggplot2)
library(caret)
library(MASS)
library(kernlab)

#加载数据
load("D:/R/DR/allDataset.RData")
rm(dataset2,dataset3,dataset4)

#PCA 降维
dataset1.cov <- cov(dataset1)
model.set1.pca <- prcomp(dataset1.cov)
summary(model.set1.pca)

#主成分分析结果
```

```
#cumulative proportion of variance
#dataset1: 58 factors can reach 99.5%
dataset1.pca <- as.matrix(dataset1) %*% model.set1.pca$rotation[,1:58]

train.flag <- flag[1:40000]
test.flag <- flag[40001:42000]

train <- dataset1.pca[1:40000,]
test <- dataset1.pca[40001:42000,]

#LDA
s <- proc.time()
lda.control <- trainControl(method = "cv",number = 10)
model.set1.lda <- train(x = train,
                        y = train.flag,
                        method = "lda",
                        trControl = lda.control)
print(s-proc.time())
pred <- predict(model.set1.lda,test)
accuracy(pred,test.flag)

#建模时间和预测的正确率
#用户    系统    流逝
#-23.25  -1.25 -24.58
#86.55%

#QDA
s <- proc.time()
qda.control <- trainControl(method = "cv",number = 10)
model.set1.qda <- train(x = train,
                        y = train.flag,
                        method = "qda",
                        trControl = qda.control)
print(s-proc.time())
pred <- predict(model.set1.lda,test)
accuracy(pred,test.flag)

#建模时间和预测的正确率
#用户    系统    流逝
#-12.53  -1.09 -13.65
#95.8%

#L2 Regularized Support Vector Machine (dual) with Linear Kernel
#高斯核
```

```
s <- proc.time()
model.set1.ksvm <- ksvm(x = train,
                        y = train.flag,
                        kernel = "rbfdot",
                        C = 0.5,
                        cross = 3)
print(s-proc.time())
pred <- predict(model.set1.ksvm,test)
accuracy(pred,test.flag)
#用户     系统     流逝
#-902.39  -26.02 -930.02
#96.6%

#线性核
s <- proc.time()
model.set1.ksvm <- ksvm(x = train,
                        y = train.flag,
                        kernel = "vanilladot",
                        C = 0.5,
                        cross = 3)
print(s-proc.time())
pred <- predict(model.set1.ksvm,test)
accuracy(pred,test.flag)
#用户     系统     流逝
#-227.66   -2.46 -230.32
#92.8%

#加载数据
load("D:/R/DR/allDataset.RData")
rm(dataset1,dataset2,dataset3)

#PCA
dataset4 <- dataset4[,-1]
dataset4.cov <- cov(dataset4)
model.set4.pca <- prcomp(dataset4.cov)
#dataset4: 156 factors can reach 99.5%
dataset4.pca <- as.matrix(dataset4) %*% model.set4.pca$rotation[,1:156]

train.flag <- flag[1:40000]
test.flag <- flag[40001:42000]

train <- dataset4.pca[1:40000,]
test <- dataset4.pca[40001:42000,]
```

```
#LDA
s <- proc.time()
lda.control <- trainControl(method = "cv",number = 10)
model.set4.lda <- train(x = train,
                        y = train.flag,
                        method = "lda",
                        trControl = lda.control)
print(s-proc.time())
pred <- predict(model.set4.lda,test)
accuracy(pred,test.flag)
#用户     系统     流逝
#-141.69   -6.16 -148.31
#85.5%

#QDA
s <- proc.time()
qda.control <- trainControl(method = "cv",number = 10)
model.set4.qda <- train(x = train,
                        y = train.flag,
                        method = "qda",
                        trControl = qda.control)
print(s-proc.time())
pred <- predict(model.set4.qda,test)
accuracy(pred,test.flag)
#用户    系统    流逝
#-55.72  -4.20 -60.05
#93.05%

#高斯核 SVM
s <- proc.time()
model.set4.ksvm <- ksvm(x = train,
                        y = train.flag,
                        kernel = "rbfdot",
                        C = 0.5,
                        cross = 3)
print(s-proc.time())
pred <- predict(model.set4.ksvm,test)
accuracy(pred,test.flag)
#用户      系统      流逝
#-2893.69   -91.76 -2990.46
#94.4%

#线性核 SVM
s <- proc.time()
```

```
model.set4.ksvm <- ksvm(x = train,
                        y = train.flag,
                        kernel = "vanilladot",
                        C = 0.5,
                        cross = 3)
print(s-proc.time())
pred <- predict(model.set4.ksvm,test)
accuracy(pred,test.flag)
#用户      系统      流逝
#-1037.14    -10.07 -1048.44
#91.85%

#加载数据
load("D:/R/DR/set1_svd.RData")

#PCA 降维
dataset1.svd.cov <- cov(dataset1.svd)
model.set1.pca <- prcomp(dataset1.svd.cov)
summary(model.set1.pca)

#主成分分析结果
#cumulative proportion of variance
#dataset1: 58 factors can reach 99.5%
dataset1.svd.pca <- as.matrix(dataset1.svd) %*% model.set1.pca$rotation[,1:58]

train.flag <- flag[1:40000]
test.flag <- flag[40001:42000]

train <- dataset1.svd.pca[1:40000,]
test <- dataset1.svd.pca[40001:42000,]

#LDA
s <- proc.time()
lda.control <- trainControl(method = "cv",number = 10)
model.set1.svd.lda <- train(x = train,
                            y = train.flag,
                            method = "lda",
                            trControl = lda.control)
print(s-proc.time())
pred <- predict(model.set1.svd.lda,test)
accuracy(pred,test.flag)
#用户    系统    流逝
#-20.08   -2.03 -22.17
#86.6%
```

```
#QDA
s <- proc.time()
qda.control <- trainControl(method = "cv",number = 10)
model.set1.svd.qda <- train(x = train,
                            y = train.flag,
                            method = "qda",
                            trControl = qda.control)
print(s-proc.time())
pred <- predict(model.set1.svd.qda,test)
accuracy(pred,test.flag)
#用户    系统    流逝
#-10.37  -1.05 -11.44
#95.85%

#L2 Regularized Support Vector Machine (dual) with Linear Kernel
#高斯核
s <- proc.time()
model.set1.svd.ksvm <- ksvm(x = train,
                            y = train.flag,
                            kernel = "rbfdot",
                            C = 0.5,
                            cross = 3)
print(s-proc.time())
pred <- predict(model.set1.svd.ksvm,test)
accuracy(pred,test.flag)
#用户     系统     流逝
#-757.12  -12.18 -770.08
#96.6%

#线性核
s <- proc.time()
model.set1.svd.ksvm <- ksvm(x = train,
                           y = train.flag,
                           kernel = "vanilladot",
                           C = 0.5,
                           cross = 3)
print(s-proc.time())
pred <- predict(model.set1.svd.ksvm,test)
accuracy(pred,test.flag)
#用户     系统     流逝
#-230.65   -3.21 -234.07
#92.6
#92.8%
```

参考文献

[1] 吴晓婷, 闫德勤. 数据降维方法分析与研究[J]. 计算机应用研究, 2009, 08:2832-2835.
[2] 毕达天, 邱长波, 张晗. 数据降维技术研究现状及其进展[J]. 情报理论与实践, 2013, 02: 125-128.
[3] 王晓慧. 线性判别分析与主成分分析及其相关研究评述[J]. 中山大学研究生学刊(自然科学、医学版), 2007, 04:50-61.
[4] 高惠璇编著. 应用多元统计分析[M]. 北京大学出版社, 2005
[5] 李航. 统计学习方法[M]. 清华大学出版社, 2013
[6] Matloff, N. 著, 陈堰平等译. R 语言编程艺术[M], 机械工业出版社, 2013
[7] Robert I. Kabacoff 著, 王小宁等译. R 语言实战[M], 人民邮电出版社, 2016