# 浙江工业大学20 22 ----20 23 学年第三学期

## 《 数据挖掘与R统计软件课程设计 》报告

姓名：        王佳豪          班级：信计2001       学号：     202003160316

实验一：查找数据挖掘算法的包

adabag

问题(1): 该包有那些函数？

adabag() autoprune() bagging() bagging.cv() boosting() boosting.cv() importanceplot() margins() plot.margins()

问题(2): 该包有那些功能？

bagging 应用装袋算法到数据集 importanceplot 绘制变量的相对重要性 bagging.cv 使用袋装进行v形折叠交叉验证

问题(3): 给出最少3个包功能的具体算例。

**bagging()  predict.bagging()**

应用装袋算法到数据集

library(rpart)  
library(mlbench)  
data(Vehicle)  
l <- length(Vehicle[,1])  
sub <- sample(1:l,2\*l/3) #sample(X,size,replace=T/F)：从数据X中随机抽取size个样本， 当replace=T时为放回取样，为F时为不放回抽样。  
Vehicle.bagging <- bagging(Class ~.,data=Vehicle[sub, ],mfinal=5, control=rpart.control(maxdepth=5, minsplit=15)) # 建立bagging分类模型：formula公式、data变量的数据框、mfinal运行boosting的迭代次数、control控制rpart算法的细节选项  
#Using the pruning option  
Vehicle.bagging.pred <- predict.bagging(Vehicle.bagging,newdata=Vehicle[-sub, ], newmfinal=3) # 预测：object训练的bagging模型、newdata需要进行预测的数据帧、newmfinal要在预测中使用的装袋对象的树数。  
Vehicle.bagging.pred$confusion # 混淆矩阵

## Observed Class  
## Predicted Class bus opel saab van  
## bus 72 15 13 1  
## opel 0 33 23 0  
## saab 1 17 32 3  
## van 2 7 9 54

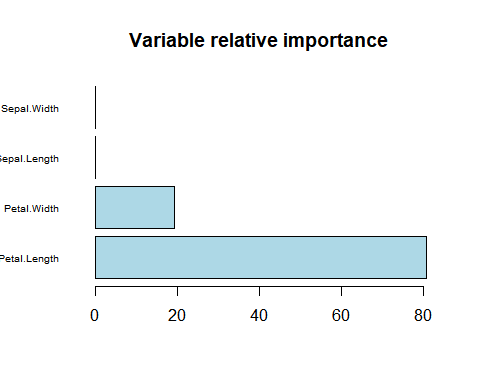
Vehicle.bagging.pred$error # 误差率

## [1] 0.322695

### importanceplot()

绘制分类中每个变量的相对重要性任务。这个这个方法考虑了一棵树中一个变量给出的基尼指数的增益，在提升的情况下，还考虑了这棵树的权重。

library(rpart)  
data(iris)  
sub <-c(sample(1:50, 25), sample(51:100, 25), sample(101:150, 25))  
iris.bagging <- bagging(Species ~ ., data=iris[sub,], mfinal=5) # 同上  
importanceplot(iris.bagging, horiz=TRUE, cex.names=.6) #



### bagging.cv()

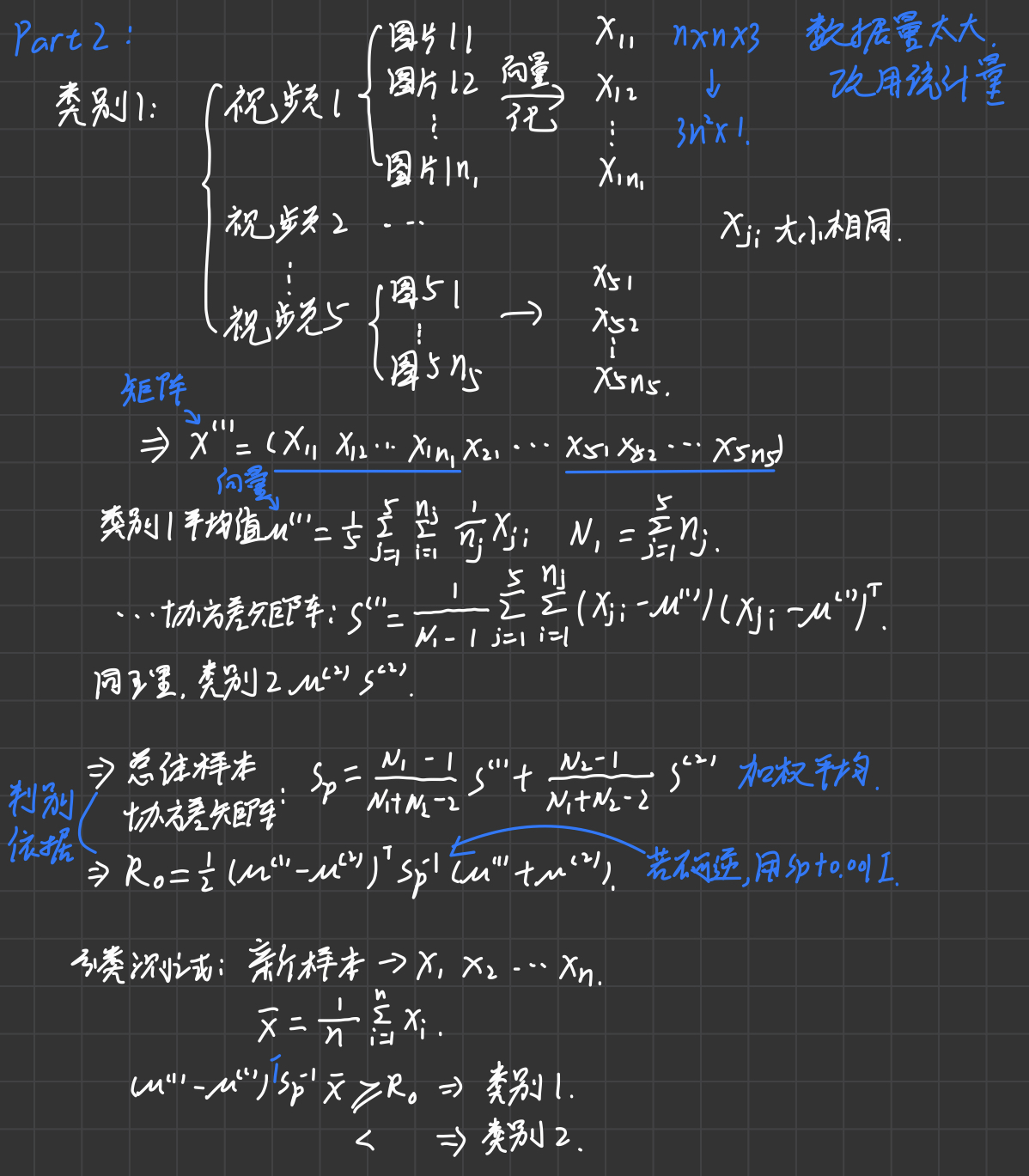
数据被分成大小大致相等的v个不重叠的子集。然后，在子集的（v-1）上应用bagging。最后，对遗漏的子集进行预测，并对每个v子集重复该过程。

library(rpart)  
data(iris) # 鸢尾花数据集：山鸢尾（Setosa）、变色鸢尾（Versicolor）、维吉尼亚鸢尾（Virginical）。花萼长度（Sepal Length）、花萼宽度（Sepal Width）、花瓣长度（Petal Length）、花瓣宽度（Petal Width）四个属性。  
iris.baggingcv <- bagging.cv(Species ~ ., v=2, data=iris, mfinal=3, control=rpart.control(cp=0.01)) # formula模型、data数据、v一个整数，指定v折叠交叉验证的类型、mfinal运行boosting的迭代次数、control控制rpart算法的细节选项  
iris.baggingcv[-1]

## $confusion  
## Observed Class  
## Predicted Class setosa versicolor virginica  
## setosa 50 0 0  
## versicolor 0 46 6  
## virginica 0 4 44  
##   
## $error  
## [1] 0.06666667

实验二：视频数据分类算法研究。

总体思路：



1.视频截图（python3.10）

借助cv2对视频进行循环截图，将图标保存在特定文件夹：”./Image”。

1. def Video\_to\_Image(category):
2. *# 1.存储图片文件夹*
3. path = 'D:/ShortTerm/Data\_Mining/Image'  *# 存放视频图片的主目录*
4. if not os.path.exists(path):  *# 如果不存在就创建文件夹*
5. os.mkdir(path)  *# 如果中间目录不存在，则会报错*
6. *# 2.读取视频文件夹*
7. filepath = 'D:/ShortTerm/Data\_Mining/Video/' + category  *# 需要读取的视频的路径*
8. pathDir = os.listdir(filepath)  *# 获取文件夹中文件名称*
9. *# 3.截视频帧数*
10. for allDir in pathDir:  *# 逐个读取视频文件*
11. *# 存储图像的子目录*
12. Imagepath = r'D:/ShortTerm/Data\_Mining/Image' + '/' + category + '/' + allDir.split('.')[0]
13. if os.path.exists(Imagepath):
14. continue  *# 该视频已截过图，跳过*
15. a = 1  *# 图片计数-不改*
16. c = 1  *# 帧数计数-不改*
17. videopath = filepath + '/' + allDir  *# 视频文件路径*
18. vc = cv2.VideoCapture(videopath)  *# 读入视频文件*
19. if not os.path.exists(Imagepath):  *# 如果不存在就创建文件夹*
20. os.makedirs(Imagepath)  *# 可以一次创建多个目录，甚至是不存在的中间目录，小心使用*
21. if vc.isOpened():
22. rval, frame = vc.read()
23. else:
24. rval = False
25. timeF = 4  *# 帧数间隔*
26. while rval:
27. rval, frame = vc.read()  *# 分帧读取视频*
28. if not rval:
29. break
30. if c % timeF == 1:
31. cv2.imwrite(Imagepath + '/' + str(a) + '.jpg', frame)  *# 保存路径*
32. a = a + 1
33. c = c + 1
34. cv2.waitKey(1)
35. vc.release()

2.截图编码（R4.3.0）

读取一张图片的维度是MxNx3，MxN为图片大小，3为rgb通道。按像素值大小对rgb三色统计像素数量，成1x768(768=256x3)的向量。

1. cal\_hist <- function(img)  # 计算并返回一个图片的颜色统计向量[1:256x3]
2. {
3. img\_hist <- c()
4. for (i in 1:3){
5. img\_vector <- as.vector(round(img[,,i] \* 255))
7. hist <- table(img\_vector)
8. hist\_vector <- as.vector(hist)
9. output\_vector <- rep(0, times=256)
10. output\_vector[match(names(hist), as.character(0:256))] <- hist\_vector
12. img\_hist <- c(img\_hist, output\_vector)
13. }
15. return(img\_hist)
16. }

对一个视频的所有截图编码成向量，将所有向量拼成矩阵，将一个类别的视频编码矩阵存在同一个文件夹，在文件夹”./Video\_Matrix”下。

1. category\_hist <- function(category){
2. *# 存放视频矩阵的列表*
3. video\_matrix\_list <- list()
5. *# 设置读取视频文件夹路径*
6. category\_path <- paste("D:/ShortTerm/Data\_Mining/Image/", category, sep="")
8. *# 获取run文件夹中所有视频文件的目录*
9. video\_list <- list.dirs(path = category\_path, recursive = FALSE)
11. library(jpeg)
12. num\_video = 0
13. for (video in video\_list){
14. num\_video = num\_video + 1
15. *# 读取一个视频文件下的所有图片path*
16. image\_list <- list.files(path = video, pattern = ".jpg", full.names = TRUE)
18. *# 读取第一张图*
19. img <- readJPEG(image\_list[1])
20. video\_hist\_matrix <- cal\_hist(img)
21. *# 循环读取每个jpg图片*
22. for (image in image\_list[2:length(image\_list)]) {
23. *# 读取图片*
24. img <- readJPEG(image)
25. *# 计算图片的颜色统计向量*
26. img\_hist\_vector <- cal\_hist(img)
27. *# 拼接于视频矩阵*
28. video\_hist\_matrix <- rbind(video\_hist\_matrix, img\_hist\_vector)
29. }  *# image cycle end*
31. *# 保存视频矩阵*
32. video\_matrix\_list[[num\_video]] <- video\_hist\_matrix
33. print(paste("video", num\_video, "complete"))
34. }  *#video cycle end*
36. t <- paste(category, "\_matrix\_list", sep="")
37. assign(t, video\_matrix\_list)
38. save(list = t, file = paste("D:/ShortTerm/Data\_Mining/Video\_Matrix/", category, ".Rdata",sep=""))
39. }  *# function end*

3. 构造类别的协方差矩阵和均值向量

按照老师提供的方法，利用一个类别下的所有视频编码矩阵构造类别的协方差矩阵和均值向量。

1. cal\_covariance\_matrix <- function(category)
2. {
3. *## 读取类别数据*
4. category\_path <- paste("D:/ShortTerm/Data\_Mining/Video\_Matrix/", category,
5. ".Rdata", sep="")
6. load(category\_path)
8. *## 计算类别的平均值*
9. list\_name <- paste(category, "\_matrix\_list", sep="")
10. matrix\_list <- get(list\_name)
12. N <- 0  *# 统计图片向量数*
13. mean\_vector <- c()
14. for (i in 1:length(matrix\_list))
15. {
16. video\_matrix <- matrix\_list[[i]]
17. matrix\_dim <- dim(video\_matrix)  *# 获取矩阵维度，[图片数,768]*
19. N <- N + matrix\_dim[1]  *# 增加统计数*
20. if (!is.null(mean\_vector))  *# 更新平均值向量*
21. {
22. mean\_vector <- mean\_vector + colSums(video\_matrix)
23. } else
24. {
25. mean\_vector <- colSums(video\_matrix)
26. }
27. }  *# video\_matrix cycle end*
28. mean\_vector <- mean\_vector / N
30. *## 计算协方差矩阵*
31. cov\_matrix <- NULL  *# 协方差矩阵*
32. for (i in 1:length(matrix\_list))
33. {
34. video\_matrix <- matrix\_list[[i]]
35. matrix\_dim <- dim(video\_matrix)  *# 获取矩阵维度，[图片数,768]*
36. for (j in 1:matrix\_dim[1])
37. {
38. image\_vector <- video\_matrix[j, 1:768]
39. if (!is.null(cov\_matrix))
40. {
41. cov\_matrix <- cov\_matrix + outer(image\_vector - mean\_vector, image\_vector - mean\_vector)
42. } else
43. {
44. cov\_matrix <- outer(image\_vector - mean\_vector, image\_vector - mean\_vector)
45. }
46. }  *# image\_vector cycle end*
47. }  *# video\_matrix cycle end*
48. cov\_matrix <- cov\_matrix / (N - 1)
50. *## 保存数据*
51. t\_N <- paste(category, "\_N", sep="")
52. t\_cov\_matrix <- paste(category, "\_cov", sep="")
53. t\_mean\_vector <- paste(category, "\_mean\_vector", sep="")
54. assign(t\_N, N)
55. assign(t\_cov\_matrix, cov\_matrix)
56. assign(t\_mean\_vector, mean\_vector)
57. save(list = t\_N, file = paste("D:/ShortTerm/Data\_Mining/Video\_Matrix/", category, "\_N.Rdata",sep=""))
58. save(list = t\_cov\_matrix, file = paste("D:/ShortTerm/Data\_Mining/Video\_Matrix/", category, "\_cov.Rdata",sep=""))
59. save(list = t\_mean\_vector, file = paste("D:/ShortTerm/Data\_Mining/Video\_Matrix/", category, "\_mean\_vector.Rdata",sep=""))
60. }

4. 分类

选定两个类别(chew, run)，利用类别的协方差矩阵和均值向量构造R0和Sp，对新视频编码x，并进行分类。

1. classify <- function(category1, category2)
2. {
3. ## 获取数据
4. load(paste("D:/ShortTerm/Data\_Mining/Video\_Matrix/", category1, "\_cov.Rdata", sep=""))
5. load(paste("D:/ShortTerm/Data\_Mining/Video\_Matrix/", category1, "\_N.Rdata", sep=""))
6. load(paste("D:/ShortTerm/Data\_Mining/Video\_Matrix/", category1, "\_mean\_vector.Rdata", sep=""))
8. category1\_name1 <- paste(category1, "\_cov", sep="")
9. category1\_name2 <- paste(category1, "\_N", sep="")
10. category1\_name3 <- paste(category1, "\_mean\_vector", sep="")
11. cate1\_cov <- get(category1\_name1)
12. cate1\_N <- get(category1\_name2)
13. cate1\_mean\_vector <- get(category1\_name3)
15. load(paste("D:/ShortTerm/Data\_Mining/Video\_Matrix/", category2, "\_cov.Rdata", sep=""))
16. load(paste("D:/ShortTerm/Data\_Mining/Video\_Matrix/", category2, "\_N.Rdata", sep=""))
17. load(paste("D:/ShortTerm/Data\_Mining/Video\_Matrix/", category2, "\_mean\_vector.Rdata", sep=""))
19. category2\_name1 <- paste(category2, "\_cov", sep="")
20. category2\_name2 <- paste(category2, "\_N", sep="")
21. category2\_name3 <- paste(category2, "\_mean\_vector", sep="")
22. cate2\_cov <- get(category2\_name1)
23. cate2\_N <- get(category2\_name2)
24. cate2\_mean\_vector <- get(category2\_name3)
26. ## 构造Sp,R0
27. Sp <- cate1\_cov \* (cate1\_N-1)/(cate1\_N+cate2\_N-2) +
28. cate2\_cov \* (cate2\_N-1)/(cate2\_N+cate2\_N-2)
29. Sp\_1 <- solve(Sp + 0.001 \* diag(dim(Sp)[1]))
30. R0 <- t(cate1\_mean\_vector - cate2\_mean\_vector) %\*% Sp\_1 %\*% (cate1\_mean\_vector +cate2\_mean\_vector)
31. R0 <- R0[1]
33. ## 分类
34. load(paste("D:/ShortTerm/Data\_Mining/Video\_Matrix/", category1,
35. ".Rdata", sep=""))
36. matrix\_list1 <- get(paste(category1, "\_matrix\_list", sep=""))
37. load(paste("D:/ShortTerm/Data\_Mining/Video\_Matrix/", category2,
38. ".Rdata", sep=""))
39. matrix\_list2 <- get(paste(category2, "\_matrix\_list", sep=""))
41. sampled\_cate1 <- sample(1:length(matrix\_list1), size=round(length(matrix\_list1)\*0.3), replace=FALSE)
42. sampled\_cate2 <- sample(1:length(matrix\_list2), size=round(length(matrix\_list2)\*0.3), replace=FALSE)
44. TT <- TF <- FT <- FF <- 0
45. for (i in sampled\_cate1)
46. {
47. video\_matrix <- matrix\_list1[[i]]
48. mean\_vector <- colSums(video\_matrix)
49. t <- t(cate1\_mean\_vector - cate2\_mean\_vector) %\*% Sp\_1 %\*% mean\_vector
50. if (t[1] >= R0) {
51. TT <- TT + 1
52. } else {
53. TF <- TF + 1
54. }
55. }  # 类别1分类测试结束
56. for (i in sampled\_cate2)
57. {
58. video\_matrix <- matrix\_list2[[i]]
59. mean\_vector <- colSums(video\_matrix)
60. t <- t(cate1\_mean\_vector - cate2\_mean\_vector) %\*% Sp\_1 %\*% mean\_vector
61. if (t[1] >= R0) {
62. FT <- FT + 1
63. } else {
64. FF <- FF + 1
65. }
66. }  # 类别2分类测试结束
68. ## 打印性能指标
69. print((TT + FF) / (TT + TF + FT + FF))
71. }  # function end

精确率：0.8640777

是在训练集上进行的测试，所以准确率偏高。

实验三

视频数据处理

问题(1): 发现视频的显著特征，比如爱情片中的拥抱镜头，动作片的打斗镜头，喜剧片中的搞笑镜头；

问题(2): 对于较长视频，能够自动统计上述显著特征的数量。

1. 视频编码 python3.10

采用Fisher Vector[1]进行编码，Fisher Vector（FV）是一种用于提取特征表示的方法，常用于图像和视频领域的计算机视觉任务中。它是一种统计模型，能够对数据的分布进行建模和描述。

对视频循环读取图像，提取图像的光流特征，将一个视频的所有光流特征合成矩阵，并进行PCA降维和标准化。

1. *# 提取图像的光流特征*
2. def extract\_features(frame, prev\_gray):
3. *# 将帧转换为灰度图像*
4. gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)
5. if prev\_gray is None:
6. *# 如果是第一帧，则只更新 prev\_gray 并返回空特征*
7. prev\_gray = gray
8. return None, prev\_gray
9. *# 计算光流*
10. flow = cv2.calcOpticalFlowFarneback(prev\_gray, gray, None, 0.5, 3, 15, 3, 5, 1.2, 0)
11. *# 将光流向量2D转换为特征向量1D表示*
12. features = flow.flatten()
13. *# 更新前一个帧*
14. prev\_gray = gray
15. return features, prev\_gray
16. *# 对视频循环读取图像，提取图像的光流特征，所有特征合成矩阵，并进行PCA降维和标准化*
17. def extract\_video\_features(video\_path, dim):
18. *# 循环对视频进行截取*
19. cap = cv2.VideoCapture(video\_path)
20. features = []
21. prev\_gray = None  *# 设置初始的前一帧*
22. while cap.isOpened():
23. ret, frame = cap.read()
24. if not ret:
25. break
26. feature\_vector, prev\_gray = extract\_features(frame, prev\_gray)
27. features.append(feature\_vector)
28. cap.release()
29. *# 消去None*
30. features[:] = [x for x in features if x is not None]
31. *# 将特征进行向量化和规范化*
32. features = np.array(features)
33. *# 进行特征向量的降维，以便通过PCA*
34. if dim - features.shape[0] + 1 > 0:  *# 样本数量不足以进行pca，随机选取行复制*
35. add\_row = dim - features.shape[0] + 1
36. index = sample(list(range(0, features.shape[0])), add\_row)
37. new\_rows = []
38. for i in index:
39. new\_rows.append(features[i, :])
40. new\_rows = np.array(new\_rows)
41. features = np.vstack((features, new\_rows))
42. pca = PCA(n\_components=dim)
43. features = pca.fit\_transform(features)
44. *# 规范化特征向量*
45. features = normalize(features)
46. return features

使用GaussianMixture类来拟合GMM模型，使用拟合好的GMM模型对特征进行编码，得到每个特征向量属于不同GMM分量的概率。接下来，对于每个特征向量的编码，通过计算两部分Fisher向量的值得到最终的Fisher向量表示。第一部分是通过加权求和的方式，计算响应（encoded\_vectors）与GMM分量均值（gmm.means\_）的乘积。第二部分是通过加权求和的方式，计算响应与GMM分量协方差矩阵和均值的平方之和的乘积。最后，将每个帧的Fisher向量编码汇总为整个视频的Fisher向量编码，采用求和的方式。

1. def encode\_fisher\_vectors(features, K):
2. *# 使用高斯混合模型对特征进行编码*
3. gmm = GaussianMixture(n\_components=K, covariance\_type='diag')  *# K为高斯混合模型中的分量数量*
4. gmm.fit(features)
5. encoded\_vectors = gmm.predict\_proba(features)
6. *# 进行Fisher向量编码*
7. fisher\_vectors = []
8. for response in encoded\_vectors:
9. response = response[:, np.newaxis]  *# 给response增加新的维度，将其形状变为（10，1）*
10. fv = np.concatenate([np.sum(response \* gmm.means\_, axis=0),
11. np.sum(response \* (gmm.covariances\_ + gmm.means\_ \*\* 2), axis=0)])
12. fisher\_vectors.append(fv)
13. *# 将所有帧的Fisher向量编码汇总为视频的Fisher向量编码*
14. video\_fv = np.sum(fisher\_vectors, axis=0)
15. *# 可选的后处理，例如规范化Fisher向量编码*
16. video\_fv = normalize(video\_fv.reshape(1,-1))
17. return video\_fv

对一个类别的视频进行循环处理，并保存。

1. def category\_fvcoding(category, K, dim):
2. coding\_path = "./Video\_Coding/" + category  *# 存放类别视频编码的路径*
3. folder\_path = "./Video/" + category  *# 类别视频的文件夹路径*
4. pathDir = os.listdir(folder\_path)  *# 获取文件夹下的视频名称*
5. category\_coding = []  *# 类别编码*
6. num = 0
7. for video\_path in pathDir:
8. num += 1
9. print("Video #" + str(num))
10. features = extract\_video\_features(folder\_path + "/" + video\_path, dim)
11. category\_coding.append(encode\_fisher\_vectors(features, K))
12. category\_coding = np.squeeze(np.array(category\_coding))  *# 多余维度消去*
13. np.save(coding\_path + ".npy", category\_coding)
14. *# npy文件转为csv文件*
15. def npy\_to\_csv(category):
16. data = np.load('./Video\_Coding/' + category + '.npy')
17. with open('./Video\_Coding/' + category + '.csv', 'w', newline='') as csvfile:
18. writer = csv.writer(csvfile)
19. writer.writerows(data)

2. 分类R4.3.0

读取n个csv文件，按ratio划分为训练集和测试集，并生成对应的标签。

1. *# 拆分数据集，n类别数，ratio拆分比例*
2. dataset\_split <- function(n, ratio){
3. *# 获取文件夹下所有 CSV 文件的文件名*
4. csv\_files <- list.files("D:/ShortTerm/Feature\_Extraction/Video\_Coding", pattern = "\\.csv$", full.names = FALSE)
5. csv\_used <- sample(csv\_files, n)
7. # 读取第一个数据集作为起始
8. csv\_file\_path <- paste("D:/ShortTerm/Feature\_Extraction/Video\_Coding/", csv\_used[1], sep="")
9. data <- read.csv(csv\_file\_path, header = FALSE)
11. index <- dim(data)[1] \* ratio
12. train\_data <- data[1:index,]
13. test\_data <- data[index:dim(data)[1],]
14. train\_label <- rep(1, index)
15. test\_label <- rep(1, dim(data)[1] - index + 1)
16. count <- 1
18. for (i in csv\_used[2:length(csv\_used)])
19. {
20. count <- count + 1  # 更新计数
22. temp\_path <- paste("D:/ShortTerm/Feature\_Extraction/Video\_Coding/", i, sep="")
23. temp\_data <- read.csv(temp\_path, header = FALSE)
25. temp\_index <- dim(temp\_data)[1] \* ratio
26. temp\_traindata <- temp\_data[1:temp\_index,]
27. temp\_testdata <- temp\_data[temp\_index:dim(temp\_data)[1],]
28. temp\_trainlabel <- rep(count, temp\_index)
29. temp\_testlabel <- rep(count, dim(temp\_data)[1] - temp\_index + 1)
31. train\_data <- rbind(train\_data, temp\_traindata)
32. test\_data <- rbind(test\_data, temp\_testdata)
33. train\_label <- c(train\_label, temp\_trainlabel)
34. test\_label <- c(test\_label, temp\_testlabel)
36. }  #
38. return(list(dataset1 = train\_data, dataset2 = test\_data,
39. dataset3 = train\_label, dataset4 = test\_label))
40. }  # end function

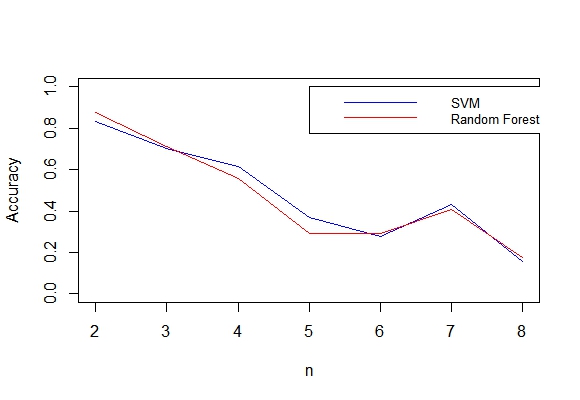
在训练集上训练分类器支持向量机和随机森林，并在测试集上验证，得出准确率。

1. classify <- function(n, ratio)
2. {
3. result <- dataset\_split(n, ratio)
4. train\_data <- result$dataset1
5. test\_data <- result$dataset2
6. train\_label <- result$dataset3
7. test\_label <- result$dataset4
9. *# 训练SVM模型*
10. svm\_model <- svm(train\_label ~ ., data = train\_data, kernel = "linear")
11. *# 进行预测*
12. predictions <- predict(svm\_model, test\_data)
13. predictions <- as.vector(round(predictions))
14. predictions <- ifelse(predictions < 1, 1, ifelse(predictions > n, n, predictions))
15. *# 计算准确率*
16. Accuracy\_svm <- confusionMatrix(factor(predictions), factor(test\_label))$overall["Accuracy"]
18. *# 构建随机森林模型*
19. rf\_model <- suppressWarnings(randomForest(x=train\_data, y=train\_label, ntree = 100))
20. *# 在测试集上进行预测*
21. predicted <- predict(rf\_model, newdata = test\_data)
22. predicted <- as.vector(round(predicted))
23. predicted <- ifelse(predictions < 1, 1, ifelse(predicted > n, n, predicted))
24. *# 计算准确率*
25. Accuracy\_rf <- confusionMatrix(factor(predicted), factor(test\_label))$overall["Accuracy"]
27. return( list( acc1 = as.numeric(Accuracy\_svm), acc2 = as.numeric(Accuracy\_rf) ) )
28. }

在二分类时，准确率分别是：SVM83.3%，RandomForest87.9%。

进行多分类的测试：从2分类到8分类，每轮测试10次，选取最大值。

1. ratio <- 0.7
2. acc\_svm\_list <- c()
3. acc\_rf\_list <- c()
4. for (i in 2:8)
5. {
6. acc\_svm\_best <- 0
7. acc\_rf\_best <- 0
8. for (j in 1:10)
9. {
10. temp <- classify(i, ratio)
11. acc\_svm <- temp$acc1
12. acc\_rf <- temp$acc2
14. if ((acc\_svm + acc\_rf) > (acc\_svm\_best + acc\_rf\_best))
15. {
16. acc\_svm\_best <- acc\_svm
17. acc\_rf\_best <- acc\_rf
18. }
19. }
21. acc\_svm\_list <- append(acc\_svm\_list, acc\_svm\_best)
22. acc\_rf\_list <- append(acc\_rf\_list, acc\_rf\_best)
23. }
24. *# 绘制第一条折线*
25. plot(1:length(acc\_svm\_list)+1, acc\_svm\_list, type = "l", ylim = c(0, 1), col = "blue", xlab = "n", ylab = "Accuracy")
26. *# 添加第二条折线*
27. lines(1:length(acc\_svm\_list)+1, acc\_rf\_list, type = "l", col = "red")
28. *# 添加图例*
29. legend(x = 5, y = 1.0, legend = c("SVM", "Random Forest"), col = c("blue", "red"), lty = 1, ncol = 1, cex = 0.8)



参考文献：

[1] Peng X, Zou C, Qiao Y, et al. Action recognition with stacked fisher vectors[C]//Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13. Springer International Publishing, 2014: 581-595.